Scientific Research Publishing

Engineering, 2018, 10, 863-875
http://www.scirp.org/journal/eng
ISSN Online: 1947-394X
ISSN Print: 1947-3931

Analytical Load Balancing Model in Distributed Open Flow Controller System

Traoré Issa[1], Zamblé Raoul[2], Adama Konaté[2,3], Joël Christian Adepo[3,4], Bernard Cousin[5], Asseu Olivier[2,3]

[1]Felix Houphouet-Boigny University, Abidjan, Cote d'Ivoire
[2]African Higher School of ICT Abidjan, Abidjan, Cote d'Ivoire
[3]National Polytechnic Institute Felix Houphouet-Boigny, Yamoussoukro, Cote d'Ivoire
[4]Virtual University of Cote d'Ivoire, Abidjan, Cote d'Ivoire
[5]Research Institute of Computer Science and Random Systems, University of Rennes 1, Rennes, France

Email: issa.traore@univ-fhb.edu.ci

How to cite this paper: Issa, T., Raoul, Z., Konaté, A., Adepo, J.C., Cousin, B. and Olivier, A. (2018) Analytical Load Balancing Model in Distributed Open Flow Controller System. *Engineering*, **10**, 863-875.
https://doi.org/10.4236/eng.2018.1012060

Received: October 29, 2018
Accepted: December 25, 2018
Published: December 28, 2018

Abstract

The Software Defined Network (SDN) is a concept based on a decoupling between the control plan and the data plan of a network. Thus, the network becomes programmable and can be coupled to the business applications of the users. The study that is discussed in this article looks at load planning and balancing in distributed controllers. To do this, a model and theoretical methods of performance evaluation related to appropriate software tools, to predict and control the quality of service offered to users is exposed. This paper exposed also a distributed architecture of controllers and then a module based on an adaptive load balancing algorithm that is fault tolerant and fluctuates controller loads. The experiments show a significant gain in efficiency of our solution.

Keywords

Open Flow, Load Balancing, Decision Distributed, Software Defined Network, Analysis, Architecture

1. Introduction

Software programmed by network [1] [2] and [3], Software-Defined Networking (SDN) based on OpenFlow [4] are currently considered as one of the most promising paradigms of Internet's future. According to the Open Network Fundation (ONF), a consortium of nonprofit companies founded in 2011 to promote SDN and standardize its protocols, the SDN is an architecture that separates the control plan from the data plan, and unifies control in an external con-

trol software called controller, to manage several elements of the data plan via APIs (Application Programming Interface). In [4], ONF offers the typical architecture of SDN.

One of the advantages of SDNs is network abstraction, which means that the control plan provides an abstract view of the applications. This allows the network to be unified and simplifies configuration and management. In addition, the use of the SDN concept in the network can provide innovative services, including multicast routing, security, access control, bandwidth management, traffic engineering, QoS energy efficiency and various forms of strategy management.

The SDN thus has several advantages, but the fact of concentrating all the intelligence of the network in the control plan raises concerns about the performance and scalability of this plan. These concerns arise more for the initial SDN architecture which proposes to use a single controller [5] and [6], which then becomes a single point of failure (SPOF) and raises the lack of scalability and performance. Therefore, the need to use multiple controllers becomes a necessity to overcome the SPOF problem and improve performance.

Indeed, the centralized controller imposes potential problems of overload, scalability and availability. As a result, the architecture of logically distributed controllers has been proposed. A cluster-based distributed controller runs on multiple physical controllers as a single logical controller to control multiple network switches.

Compared to conventional centralized controllers, cluster-based distributed controllers provide better scalability and fault tolerance [7] and [8]. These technologies allow SDNs to operate reliably when traffic increases beyond the levels initially forecast. By deploying an SDN with these technologies over an extensive network, the infrastructure can quickly recover from disasters or other network failures while performing regular network operations.

To our knowledge, there are very few performance studies of SDN/OpenFlow networks on the analytical model. Our objective throughout this article is to study and evaluate controllers' loads and establish a load balancing if necessary.

