Hindawi Publishing Corporation
International Journal of Distributed Sensor Networks
Volume 2015, Article ID 454159, 8 pages
http://dx.doi.org/10.1155/2015/454159

Research Article

A Multicontroller Load Balancing Approach in Software-Defined Wireless Networks

Haipeng Yao,[1] Chao Qiu,[1] Chenglin Zhao,[1] and Lei Shi[2]

[1] *State Key Lab of Networking and Switching Technology, Beijing University of Posts and Telecommunications, No. 10 Xitucheng Road, Beijing 100876, China*
[2] *China Academy of Electronics and Information Technology, No. 10 Xitucheng Road, Beijing 100876, China*

Correspondence should be addressed to Haipeng Yao; yaohaipeng@bupt.edu.cn

Received 8 March 2015; Accepted 3 May 2015

Academic Editor: Sherwood W. Samn

Software-defined networking (SDN) is currently seen as one of the most promising future network technologies, which can realize the separation between control and data planes. Furthermore, the increasing complexity in future wireless networks (i.e., 5G, wireless sensor networks) renders the control and coordination of networks a challenging task. Future wireless networks need good separation of control and data planes and call for SDN method to handle the explosive increase of mobile data traffic. Relying on a single controller in future wireless networks imposes a potential scalability problem. To tackle this problem, the thought of using multiple controllers to manage the large wide-area wireless network has been proposed, where the load balance problem of multicontroller needs to be resolved. In this paper, we propose a multicontroller load balancing approach called HybridFlow in software-defined wireless networks, which adopts the method of distribution and centralization and designs a double threshold approach to evenly allocate the load. Simulation results reveal that the proposed approach can significantly relieve the working load on the super controller and reduces the load jitter of multicontroller load in a single cluster compared with the BalanceFlow method.

1. Introduction

Hybrid wireless sensor networks consist of wireless networks (such as cellular network) and wireless sensor networks; such network is important to overcome the limitations of conventional sensor network where transmission range and data rate are quite limited. Wireless sensor network uses distributed data sensing and control [1, 2], while traditional wireless network uses centralized infrastructure. In this paper, we study SDN where control and data planes of the network are decoupled, which could be integrated into hybrid wireless sensor networks.

First of all, with the help of TCP/IP protocols, the Internet has made great success during the past decades. However, many challenges have emerged during this process, such as scalability, security, and QoS. The root of these problems lies in the complexity of the control and management planes—the software and protocols coordinating network elements—and particularly the way in which the decision logic and the distributed-system issues are intertwined [3], which call for new Internet architectures. As for Future Internet, software-defined network (SDN) is currently seen as one of the most promising paradigms.

Furthermore, it is envisioned that machine-to-machine (M2M) communications are rapidly developing based on the large diversity of machine type terminals, including sensors, mobile phones, consumer electronics, utility metering, and vending machines. With the dramatic penetration of embedded devices, M2M communications will become a dominant communication paradigm in the communication network, which currently concentrates on machine-to-human or human-to-human information production, exchange, and processing. M2M communications are characterized by low power, low cost, and low human intervention.

M2M communications are typically composed of billions of wireless sensors which will be developed and deployed over the coming years. The capabilities of sensors are generally limited which puts several constraints in M2M communications, including communication spectrum, energy, computation, and storage. These constraints pose a number of unique challenges in the design of network architecture and network control mechanism to achieve a highly connected, efficient, and reliable M2M communication.

This demand generates new requirements for the network architecture, such as flexibility in management and configuration, adaptability, and vendor independence. As one of the key enabling technologies of the future wireless networks (i.e., 5G), SDN offers a logically centralized control model, unprecedented programmability, and a flow-based paradigm that is ideally suited for highly scalable mobile and wireless networks, from access to backhaul and core part. With SDN, network operators can configure the behavior of both the traffic and the network in a centralized way. To meet these requirements, software-defined wireless network (SDWN) is proposed as a cost-effective solution. SDWN decouples the data plane and the control plane, enabling the direct programmability of network control and the abstraction of the underlying infrastructure from the wireless applications. Moreover, nowadays several SDN approaches have been proposed for future mobile networks (including wireless sensors), which provide a software-controlled end-to-end service management framework for future cellular networks. The flexible programmability feature of SDN makes it convenient to deploy cross-layer technological innovations and benefits the current mobile network evolution.

Unlike traditional networks, the control and data planes of the future wire and wireless network are decoupled in SDN [4]. The data plane is a packet-forwarding abstraction whereas the control plane is a centralized network-wide operating system providing open APIs for network applications and services. In SDN, the data plane is controlled by the control plane through a well-defined API. One important feature of SDN is flow table, which makes the controller flexibly configure the network switches. Another feature is that it consolidates the control plane, so that a single software control program controls multiple data planes via a famous protocol called OpenFlow [5, 6]. OpenFlow is a protocol that allows a controller to tell switches where to send packets, how to handle packets, and so on.

