Efficient Load Balancing for Multi-Controller in SDN-based Mission-critical Networks

Nguyen Tien Hai, Dong-Seong Kim
Department of IT Convergence, School of Electronic Engineering
Kumoh National Institute of Technology, Gumi, South Korea
{haint,dskim}@kumoh.ac.kr

***Abstract*—This paper proposes an efficient load balancing scheme for distributed controller in Software-Defined Networking-based mission-critical networks. Based on load status and dynamic weight coefficient on each controller, the proposed scheme guarantees load balancing in control plane to achieve better resource usage, reliability and resilience in the network. Furthermore, by using pre-defined load threshold, communication overhead is significantly reduced in comparison with existing approaches. Simulation results demonstrate that the proposed scheme is effective in terms of communication overhead and performance of load balancing in the control plane.**

***Keywords:* Software-Defined Networking (SDN), load balancing, distributed controller, communication overhead, Single Point of Failure, mission-critical networks.**

I. INTRODUCTION

Nowadays, mission-critical networks provide applications to control and monitor energy, railways, public-safety, aviation management, etc. The network thus must be reliable and resilient. In case of a mission-critical network fails, people's lives and companies can be at risk. Load balancing has been proven as one of the very critical problems in providing reliable network by assuring the system is not overloaded and also by high resource utilization [1]–[4]. With the emergence of Software-Defined Networking (SDN) paradigm which has been touted as a promising solution for future Internet, mission-critical networks can be improved in reliability, resilience and security by leveraging SDN's features.

Software Defined Networking is an emerging network paradigm that separates the networking control plane and the data forwarding plane. This separation allows managing the network in a flexible and efficient way through software programs. In SDN, the control plane is implemented using a logically centralized controller, which runs on a server hardware. The programmable controller, which acts as a bridge between forwarding plane and network administrator, manages forwarding plane by keeping the global network status which is reported from forwarding plane. Because control intelligence is decoupled, OpenFlow-enabled switching devices in forwarding plane (i.e. OpenFlow (OF) switches) are responsible for switching and forwarding traffic flows entering the network based on rules in the flow table installed by the controller.

In SDN, OpenFlow is a widely used protocol that enables the controller to communicate with OF-switches [5]. Specifically, it enables the controller to remotely install, modify, and even delete forwarding rules in OF-switch's flow table via the OpenFlow channel [6]. When a new flow enters the network, the OF-switch will encapsulate a new packet of the flow into a *Packet-In* message and send to the controller to ask for an appropriate behavior. The controller then selects a forwarding path for the new flow based on the current global network status and sets up the forwarding path by using *Flow_Mod* messages to reactively install rules on related OF-switches.

In large-scale SDN (e.g. data center network which is a typical type of the mission-critical networks), when number of network elements and traffic flows increase considerably, using a single physically centralized controller creates poor reliability problem. This problem is known as Single Point of Failure (SPoF). To this end, a basic idea is to use multiple controllers working together as logically centralized controller, which not only benefits from scalability and reliability of the distributed architecture but also preserves simplicity of the logically centralized system. However, another crucial problem is how to efficiently share flow requests from OF-switches to controllers in order to guarantee load balancing which remarkably affects network performance.

Several recent studies have focused on load balancing in the control plane. BalanceFlow [7] introduces controller X as an extension for OF-switch. This does not only make operation of OF-switch more complex, but also causes high communication overhead at the control plane from using a periodical approach. In this scheme, the controllers need to publish their load information periodically via a cross-controller communication channel. Similarly, authors in [8] proposes a dynamic load re-balancing method based on switch migration mechanism for clustered controllers. However, this mechanism faces the same problem between controllers in a cluster due to periodical load collection of coordinator node. Thus, it can not be applied for large-scale networks. Additionally, a dynamic and adaptive algorithm named DALB proposed by Y. Zhou *et al.* [9] forces controllers to collect load from each other in case load on each controller exceeds a certain load threshold at the same time. This approach also causes high communication overhead in the control plane.