Our approach, is totally based on the distributed architecture of the controllers. With the help of our algorithm, each controller collects its own loads, and is informed about the topology and loads of other controllers. In this way, the controller can make decisions by making sensible choices in the direction of high fault-tolerant, fluctuating load availability.

In the second part of this article we present the centralized and distributed architecture of the SDN. We expose their modes of operation. The third part is devoted to the presentation of our model of analysis and load balancing. Finally, an evaluation of our solution is presented followed by a conclusion.

2. Centralized and Distributed Architecture of SDN Network

Two categories of control plan architectures have been proposed in the literature

[9] and [10]. The first category uses a logically centralized control plan, where all the controllers work together to function as a single centralized controller with boosters within a super-controller. Thus, the super-controller has a global view of all controllers. However, this approach may have limitations. The performance of a centralized node is limited by memory, processor power, and bandwidth. In addition, a centralized node collects load information periodically and exchanges many messages frequently with other controllers, which will lead to reduced performance of the entire system [11]. In addition, if the central node collapses, the entire load balancing strategy falls.

The second category consists in using logically distributed controllers for where each controller has only one view of the domain for which it is responsible for and shares the necessary information with the other controllers [12]. In this case a controller is not only an ordinary controller but also a super-controller. In the rest of this article, we will study the performance and expose an analytical method of load balancing in the case of an architecture having distributed controllers.

Logically Distributed SDN Control Plan Architecture

In this context, the second category of logically distributed control plan is proposed to extend SDN on large multi-domain networks such as WAN networks. It allows each controller to have a view of the domain for which it is responsible for. It can make decisions for this area and communicate its information to other controllers. As WANs are characterized by high cost and latency due to the complexity of the infrastructure and protocol that handle traffic (BGP and MPLS), communication between controllers is of paramount importance in SDN. Several implementations have been dedicated to this category and its deployment is certainly beneficial for load balancing, fault tolerance, security, performance and scalability.

In the SDN with a distributed architecture, the controller mainly exchanges with the application plan (NorthBound), between controllers (Wests and East bound) and the data plan (Southbound). These exchanges generate queues that it is important to analyze in the consideration of load balancing. This system can be seen as a multi-class and uni-controller queue, each class of requests corresponding to one of the n possible assignments.

We focus our work on queries exchanged between APIs, controllers and switches. The goal is to evaluate the load of a controller.

We propose the service mode as the communication mode used between the controllers. The service mode allows the synchronization of information about flow tables, metric tables, link bandwidth, link status, and rules applied to other controllers. It is beneficial for applications that require a global view of the entire network and a certain quality of transmission. This mode is intended to improve the quality of service, security, routing, etc.

The orchestration and coordination of the control plan are provided by the

API Jgroups [13] and Zookeeper APIs [14]. Indeed, JGroups is a perfect support for remote procedure call (RPC) between controller nodes, while ZooKeeper is a centralized service for managing configuration information, names, distributed synchronization, and service provisioning.

3. OpenFlow Controllers Operation

3.1. Message Flow Management between Switch and Controller

The architecture of the flow control model that we propose is illustrated in Figure 1. In this architecture, the relationship between controllers and switches is multiple, which is supported by OpenFlow 1.5.1. The switch has multiple input/output ports and multiple ports providing control communication with multiple SDN controllers.

Figure 2 illustrates the flow input structure on a switch. The data flows consist of IP packet flows. It may be necessary for the routing table to define inputs based on higher layer protocol header fields, such as: TCP, UDP, SCTP, or application protocol.

3.2. Entry of a Switch Flow Tables

The switch flow tables contain a set of stream entries that show the rules and packet routing actions dictated by the controller. A feed entry is composed among others of:

- *match fields*: match fields that define the packet flow model through the instantiation of the header fields from the Ethernet layer to the Transport layer;
- *counters*: updated when packets are matched and counters on packets.;
- *priority*: matching precedence of the flow entry;
- *cookie*: opaque data value chosen by the controller. May be used by the controller to filter flow entries affected by flow statistics, flow modification and flow deletion requests. Not used when processing packets.