In the early days of OpenFlow protocol design, only a single controller is proposed [7, 8]. Since one OpenFlow controller can only support a limited number of flow setups [9, 10], with the increase of network scale and scalability requirements [11], the disadvantages of a single controller are gradually obvious. First, traffic flow to a single controller will grow with the number of switches, resulting in the need of bandwidth between a single controller and switches increasing, but the bandwidth between them is limited. Second, considering the large scale network scenario, there will always be some switches to suffer from the time delay of flow table establishment, so as to reduce the forwarding rate of data plane. Third, due to the performance of the controller processing capacity constraints of the whole system performance, the performance of a single controller is limited. With the expanding of network scale, a single controller will be difficult to deal with. In order to solve these problems, researchers have proposed distributed OpenFlow controller implementations, such as HyperFlow [12] and Onix [13]. These implementations use multiple controllers to manage the entire network and exchange controller information to make sure of the consistency of global information, which achieves the centralized control logic in the domain and improves the scalability of control plane. Moreover, the upcoming Internet2 OpenFlow production deployment also suggests using multiple controllers to manage the large wide-area network.

Although multicontrollers can be used to solve the bottlenecks of a single controller, how to realize the load balancing of multicontrollers working together is a challenging problem, especially for the large scale wireless network. Furthermore, the normal load balance in SDN is only focused on traditional scheme that is adopted in traditional networks, and the multicontroller based method is not carefully considered. The multicontroller based method will improve the whole large scale network performance considering that it holds more information about the whole network state. And the traditional method will not be suitable for the SDWN since the networks become more complex and need more control mechanism to meet the users' diversity requirements. The author of [14] proposes BalanceFlow, a controller load balancing architecture for wide-area OpenFlow networks. BalanceFlow uses CONTROLLER X action for switches and cross-controller communication and elects a controller called super controller to flexibly adjust the flow request without introducing the unnecessary transmission delay. Due to the emergence of the super controller, there is still a centralized control problem. And when frequent load balancing events happen, the stress of super controller limits the scalability of BalanceFlow. BalanceFlow can flexibly tune the flow requests handled by each controller, without introducing unacceptable propagation latencies. Experiments based on real topology show that BalanceFlow can adjust the load of each controller dynamically.

Furthermore there have also been some efforts looking at the use of SDN in wireless networks. OpenRoad [15] project at Stanford University introduced OpenFlow and FlowVisor to wireless networks to enhance the control plane. Base station virtualization from NEC concentrated on slicing radio resources at the medium access control (MAC) layer [16]. CloudEPC from Ericsson modified Long Term Evolution (LTE) control plane to control openFlow switches. SoftRAN [17] from Alcatel-Lucent considered a logically centralized control plane and scalable distributed enforcement of quality of service (QoS) and firewall policies in the data plane.

In this paper, we propose a multicontroller load balancing approach called HybridFlow in SDWN. Unlike BalanceFlow adopting the method of centralization, HybridFlow combines the method of centralization with the method of distribution. In HybridFlow, a super controller manages multiple clusters; each cluster is composed of multicontroller and switches which are in charge of the controller in this cluster, using the idea of centralizing between each cluster and using the idea of distributing cooperation between each controller in the cluster. In addition, we innovatively propose double threshold method of load balancing, where load balancing that happens is divided into the internal cluster and the external cluster according to the threshold. The load balancing scheme will take place within the cluster when a controller is slightly

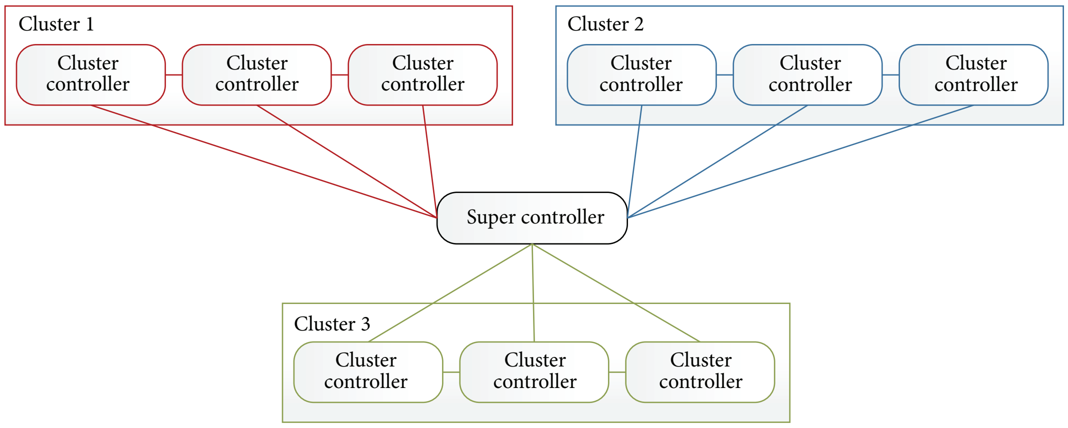


Figure 1: HybridFlow architecture.

overloaded and uses the super controller to offload the traffic among clusters. In this way, these controllers which are in the clusters handle most of the frequent load balancing and effectively shield the super controller. Compared with BalanceFlow, the design of HybridFlow relieves the working load on the super controller which is potential bottleneck in terms of scalability. Meanwhile, because of using the super controller to distribute load in multiple clusters, compared with the load balancing in a single cluster, HybridFlow can effectively reduce the jitter of the controller load.