Motivated from these above challenges and inspired from FlowVisor [10], which is a network visualization layer between the control plane and data plane, this paper proposes an efficient load balancing scheme built on a load balancer for multi-controller in SDN-based mission-critical network. The main contributions of this paper can be summarized as follows:

978-1-5090-2870-2/16/$31.00 ©2016 IEEE

1) Firstly, by using multi-controller architecture, the proposed approach eliminates the SPoF problem in the control plane and provides a reliable network, which is a crucial requirement for SDN-based mission-critical network.
2) Secondly, distinguished from previous works, the paper proposes a dynamic and adaptive load balancing strategy that automatically allocates flow requests to the control plane in a balanced manner, based on the dynamic weight coefficient on each controller. The weight coefficients are adjusted adaptively according to load status on controllers. Furthermore, in order to reduce communication overhead, the load status on each controller is only sent to the load balancer in cases of exceeding a predefined load threshold value. Simulation results show that the proposed scheme achieves good load-balancing performance in the control plane with much less communication overhead compared with existing schemes.

The rest of this paper is organized as follows. Section II presents system model. Section III analyzes the dynamic and adaptive load balancing algorithm. Then, the proposed approach is evaluated in Section IV and finally, section V finishes the paper with a brief conclusion.

II. System Model

The network system consists of three planes: control plane, load balance plane, and data plane, respectively, as shown in Fig. 1. In the top plane, there are multiple controllers, denoted from controller 1 to n, having a global view of the entire network. According to processing capacity of the controller, the administrator configures an initial weight coefficient (w) to each controller. Then, the weight coefficients would be adjusted based on load status on the controllers in order to guarantee load balancing.

In the middle plane, namely load balance, there are two load balancers with the roles as primary and secondary, respectively. The load balancers communicate with both the controllers and OF-switches via OpenFlow protocol. The primary load balancer stays in active status while the other one is in standby status. A hot standby strategy [11] is executed by two load balancers for guaranteeing redundancy and keeping network available in real-time processing in case primary load balancer fails. Specifically, state of the primary load balancer is completely synced with the secondary one when any changes occur. If primary load balancer becomes unavailable, the secondary one is replaced automatically and continues processes based on the current state. Hence, this procedure can provide the minimum recovery time.

Finally, the data plane contains a set of interconnected OF-switches which can be virtually or physically running OpenFlow protocol. Different from conventional switch devices, these forwarding elements are simply responsible for gathering and reporting network status to the control plane as well as performing packet forwarding according to forwarding rules imposed by the controller in the flow table. Consequently, OF-switch becomes more simpler and easier to manufacture which can provide a low cost solution.

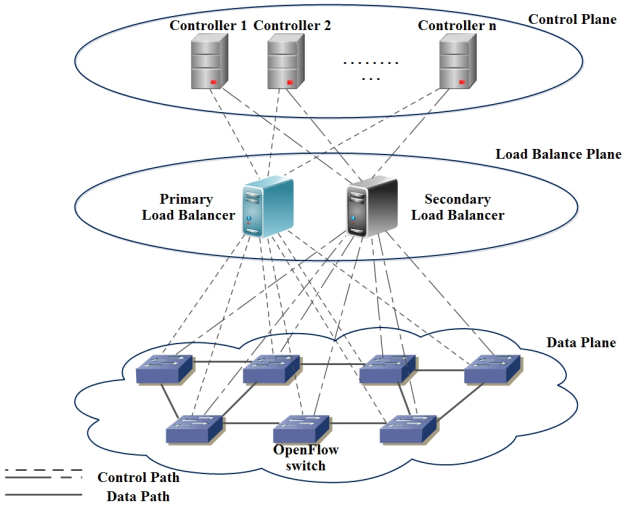

Fig. 1. Deployment of load balancer for a large-scale SDN-based mission-critical network.

A. Load Balancer Architecture

Load balancer takes responsibility for prioritizing flow requests in queue to guarantee QoS and then forwarding them to the controllers based on weight coefficients. Assuming that traffic flows enter the network being classified into two main types of traffic: critical traffic (e.g. real-time traffic such as VoIP or VoD, etc.) and non-critical traffic (e.g. non-real-time traffic such as HTTP or FTP, etc.). To guarantee QoS, the priority procedure ensures that the critical traffic is processed by the controllers prior to the non-critical ones based on necessary information in packet header. Architecture of the load balancer is illustrated in Fig. 2.