Counters can be maintained for each flow table, stream entry, port, queue

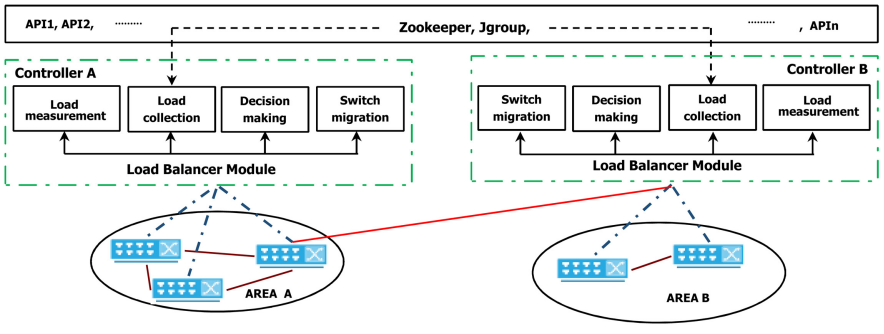

Figure 1. Distributed controller architecture.

Match Field	Priority	Counters	Instructions	Timeouts	Cookie	Flags

Figure 2. Main components of a flow entry in a flow table.

Wait. The description of the counters field gives the number of packets transmitted on this queue (Transmit Packets), the number of bytes transmitted on this queue (Transmit Bytes), the number of packets transmitted on this queue and those rejected for lack of memory on the queue (Transmit Overrun Errors).

When a counter reaches its maximum value, it returns to 0 without further indication. If a counter is not available, its value must be set to (−1).

OpenFlow1.3 introduces counters into the OpenFlow protocol. The counters complete the queue framework already in place in OpenFlow by allowing the monitoring of the input rate of data processed by the controller. Specifically, with counters, we can monitor the traffic input rate as defined by the **Figure 2** structure. Flows can direct packets to a counter using the OpenFlow goto-meter statement.

3.3. Message Flux Management between Controllers

Flux management is shown schematically in **Figure 3**. We propose an analytical flow model based on the mathematical notion of queue M/M/1/K. J Groups comes with a large number of protocols including FIFO (First In First Out), Total Order and Flow Control to prevent slow receivers from being overloaded by fast senders. We will use the Zookeeper API [14] to coordinate load balancing operations and then check if the migration has been successful with corollary load balancing.

The proposed load balancing mechanism is a module implemented at each SDN controller. It is called load balancing module. This module has four components:

1) *load measurement*: measures the loads and judges if the controller load exceeds the threshold. Note and respectively the load and the predefined load threshold of a controller;

2) *the equilibrium decision*: makes load balancing decisions or not based on the result of the load measurement;

3) *the load information collector*: receives and then organizes information on its load and sends it to other controllers;

4) *switch migration*: is responsible for putting one switch under control to

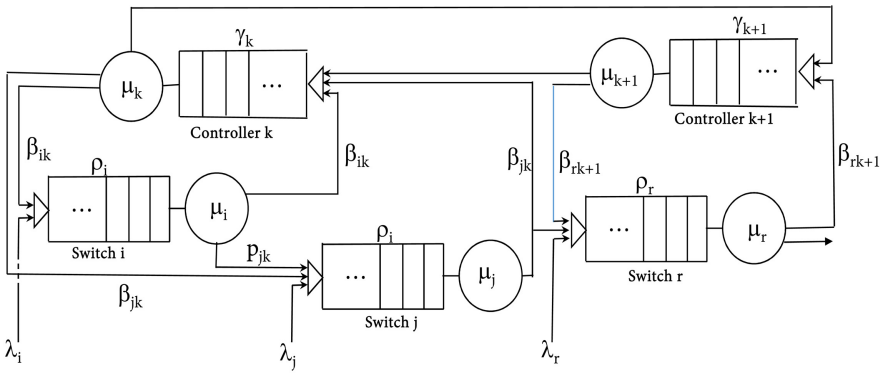

Figure 3. Model of queue between controller and switch.

another less loaded controller to balance the load.