The rest of this paper is organized as follows. In Section 2, the design of HybridFlow architecture is given. In Section 3, we will introduce the double threshold load balancing approach. In Section 4, simulation results are presented and discussed. Finally, we will conclude this paper in Section 5.

2. System Model

In this section, we will present the system model of HybridFlow in a multicontroller load balancing scenario. As shown in Figure 1, we model the large scale wireless network consisting of multiple controllers and suppose that the HybridFlow's control plane consists of a super controller and three clusters, where each cluster contains three controllers.

Each cluster controller can realize the load balancing in its cluster according to the principle of proximity. The controllers in a cluster can communicate with each other to make load balancing. When the load exceeds the processing limit of a cluster, the controllers in the cluster will transfer network requests to the super controller, which transfers excessive load to other clusters. In this way, we will not frequently transfer requests to super controller to make load balancing, which reduced super controller network overhead compared with BalanceFlow. Moreover, the super controller can realize load balancing among different clusters. It connects the controllers in each cluster, respectively, and monitors the average load status of each cluster in real time. When the following two kinds of circumstances happen, global load balancing occurs. (i) The load of one cluster is beyond its limit. (ii) Average load of one cluster increases suddenly and sharply.

As shown in Figure 2, the load status judgment thresholds are set to distinguish the above two situations. According to the two thresholds and one parameter, by using two thresholds, we can judge where load balancing happens; by using one parameter, we can effectively reduce resource consumption on signaling transmission. The signaling process to interact with other controller will not work when the traffic is low.

The problem will be divided into the following four situations: *(i) local processing without interaction*, *(ii) local processing with interaction*, *(iii) cluster processing*, and *(iv) global processing*.

The horizontal axis is load status value, calculated using the formula

$$V_i = \frac{P_{n_i}}{P_{\max}}, \qquad (1)$$

where V_i is the load status value of controller i, P_{ni} is load value of controller i at this moment, and $P_{\max}$ is the maximal load of the controller i.

(i) Local Processing without Interaction. In this situation, load status value of controller i is between 0 and parameter 1. It means that its load is lower and can deal with the load by itself.

(ii) Local Processing with Interaction. In this situation, load status value of controller i is between parameter 1 and threshold 1. The load is lower relatively but higher than situation i. The load of one controller can be handled by itself but has entered the excess load of early warming. This controller needs to start collecting load status of other neighbors' controllers in this cluster and be ready to share the load with others.

(iii) Cluster Processing. In this situation, load status value of controller i is between threshold 1 and threshold 2. It means that this controller is overloaded and immediately sends the signaling to notify other controllers in the same cluster to deal with the extra requests. If there is still some load which cannot be processed in the cluster, the overloaded controller will send the signaling to super controller to trigger a global load

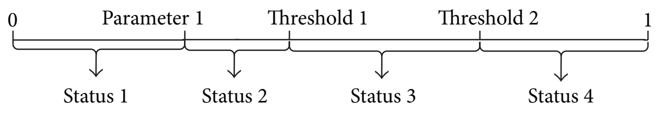


FIGURE 2: An example of distinguishing different situations according to two thresholds.

balancing. In this way, the load status of overloaded controller recovers the normal levels.

(iv) Global Processing. In this situation, load status value of controller i is between thresholds 2 and 1. This situation often happens when its load increases rapidly, which exceeds the processing capacity of the whole cluster. When the super controller obtains this information, overloaded cluster controllers are forced to transfer the excessive load for global load balancing.

3. Double Threshold Load Balancing Algorithm

The main goals of HybridFlow are to make load balancing of multicontroller and reduce the overhead of the super controller. In order to achieve these two goals, we design the double threshold load balancing algorithm. In this model, we set m clusters, $1, 2, \ldots, m$, which cluster has n controllers, $1, 2, \ldots, n$. Assuming that the total processing capacity of each controller is the same. That is to say, the biggest load capacity of each controller is $P_{\max}$. Once the load continues to increase, the controller will not receive requests. Of course, this method can also apply to the different controllers processing capacity. The implementation of the method is similar.

When starting network, each cluster controller uploads the *load_notice* signaling to super controller. The function of *load_notice* signaling is to report the V_i of each controller to super controller for cycle time T; *load_notice* signaling format is

$$\boxed{\text{Source controller IP address} \mid \text{Super controller IP address} \mid V_i} \quad (2)$$

Super controller puts the V_i which is from the same cluster in one row vector, eventually forming *status matrix* V with $M \times N$. In the *status matrix* V, the ith row is load status from cluster i, and the value of ith row and jth column denotes the load status from controller j in cluster i. Super controller calculates each cluster average load status value avg_V_k in the following formula according to each row vector in *status matrix* V:

$$\mathrm{avg}_{V_k} = \frac{\sum_{i=1}^{n} V_i}{n}. \quad (3)$$

Then we will introduce the algorithm based on the four situations mentioned in Section 2.