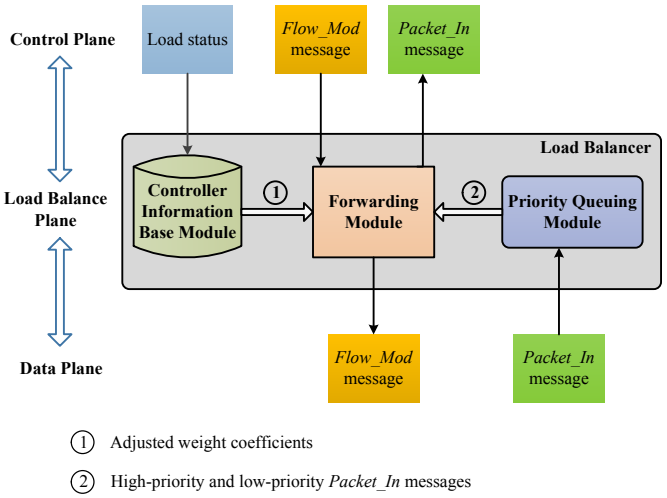

Fig. 2. Load balancer architecture.

The architecture of the load balancer consists of three modules: Controller Information Base module, Priority Queuing module, and Forwarding module. The rest of this subsection presents these modules in detail.

1) Priority Queuing Module: When a new flow arrives at OF-switch, the OF-switch would assess header of a new packet of the flow and match it against flow entries in the flow table. If a matching flow entry is found, the OF-switch would apply an appropriate action (forward, drop, or policies) associated with the matching flow entry and update its respective entry counter. Otherwise, the packet would be encapsulated into *Packet_In* message and then sent to the controller by default to ask for an appropriate behavior. Upon receiving *Packet_In* message from the OF-switch, based on the necessary information (e.g. source/destination MAC address, source/destination IP address, TCP/UDP port, etc.) in packet header, this module will decide to put the *Packet_In* message into high-priority or low-priority level in queue. The priority process of this module will be demonstrated in section II-B.

2) Controller Information Base Module: This module collects load status from controllers and performs adjusting weight coefficients following Dynamic and Adaptive Load Balancing algorithm, which is analyzed in section III. Then, it will send the weight coefficients to the Forwarding module.

3) Forwarding Module: Receiving both the adjusted weight coefficients and prioritized *Packet_In* messages from Controller Information Base module and Priority Queuing module, respectively, Forwarding module distributes *Packet_In* messages to controllers in such a way that guarantees load balancing. Furthermore, this module transfers *Flow_Mod* messages, which are response to *Packet_In* ones, from controllers to OF-switches that are on the forwarding path. This transfer will install forwarding rules in the flow table.

B. Priority Queuing

When *Packet_In* message arrives at the load balancer, Priority Queuing module would extract the packet header to classify the message's priority level based on necessary information, such as source/destination IP/MAC address, source/destination port number or value of the Type of Service (ToS) bits, etc. For example, VoIP traffic using UDP port 16376 would be queued into high-priority while FTP traffic using TCP port 21 stays in low-priority queue. Architecture of Priority Queuing module is illustrated in Fig. 3. A message, which has a higher priority in the queue, would be transmitted first. The messages with the same priority will be processed in First-In-First-Out (FIFO) manner [12]. In this paper, priority queuing procedure is assumed to be non-preemptive. That means that while low-priority message is being processed, high-priority message can not interrupt.

Fig. 4 demonstrates priority queuing procedure in processing prioritized messages. When high-priority message 1 arrives at the load balancer, it is transmitted to the controller for processing. Meanwhile, message 2 and message 3 arrives and are then queued in the low and high-priority. When the transmission of message 1 finishes, even though message 2 came earlier but has a lower priority than message 3, message 3 would be transmitted next. Moreover, due to non-preemptive priority queuing procedure, when message 4 arrives, the transmission of message 2 is not interrupted once it has begun. That means message 4 needs to be queued until transmission of message 2 is completed.