The load balancing modules of the controllers cooperate with each other to ensure load balancing by coordinating the activities of the components. Thus, the measurement of the load periodically measures the loads of the controllers. There are three types of information that are collected locally and disclosed to other controllers:

- *Normal load* ($\frac{1}{2}\Gamma_{\max} < \Gamma_k < \Gamma_{seuil}$): packet transmissions are done correctly. However,
- the controller is not able to receive switch migration the value of its decision is (0);
- *under load* ($\Gamma_k < \frac{1}{2}\Gamma_{\max}$): the packet transmissions are done correctly and is able

To receive a switch migration. The value of it decision is (1);

- *Overload* ($\Gamma_k \geq \Gamma_{seuil}$): packet transmissions may be disturbed shortly. The probability of rejection of packets in the queue is high. The value of the decision is (-1).

In the following section, we propose an analytical method of the load balancing described above. This assumption also allows the use of aggregation and division of Poisson processes to determine the intensity of the arrival packets on each node. Finally, it is assumed that the controller has a complete knowledge of the network topology, and can therefore, configure the nodes for optimal packet transfer.

In [15] the authors presents a method for efficient adjustment of traffic flows to achieve load balance among multiple controllers using three modules comprising a load collector, a load balancer, and switch migrater. We propose in our approach, a fourth module that of the decision of balancing and migration of switches. This makes it possible to control migrations without overloading the less loaded controllers during the migration.

4. Load Balancing Model Description

4.1. Data Plan Level Model

The data plan system model is considered to be an open Jackson network of J switches, $j = 1, 2, \cdots, J$ and a single controller as shown in **Figure 3**. In addition, we assume that the rate of arrival in the data plan with a parameter λ_j and the probability that a packet goes from switch j to the controller C_k is ρ_{jk}. The probability of routing the switch r to the switch j is ρ_{rj}. In addition, the service rate in the switch j is μ_j, while it is μ_k for the controller. According to Jackson's theorem for open queuing networks, all queues of switches behave locally like queues M/M/1/K with load $\rho_j = \frac{\gamma_j}{\mu_j}$.

At controllers level, the arrival in the queue is: $\Gamma_k = \sum_{j=1}^{K} p_{jk}\Gamma_j$ the load of the

controller is therefore $\Delta_k = \frac{\Gamma_k}{\mu_k}$.

The balancing equation of the M/M/1/K system under the FIFO discipline is given by the formula for the input γ_j to node j we have in [16]:

$$\Delta_j = \gamma_j + (1 - P_b)\left(p_{ik}\lambda_i + \sum_{j=1, j\neq m}^{K} \left(p_{mj}\xi_m \right)\lambda_j \right) \tag{1}$$

The knowledge of P_b (packet blocking capacity), $E(N_c)$ (the average number of packets in the controller) and $E(T_c)$ (the waiting time of the packet in the queue) allows to calculate Δ_j.

4.2. Model at Controller Plan Level

The model defined in the control plan consists of several controllers, where each would be responsible for a portion of the network. However, each switch is managed by only one controller. In its operation the controllers detect the neighboring controllers and also the edge switches which are located at the edge of a network portion as shown in **Figure 1**.

The purpose of this section is to evaluate the load of a controller and decide to migrate one or more switches to other less loaded controllers. We consider a simple controller system identical to the queue M/M/1/K whose capacity of the queue is finished. When a packet arrives in the system when there are already K packets in the queue it is rejected and lost and has no influence on the system ("lost customer cleared"), which explains why there is no a priori hypothesis on the load $\rho = \lambda T$. Clients are served according to the FIFO discipline, the service life being constant and equal to T units of time and controller load C_k is: $\rho_k = \frac{\lambda_k}{\mu_k} < 1$ when $k = 1, 2, \cdots, K$