3.1. Local Processing without Interaction. In this scenario, there is no need to use the algorithm; controllers only need to process the load according to the protocol as status 1 in Figure 2. By setting this status, we can effectively reduce the signaling transmission overhead. When load status value is low, there is no need to exchange information with other controllers.

3.2. Local Processing with Interaction. In this scenario, assume that the load status value V_{11} of controller C_{11} is greater than threshold 1 as status 2 in Figure 2. At this point, the controller C_{11} starts to collect other neighbor controllers in cluster 1, sends *overload_notice* signaling to others to notice that the controller C_{11} may be overloaded, and asks the load status value of others. And the transmission cycle of *overload_notice* signaling is also T which is the same as *load_notice* signaling. Its format is

$$\boxed{\text{Source controller IP address} \mid \text{Destination controller IP address}} \quad (4)$$

After the controller j receives *overload_notice* signaling, it will do the following judgment:

$$\begin{aligned} V_j &< \text{parameter 1} \quad \text{subsituation 1}, \\ V_j &> \text{parameter 1} \quad \text{subsituation 2}. \end{aligned} \quad (5)$$

The controller j will reply the *load_able* information when the sub-situation 1 happens and continues $(1/4) * T$ time. The controller C_{11} will know the controller which takes the traffic offloading responsibility, and also knows the load status value. The format of *load_able* signaling is

$$\boxed{\text{Source controller IP address} \mid \text{Destination controller IP address} \mid V_j} \quad (6)$$

The controller C_{11} reads *load_able* signaling, knows which neighbor controllers in cluster 1 can do load distribution and the number of loads they can receive, and stores the source controller address and its V_j from *load_able* signaling to vector A. Signaling process is shown in Figure 6.

When subsituation 2 happens, the controllers which receive *overload_notice* signaling do not do any processing to this signaling. Because this controller also has the possibility of overload and will not receive any load from other controllers.

3.3. Cluster Processing. In this scenario, assume that the controller C_{11} detects its own load status value V_{11} being greater than threshold 1 and less than threshold 2 as status 3 in Figure 2. The controller C_{11} reads the vector A and calculates the load needed to be allocated to others using formula

$$V_{11} - \text{parameter } 1. \tag{7}$$

The controller C_{11} generates the empty load distribution vector E whose dimension is the same as A' to store the load distribution results. By vector A, we can calculate the number of maximum loads which cluster 1 can handle by using formula

$$\sum_{j \in A} \left(\text{parameter } 1 - V_{1j}\right). \tag{8}$$

When jointly considering the formulas (7) and (8), we will have the following judgment:

$$\begin{aligned} &\sum_{j \in A} \left(\text{parameter } 1 - V_{1j}\right) - \left(V_{11} - \text{parameter } 1\right) \geq 0 \\ &\qquad\qquad \text{subsituation } 3, \\ &\sum_{j \in A} \left(\text{parameter } 1 - V_{1j}\right) - \left(V_{11} - \text{parameter } 1\right) < 0 \\ &\qquad\qquad \text{subsituation } 4. \end{aligned} \tag{9}$$

When subsituation 3 happens, this load balancing process is initiated by overloading controller to the cluster internal. It means that the cluster 1 can satisfy the load distribution requirement from the controller C_{11}. C_{11} calculates the elements e_j in the vector E, corresponding to the load distribution proportion of j controller using the following formula:

$$e_j = \frac{\text{parameter } 1 - V_{1j}}{\sum_{k \in A} \left(\text{parameter } 1 - V_{1k}\right)}. \tag{10}$$

The j controller will be assigned to the load: $e_j * (V_{11} - \text{parameter } 1) * P_{\max}$, where $P_{\max}$ is load value of the controller i which can bear the maximum; e_j is the load distribution proportion of j controller. According to the calculation results of vector E, the controller C_{11} will distribute the overload to the other in cluster 1 and complete the processing of load balancing.

When subsituation 4 happens, this load balancing process is initiated by overloading controller to the cluster internal and super controller. In the cluster, only a part of load will be handled; its value is as follows: $V_s = \sum(\text{parameter } 1 - V_{1j})$, where V_s is the load which will deal with in cluster 1 and V_{1j} is load status value of controller that can receive the load in cluster 1. This part of load can use subsituation 3 solutions for load balancing.