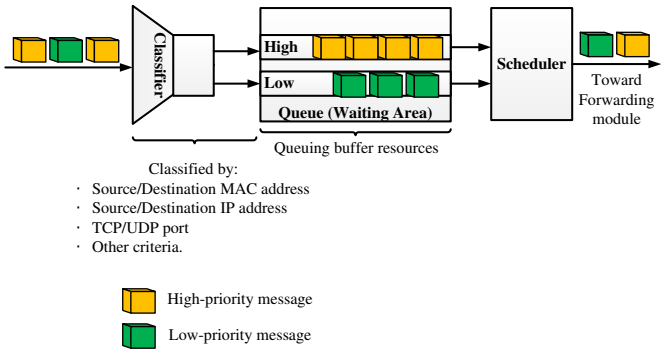

Fig. 3. Architecture of Priority Queuing module with two priority levels.

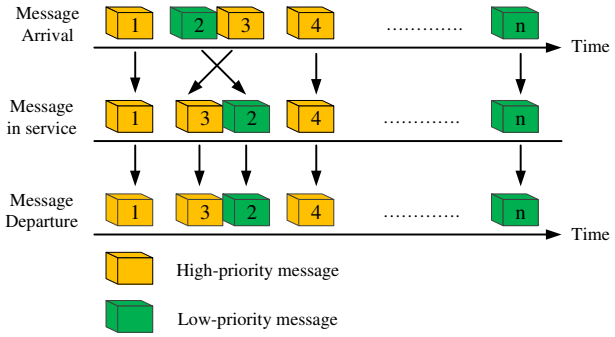

Fig. 4. Procedure of priority queuing in processing prioritized messages.

III. Dynamic and Adaptive Load Balancing Algorithm

A. Load Status Measurement

In our proposed network system, each controller calculates its own current load status $L_{cur}(C_i)$ based on load metrics including current CPU usage (L_{cpu}), current memory usage (L_{mem}), and current hard disk usage (L_{disk}) according to the following formula:

$$L_{cur}(C_i) = A_1 L_{cpu}(C_i) + A_2 L_{mem}(C_i) + A_3 L_{disk}(C_i) \quad (1)$$

with below constraints:

$$\begin{cases} i = 1, 2, ..., n \\ \sum_{j=1}^{n} A_j = 1 \\ (L_{cpu}(C_i), L_{mem}(C_i), L_{disk}(C_i)) \in (0, 1) \end{cases}$$

where C_i expresses each controller and n denotes number of controllers. Set of coefficients (A_j) represents important level of load metrics and can be modified by administrator until obtaining suitable values for the current load status. In practice, load status is proportional to the number of *Packet_In* messages needed to process by the controller.

In order to avoid high communication overhead between the load balancer and controllers, we use a pre-defined load threshold (T_i) on each controller, which is configured by administrator according to the controller's processing capacity.

The controller periodically measures load status following Eq. 1 and checks whether its load exceeds the load threshold or not. If the load exceeds the load threshold, the controller would send a notification message including its current load status to the load balancer until it receives acknowledge message. The other controllers would send their load status upon receiving of load status requests from the load balancer. Fig. 5 shows the procedure of updating load status on the load balancer.

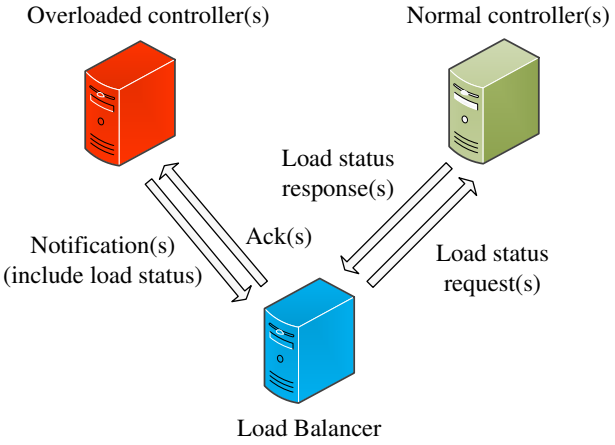

Fig. 5. The procedure of updating load status of controllers on the load balancer.