Then we will focus on the remaining load balancing processing. The rest of load value V_r is $V_r = (V_{11} - \text{parameter } 1 - V_s)$. The controller C_{11} has launched a global load balancing processing towards super controller using *superload_notice* signaling (see Figure 3). Its format is

$$\boxed{\text{Source controller IP address} \mid \text{Super controller IP address} \mid V_r} \tag{11}$$

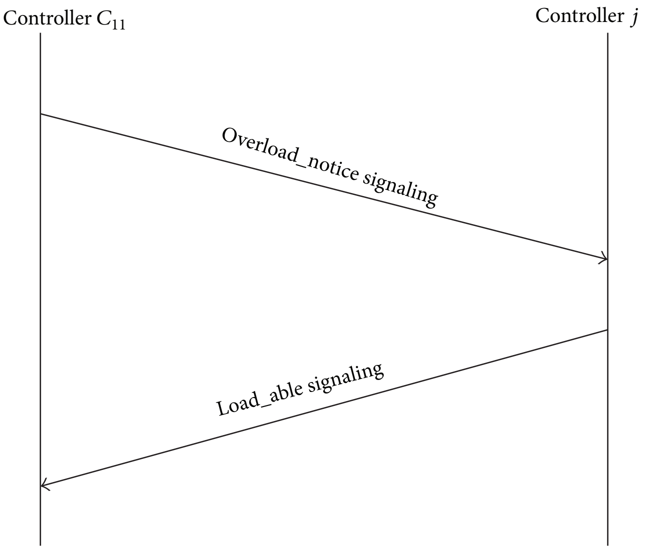


Figure 3: Workflow of signal processing.

By using this signaling, super controller can learn which controller in which cluster needs to be global load balancing and required number. Super controller combines *superload_notice* signaling and *status matrix V* in this cycle to do the global load balancing. First of all, super controller immediately checks in $(m-1) * n$ dimensional *status matrix V* except cluster 1 and generates $(m-1) * n$ dimensional empty *load distribution matrix R* which is used to store the results of load distribution proportion. Combined with formula (3) to calculate the average, super controller calculates allocation proportion of each cluster according to formula

$$r_i = \frac{1/\text{avg\_}V_i}{\sum_{k=2,\ldots,m} \left(1/\text{avg\_}V_k\right)}. \tag{12}$$

Check the i row of *status matrix V* to determine the distribution proportion of controller j in cluster i using formula

$$\begin{aligned} &\text{when } \text{th}_1 - V_{ij} \geq 0, \quad r_{ij} = \frac{\left(\text{th}_1 - V_{ij}\right)}{\sum_{\text{th}_1 - V_{ik} \geq 0} \left(\text{th}_1 - V_{ik}\right)}, \\ &\text{when } \text{th}_1 - V_{ij} < 0, \quad r_{ij} = 0, \end{aligned} \tag{13}$$

where th_1 means threshold 1 in Figure 2.

So the distribution proportion of controller j in cluster i is $r_i * r_{ij}$; accordingly, the distribution number of controller j in cluster i is $r_i * r_{ij} * V_r * P_{\max}$. According to the calculation results, the controller C_{11} will distribute the overload to the other in cluster 1 and other clusters and complete the processing of load balancing.

3.4. Global Processing. Starting from the network operation, super controller has detected each cluster's average load status value $\text{avg\_}V_k$ whether it exceeds threshold 2 in the cycle of T which is the same with *load_notice* signaling update cycle.

We assume that the controller C_{21} in cluster 2 is safe. Load balancing processing will be initiated by super controller. Super controller sends *load_ask* signaling to the controller C_{21}. Its format is as follows:

$$\boxed{\text{Super controller IP address} \mid \text{Destination controller IP address} \mid V_s} \quad (14)$$

where $V_s = (V_{21} - \text{parameter } 1)$.

The overloaded controller will know the amount of the traffic load to be uploaded to super controller by using this signaling. When the controller C_{21} receives this signaling, the number of loads corresponding to V_s will be uploaded to super controller. After receiving the load, super controller will be in accordance with the method of load balancing in subsituation 4. Thus, it will accomplish the load balancing processing initiated by super controller.

4. Simulation Results

In this section, we use computer simulation to evaluate the performance of the HybridFlow architecture and double threshold load balancing algorithm. We first describe the simulation settings and then present the simulation results.

4.1. Simulation Settings

(1) Simulation Tools. We simulate the system and algorithm in MATLAB.

(2) Network Topologies. The simulation is carried out in the topology as follows: there are 9 cluster controllers and one super controller in my system, that is to say, $m = 3$, $n = 3$, respectively, labeled in C_{11}, C_{12}, C_{13}, C_{21}, C_{22}, C_{23}, C_{31}, C_{32}, and C_{33}. We assume that the simulation is event-based, and when the network starts the controllers will receive the load randomly and immediately. We set the update cycle for *load_notice* signaling being T, so the update cycles for relevant parameters are also T, and super controller detects the average load status value in the period of T. Upon load imbalance, the system begins to load balancing algorithm presented in Section 3.

(3) Parameter Settings. In double threshold load balancing approach, the maximum working load is set to 800, $P_{\max} = 50$, parameter $1 = 0.5$, threshold $1 = 0.6$, threshold $2 = 0.8$, and $T = 10$ ms, and the time of simulation is 0–15 s. In a single threshold load balancing approach, $P_{\max} = 50$, threshold $= 0.8$, and $T = 10$ ms, and the time of simulation is 0–15 s.