B. Dynamic and Adaptive Load Balancing Algorithm

The proposed scheme aims to eliminate SPoF problem and guarantees load balancing in the control plane with much less communication overhead. To do this, when overload problem occurs in the controller, the proposed scheme adjusts the weight coefficients to make current load status on controllers become closer to current mean load, which is determined as below:

$$\overline{L}_{cur} = \frac{1}{n}\sum_{i=1}^{n} L_{cur}(C_i) \tag{2}$$

where $\overline{L}_{cur}$, n, and $L_{cur}(C_i)$ denotes the current mean load, number of controllers, and the current load status on controller C_i, respectively.

Considering the ratio of the current load status to the current mean load: $r = L_{cur}(C_i)/\overline{L}_{cur}$. It is clearly observable that there can be three cases:

1) $r > 1 \Leftrightarrow L_{cur}(C_i) > \overline{L}_{cur}$: The current load status is bigger than the current mean load. In this case, the weight coefficient on the controller should be decreased.
2) $r < 1 \Leftrightarrow L_{cur}(C_i) < \overline{L}_{cur}$: The current load status is smaller than the current mean load. The weight coefficient on the controller needs to be increased.
3) $r = 1 \Leftrightarrow L_{cur}(C_i) = \overline{L}_{cur}$: The current load is equal to the current mean load. The weight coefficient is to remain unchanged.

From these cases, the weight coefficient function can be constructed:

$$W_{adj}(C(i)) = [W_{cur}(C(i)) + (1 - \frac{L_{cur}(C(i))}{\overline{L}_{cur}(C(i))})] \tag{3}$$

where $W_{adj}(C(i))$ and $W_{cur}(C(i))$ are the adjusted and current weight coefficient, respectively.

Algorithm 1 Dynamic and Adaptive Load Balancing Algorithm

1: **Input:** Collection of current load status $(L_{cur}(C_i))$ and current weight coefficients $(W_{cur}(C(i)))$;
2: **Output:** Adjusted weight coefficients.
3: **Upon** *Current load status* **received** *Controller Information Base module* **do**
4: Calculate the current mean load value $(\overline{L}_{cur})$, as shown in Eq. 2.
5: **for all** *Controllers* **do**
6: Calculate the adjusted weight coefficient $(W_{adj}(C(i))$, as shown in Eq. 3.
7: **end for**.
8: Send adjusted weight coefficients to *Forwarding module*.

Algorithm 1 shows the procedure of weight coefficient adjustment that is dynamic and adaptive to changes in load status on the controller. When overload problem occurs in a specific controller, the controllers would send their current load status to the load balancer. Upon receiving the current load status from the controllers, Controller Information Base module obtains the current mean load value following Eq. 2. Then, the weight coefficients would be adjusted according to Eq. 3 to guarantee load balancing among controllers. Finally, Controller Information Base module sends these weight coefficients to Forwarding module for forwarding prioritized *Packet_In* messages to the control plane.

IV. Performance Evaluation

The proposed approach is verified under a specific network topology, which is described in the below subsection, so as to evaluate two properties: 1) communication overhead between the controllers and the load balancer; 2) performance of load balancing in the control plane. In addition, the comparison of Round Trip Time (RTT) delay between real-time traffic and non-real-time traffic is also investigated.

A. Experimental Setup

Fig. 6 shows the network topology which is created by using Mininet network emulator version 2.2.1 [13]. Mininet emulator provides a high level of reliability for realistic and reproducible network experiments. The fat-tree topology is chosen for evaluation because it is a typical type of real data center network in which there are much traffic flows and various traffic types that enter the network.