(4) Performance Metrics. Given our objective to make sure of the load balancing in systems, the load jitter is the main metric in our simulation. We will compare multicontroller load jitter between HybridFlow, BalanceFlow, and traditional distribution. We also evaluate working load on super controller. Because there is no super controller in the traditional distribution, we will compare the working load on super controller in HybridFlow and BalanceFlow.

4.2. Performance Evaluation Results. Figure 4 shows the working load on super controller with different architecture. The simulation scene, respectively, is as follows: (1) the blue line is HybridFlow which consists of three clusters and one super controller, three controllers in each cluster. And we use double threshold load balancing approach which is proposed in this paper; (2) the red line is BalanceFlow which consists of nine controllers and one super controller without cluster. And we use a single threshold load balancing approach. The same simulation environment for both of them is as follows: (1) one super controller to do centralized load balancing, (2) the same data receiving rate on each controller, and (3) the same processing capacity on controllers and super controller.

Through the simulation, we can see that when the network just begins to run, the controllers in the system are at low load, and super controller workload of two methods is similar. But with the increase of time when the whole network load is increased, the working load on super controller in HybridFlow is significantly lower than BalanceFlow. Because in HybridFlow when the load is not higher than threshold 1 or in one cluster the load can be handled, there is no need to do load balancing in super controller. Thus, the load which needs to be handled by the super controller is greatly reduced, so as to relieve the working load of super controller.

Figure 5 shows the load jitter in the system with different architecture between BalanceFlow and HybridFlow. The meaning of load jitter is load standard deviation of nine controllers; when load jitter is smaller, the system load is more balanced. The simulation scene, respectively, is as follows: (1) the blue line is BalanceFlow which consists of nine controllers with one super controller. The overload can be shared between the nine controllers, and there is a super controller to centralized dispatch. And we use a single threshold load balancing approach; (2) the red line is HybridFlow which consists of three clusters and one super controller, three controllers in each cluster. And we use double threshold load balancing approach which is proposed in this paper. The same simulation environment for both of them is as follows: (1) nine controllers to handle the load, (2) the same data receiving rate on each controller, and (3) the same processing capacity on controllers and super controller.

Through the simulation, we can see that when the network just begins, since the load is lower, the system has not triggered the load balancing mechanism proposed in this paper. So load jitter in two methods is similar. But when the load is increasing, because of the difference between each controller's processing capacities, the differences between each controller's load statuses begin to emerge, which leads to the increasing of load jitter. When they arrive in a certain limit, the processing capacity of each controller is basic

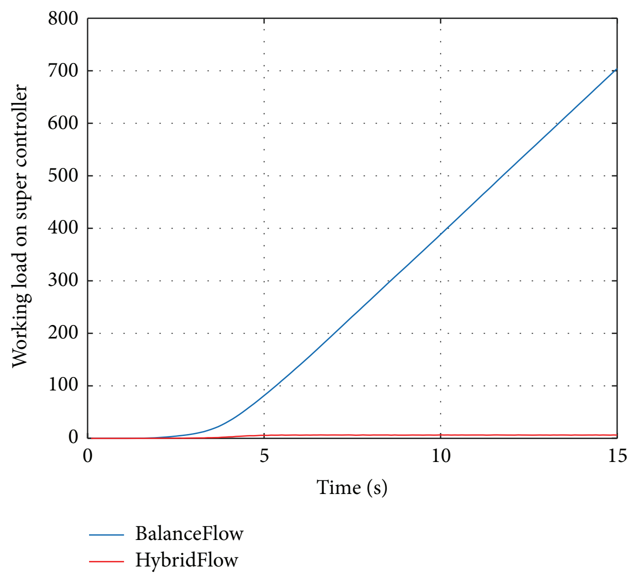


Figure 4: Working load of super controller with the same data receiving rate on controllers.

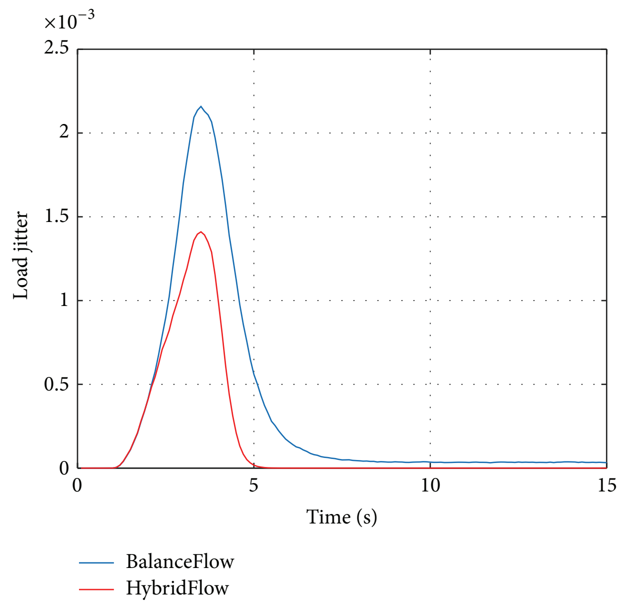


Figure 5: Load jitter of HybridFlow and BalanceFlow with the same data receiving rate on cluster controllers.

saturated, and the maximum quantity of each controller accommodating is the same, so the gap between different controllers' loads is decreasing, which leads to the decreasing of load jitter. When the system is fully saturated, each controller almost has the same number of loads, so the final load jitter is zero. Through the simulation, we can see that in the whole process, load jitter of HybridFlow is not greater than BalanceFlow's. In BalanceFlow, with the control of super controller, there are nine controllers to load balancing. When the load of a certain controller is not more than the threshold, there is no load balancing processing in the system; since the load of each controller is uneven, the value of load jitter is bigger. But in HybridFlow when the load of a certain controller exceeds threshold 1 but is not more than threshold 2, load balancing process is beginning in the cluster; that is to say, when the load between the controllers did not vary a lot, load balancing begins, so this mechanism makes the load jitter decrease obviously. From Figures 4 and 5, we can see that HybridFlow not only can reduce the workload of super controller but also has the better load balancing effect.

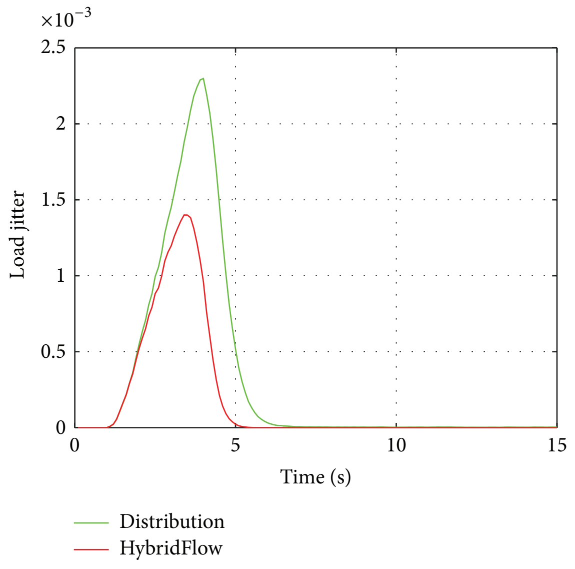


Figure 6: Load jitter of HybridFlow and Distribution with the same data receiving rate on cluster controllers.

Figure 6 shows the load jitter in the system with different architecture between Distribution and HybridFlow. The simulation scene, respectively, is as follows: (1) the green line is Distribution which consists of nine controllers without super controller. The overload can be shared between the nine controllers, but there is no super controller to centralized dispatch. In other words, these nine controllers form one cluster. And we use a single threshold load balancing approach; (2) the red line is HybridFlow which consists of three clusters and one super controller, three controllers in each cluster. And we use double threshold load balancing approach which is proposed in this paper. The same simulation environment for both of them is as follows: (1) nine controllers to handle the load, (2) the same data receiving rate on each controller, and (3) the same processing capacity on controllers and super controller.

Through the simulation, load jitter in HybridFlow is not greater than the traditional method. The tendency of the curves and the reasons are the same as shown in Figure 5. It is showed that the effect of load balancing in HybridFlow is better.

5. Concluding Remarks

In this paper, we propose a multicontroller load balancing approach called HybridFlow in SDN, which adopts the method of distribution and centralization and designs a double threshold approach to evenly allocate the load. Simulation results reveal that the proposed approach can significantly relieve the working load on the super controller and reduces the load jitter of multicontroller load in a single cluster compared with the BalanceFlow method.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

This work was supported by NSFC (61471056, 61431008), China Jiangsu Future Internet Research Fund (BY2013095-3-1), and BUPT Youth Research and Innovation Plan (2014RC0103).

References

[1] Q. Ren and Q. Liang, "Fuzzy logic-optimized secure media access control (FSMAC) protocol for wireless sensor networks," in *Proceedings of the IEEE International Conference on Computational Intelligence for Homeland Security and Personal Safety (CIHSPS '05)*, pp. 37–43, Orlando, Fla, USA, March 2005.

[2] Q. Liang and L. Wang, "Event detection in wireless sensor networks using fuzzy logic system," in *Proceedings of the IEEE International Conference on Computational Intelligence for Homeland Security and Personal Safety (CIHSPS '05)*, pp. 52–56, Orlando, Fla, USA, April 2005.

[3] A. Greenberg, G. Hjalmtysson, D. A. Maltz et al., "A clean slate 4D approach to network control and management," *ACM SIGCOMM Computer Communication Review*, vol. 35, no. 5, pp. 41–54, 2005.

[4] N. Feamster, J. Rexford, and E. Zegura, "The road to SDN: an intellectual history of programmable networks," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 2, pp. 87–98, 2014.

[5] N. McKeown, T. Anderson, H. Balakrishnan et al., "OpenFlow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[6] D. Drutskoy, E. Keller, and J. Rexford, "Scalable network virtualization in software-defined networks," *IEEE Internet Computing*, vol. 17, no. 2, pp. 20–27, 2013.

[7] Openflow Switch Consortium, *Openflow Switch Specification Version 1.0.0*, 2009.

[8] Specification O F S. Version 1.1.0 Implemented, 2011.