In control plane, due to limitation of hardware, we only chose three Floodlight controllers version 1.0 [14], denoted by

TABLE I. SIMULATION PARAMETERS

Controller	Load threshold (Messages/sec.)	Weight coefficient
A	1400	3
B	1300	2
C	1200	1

A, B, and C, connecting to the load balancer in the load balance plane. In practice, the proposed scheme can be implemented without limitation of number of controllers. Floodlight is an open source OpenFlow controller and high performance, which can support a broad range of physical and virtual OF-switches. The load threshold values and initial weight coefficients for controller A, B, and C are listed in Table I. The whole system runs on Ubuntu 64bit 14.04 computer with Intel Corei5-4460 CPU 3.2GHz and 16GB RAM. We generated background traffic by using *hping* tool [15] for the following two cases:

1) Generating background traffic at speeds: 200Mbps, 400Mbps, 600Mbps, 800Mbps, and 1000Mbps, then injecting them into the network to evaluate performance of load balancing among controllers. In each time, a number of *Packet_In* messages on each controller is counted and compared to each other.
2) Generating background traffic between hosts: host 1 - host 8 using UDP port and host 2 - host 7 using TCP port. The operation is repeated 20 times until the traffic between two hosts reaches 1000Mbps for one direction. In each time, 200 additional data flows are generated with each of them transferring data at speeds of 250Kbps using different UDP ports for host 1 - host 8 and different TCP ports for host 2 - host 7.

In this simulation, we assume that the background traffic using UDP port and TCP port is real-time traffic and non-real-time traffic, respectively. In order to clearly observe the effectiveness of the proposed Priority Queuing module, Round Trip Time (RTT) delay of each couple of hosts is measured using *hping* tool. This delay is the amount of time required to handle *Packet_In* message of each data flow by the controllers, and installing forwarding rules along with forwarding data of all OF-switches that are on forwarding path.

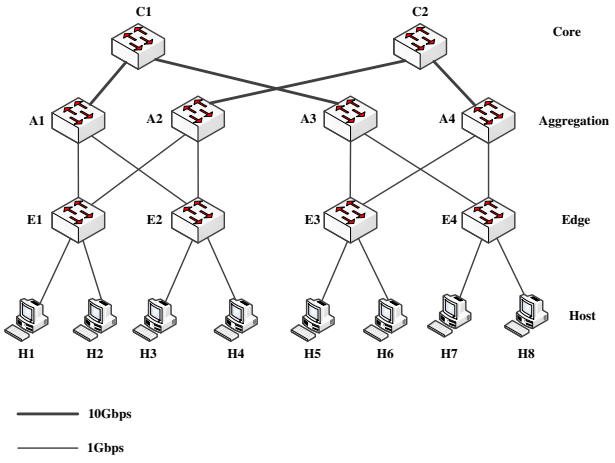

Fig. 6. Experiment network topology in Mininet.

B. Numerical Results

1) Communication Overhead: To investigate the efficiency of the proposed approach in terms of communication overhead, we compare the number of messages needed to exchange load status in the proposed approach to that of the existing approaches (e.g. BalanceFlow, DALB), which are mentioned in section I.

Assuming that there are n controllers in the control plane, the existing schemes require $n(n-1)$ messages needed to exchange load status among the controllers because they need to collect load status on each other. In contrast, there are only $2n$ messages exchanging between the load balancer and the controllers in the proposed scheme when overload problem occurs. Based on this theoretical evaluation, without loss of generality, we conduct comparison under different number of controllers from 3 to 10. Fig. 7 shows that the proposed approach significantly outperforms the existing approaches regarding the number of messages. Obviously, when the number of controllers increases, the number of messages in the proposed approach is much lower than that in existing approaches. For instance, in case of 10 controllers, the number of messages in the proposed approach only equals to 22% of that in the existing schemes.

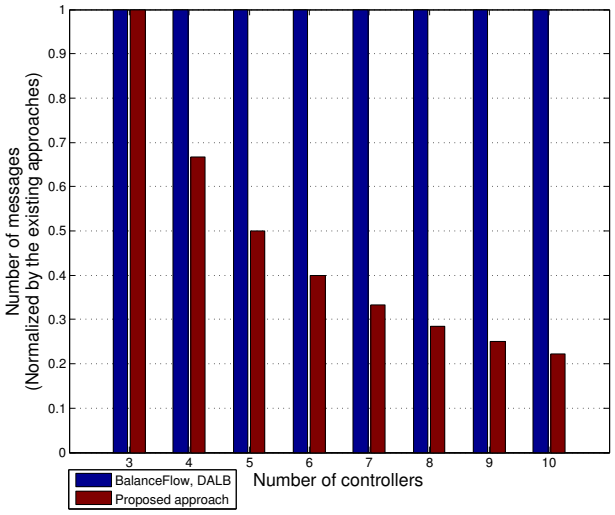

Fig. 7. Normalized number of messages needed to exchange load status against number of controllers.

2) Performance of Load Balancing in Control Plane: The performance of load balancing among controllers is shown in Fig. 8. At 200Mbps and 400Mbps, the load on three controllers are relatively equal to each other. When background traffic increases, the load on controller A reaches 1423 *Packet_In* messages and exceeds the pre-defined load threshold (1400 messages/sec) at 800Mbps. Because overload problem occurs in controller A, three controllers send their load status to the load balancer in order to adjust weight coefficients. Consequently, at 1000Mbps, the load on controller A, B, and C is measured as 1291 messages, 1250 messages and 1195 messages, respectively. The results indicate that the load balancing in the control plane is guaranteed.

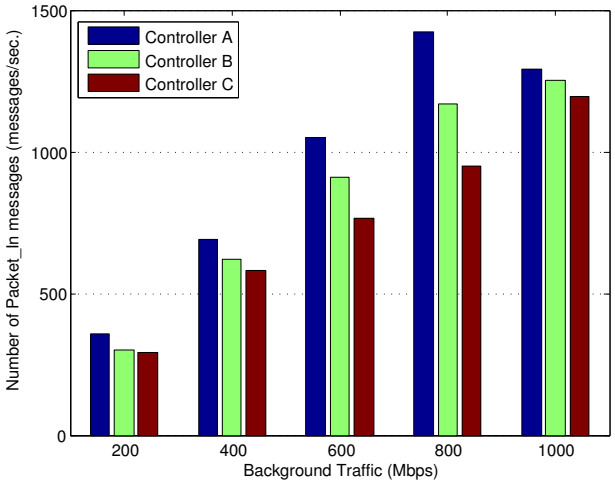

Fig. 8. Number of *Packet_In* messages handled by three controllers.

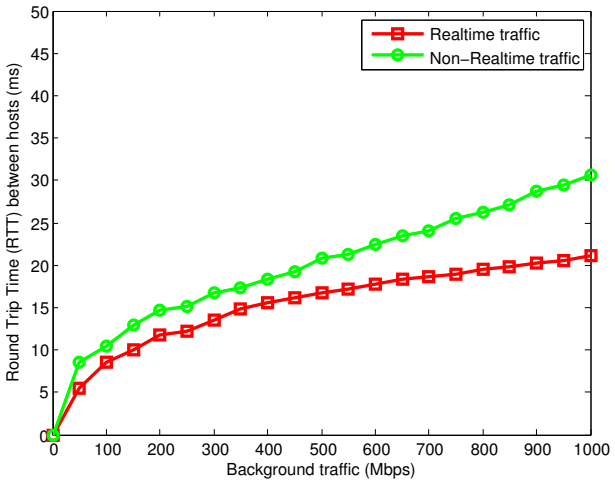

Fig. 9. Round Trip Time (RTT) between hosts: host 1 - host 8 and host 2 - host 7 transferring real-time traffic and non-real-time traffic, respectively.

3) Round Trip Time: Figure 9 shows the comparison of Round Trip Time (RTT) between hosts: host 1 - host 8 and host 2 - host 7. It is clearly seen that the time delay between host 1 - host 8 is smaller than that in host 2 - host 7. In particular, RTT is 21.1ms in case of host 1 - host 8 while host 2 - host 7 is 30.6ms at 1000Mbps. This occurs because Priority Queuing module is responsible for classifying *Packet_In* messages in queue based on port information in packet header. *Packet_In* messages of the data flows in real-time traffic are guaranteed to be handled by the controllers prior to that in non-real-time traffic.