[9] N. Gude, T. Koponen, J. Pettit et al., "NOX: towards an operating system for networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 3, pp. 105–110, 2008.

[10] A. Tavakoli, M. Casado, T. Koponen et al., *Applying NOX to the Datacenter*, HotNets, 2009.

[11] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "Devoflow: scaling flow management for high-performance networks," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4, pp. 254–265, 2011.

[12] A. Tootoonchian and Y. Ganjali, "HyperFlow: a distributed control plane for OpenFlow," in *Proceedings of the Internet Network Management Conference on Research on Enterprise Networking*, p. 3, USENIX Association, 2010.

[13] T. Koponen, M. Casado, N. Gude et al., "Onix: a distributed control platform for large-scale production networks," in *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation (OSDI '10)*, pp. 1–6, October 2010.

[14] Y. Hu, W. Wang, X. Gong, X. Que, and S. Cheng, "BalanceFlow: controller load balancing for OpenFlow networks," in *Proceedings of the 2nd IEEE International Conference on Cloud Computing and Intelligence Systems (IEEE CCIS '12)*, vol. 2, pp. 780–785, November 2012.

[15] A. Gudipati, D. Perry, L. E. Li, and S. Katti, "Softran: software defined radio access network," in *Proceedings of the 2nd ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN '13)*, pp. 25–30, August 2013.

[16] C. J. Bernardos, A. de la Oliva, P. Serrano et al., "An architecture for software defined wireless networking," *IEEE Wireless Communications*, vol. 21, no. 3, pp. 56–61, 2014.

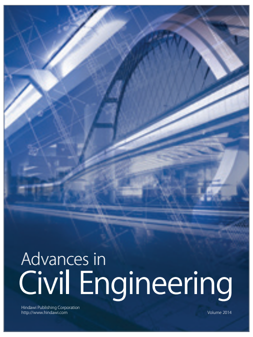
[17] K. Pentikousis, Y. Wang, and W. Hu, "Mobileflow: toward software-defined mobile networks," *IEEE Communications Magazine*, vol. 51, no. 7, pp. 44–53, 2014.

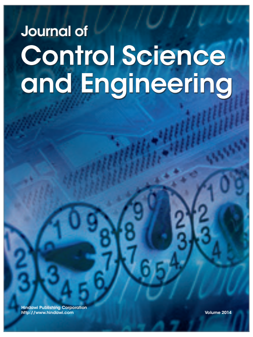
Journal of
Engineering

The Scientific World Journal

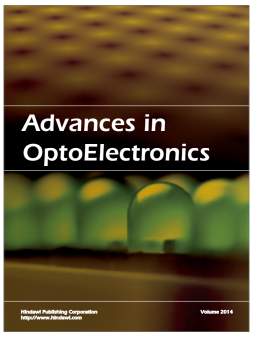
International Journal of
Rotating Machinery

Journal of
Sensors

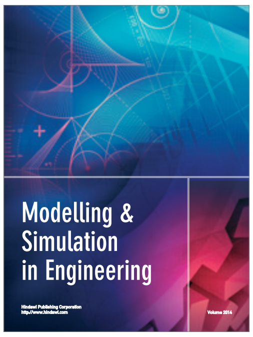
International Journal of
Distributed Sensor Networks

Advances in
Civil Engineering

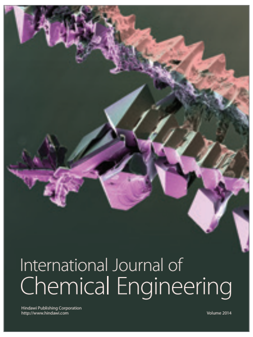
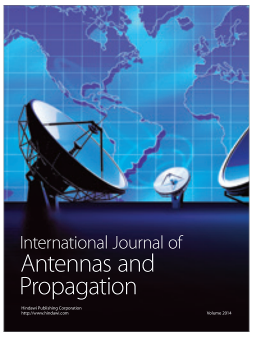

Journal of
Control Science and Engineering

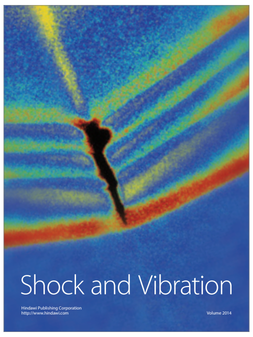
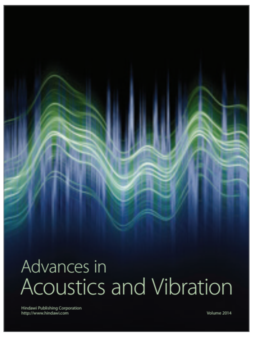
Journal of
Robotics

Journal of
Electrical and Computer Engineering

Advances in
OptoElectronics

VLSI Design

International Journal of
Navigation and Observation

Modelling & Simulation in Engineering

International Journal of
Aerospace Engineering

International Journal of
Chemical Engineering

International Journal of
Antennas and Propagation

Active and Passive Electronic Components

Shock and Vibration

Advances in
Acoustics and Vibration