V. Conclusion

In this paper, an efficient load balancing scheme is proposed for multi-controller in SDN-based mission-critical networks. By using a distributed architecture, the proposed scheme eliminates Single Point of Failure (SPoF) problem in the control plane to provide a reliable and resilient network. To assure load balancing in the control plane, the load status of each controller is taken into account to adjust weight coefficients when overload problem occurs. The simulation results showed that the proposed scheme is effective in the performance of load balancing and considerably reduces communication overhead in comparison with existing approaches by using pre-defined load threshold value. Furthermore, with the objective of guaranteeing QoS, the effectiveness of proposed Priority Queuing module is also proven for real-time traffic and non-real-time traffic in terms of Round Trip Time.

VI. Acknowledgment

This research was supported by the MSIP(Ministry of Science, ICT and Future Planning), Korea, under the ITRC(Information Technology Research Center) support program(IITP-2016-H8601-16-1011) supervised by the IITP(Institute for Information & communications Technology Promotion).

References

[1] R. Dilmaghani and D. Kwon, "Evaluation of openflow load balancing for navy," in *IEEE Military Communications Conference, MILCOM 2015*, Oct 2015, pp. 133–138.

[2] D.-S. Kim, Y. S. Lee, W. H. Kwon, and H. S. Park, "Maximum allowable delay bounds of networked control systems," *Control Engineering Practice*, vol. 11, no. 11, pp. 1301 – 1313, 2003.

[3] S. Q. Hojiev and D.-S. Kim, "Dynamic load balancing algorithm based on users immigration in wireless lans," *Journal of Advances in Computer Networks*, vol. 3, no. 2, 2015.

[4] P. T. A. Quang and D.-S. Kim, "Throughput-aware routing for industrial sensor networks: Application to isa100.11a," *Industrial Informatics, IEEE Transactions on*, vol. 10, no. 1, pp. 351–363, Feb 2014.

[5] Openflow protocol. [Online]. Available: https://www.opennetworking.org/sdn-resources/openflow

[6] T.-T. Nguyen and D.-S. Kim, "Accumulative-load aware routing in software-defined networks," in *IEEE 13th International Conference on Industrial Informatics (INDIN), 2015*, July 2015, pp. 516–520.

[7] Y. Hu, W. Wang, X. Gong, X. Que, and S. Cheng, "Balanceflow: Controller load balancing for openflow networks," in *IEEE 2nd International Conference on Cloud Computing and Intelligent Systems (CCIS), 2012*, vol. 02, Oct 2012, pp. 780–785.

[8] C. Liang, R. Kawashima, and H. Matsuo, "Scalable and crash-tolerant load balancing based on switch migration for multiple open flow controllers," in *2014 Second International Symposium on Computing and Networking (CANDAR)*, Dec 2014, pp. 171–177.

[9] Y. Zhou, M. Zhu, L. Xiao, L. Ruan, W. Duan, D. Li, R. Liu, and M. Zhu, "A load balancing strategy of sdn controller based on distributed decision," in *IEEE 13th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, Sept 2014, pp. 851–856.

[10] R. Sherwood, G. Gibb, K. kiong Yap, M. Casado, N. Mckeown, and G. Parulkar, "Flowvisor: A network virtualization layer," Deutsche Telekom Inc. R&D Lab, Stanford University, Nicira Networks, Tech. Rep., 2009.

[11] V. Pashkov, A. Shalimov, and R. Smeliansky, "Controller failover for sdn enterprise networks," in *International on Science and Technology Conference (Modern Networking Technologies) (MoNeTeC)*, Oct 2014, pp. 1–6.

[12] M. S. Momanyi E.O, Oduol V.K, "First in first out (fifo) and priority packet scheduling based on type of service (tos)," vol. 4, no. 7. Journal of Information Engineering and Applications, 2014.

[13] Mininet network emulator. [Online]. Available: https://mininet.org/blog/2015/04/16/announcing-mininet-2-2-1/

[14] Floodlight controller. [Online]. Available: http://www.projectfloodlight.org/download/

[15] hping. [Online]. Available: http://www.hping.org/