The distribution of the number of C_k controller packets is given by the Equation (2):

$$P_k(n) = P_k(X = n) = \frac{1 - \rho_k}{1 - \rho_k^{K+1}} \rho_k^n \tag{2}$$

The average number of packets in the controller C_k is given by the Equation (3):

$$\Gamma_k = E(N_k) = \sum_{k=0}^{K} kP(k) = \frac{\rho}{1 - \rho^{K+1}} \left[\frac{1 - \rho^{K+1}}{1 - \rho} - K\rho^K \right] \tag{3}$$

and the probability of blocking, therefore the rejection is given by the formula (4) below :

$$P_b(n) = \frac{1 - \rho_k}{1 - \rho_k^{K+1}} \rho_k^K \tag{4}$$

Our approach corresponds to the scenario where the buffer space in the controller is limited to at most K packets in the queue. Applying the formula of Little expressing the residence time of the packet in the queue of the controller we

have the equation:

$$E(T_k) = \frac{E(N_k)}{\Gamma_k(1-P_b)} \tag{5}$$

There is rejection if and only if the queue with limited capacity is full. We can express K according to ρ and P_k:

$$K = \frac{\ln\left(\frac{P_b}{1-\rho+P_b\cdot\rho}\right)}{\ln(\rho)} \tag{6}$$

Formula (6) gives the capacity of the controller queue. Indeed, it makes it possible to evaluate the maximum number of packets in the queue according to the probability of blocking and the rate of charge.

Figure 6 gives the K capacity of a controller when the probability of rejection is less than 10^{-n}. It has been assumed that the messages form a fish traffic of variable load ρ. The assignment of tasks to the machines is decided during the execution phase, acdepending on the information that is collected on the state of charge of the system. This makes it possible to improve the execution performance of the tasks but at the cost of a complexity in the implementation of this strategy. It is necessary to know the load of the switch to be migrated, then select the controller whose load would be below the threshold after the migration. It is assumed that the maximum load of a controller is $\Gamma_{\max}$. We note the threshold load between $\frac{1}{2}\Gamma_{\max}$ and $\Gamma_{\max}$. Thus in algorithm 1 of Figure 4, the load balancing module makes decisions against a controller based on its average load.

The load measurement component returns to values 0, 1 or (−1) according to the received load measurements. The decision component of the controller k, decides to alert the other controllers in the event that the Status Controller (k) returns to value (−1). The Migration function $\left(\omega_i^k, j\right)$ of switch i of the controller k towards controller j is facilitated by the classification of the load of the switches.

Collection function (Δ_k) Controller k collects load information and ranks it

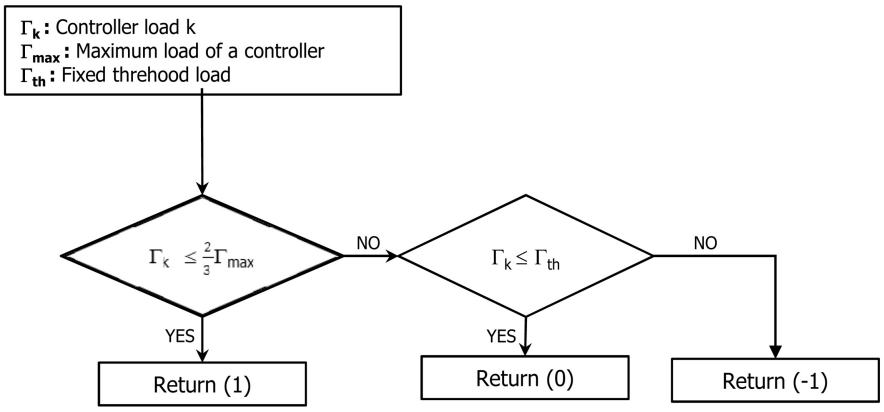

Figure 4. Algorithm 1: Controller load measurement.

in ascending order. Thus, $\text{Collecte}(\Delta_k) = \bigcup_{k=1}^{m} \left\{ \Gamma_k, \left\{ \omega_l^k < \omega_r^k < \cdots < \omega_s^k \right\} \right\}$ gives information about the total load of the switch k and the load of the switches of its stored area in ascending order. Below is the switch migration algorithm. Algorithm 1 can execute with a linear time complexity, *i.e.*, in $O(n)$ time.

It is important to note that the selection of switches to migrate is delicate because other overloads and oscillations between switches and controllers must be avoided. To avoid that, the load collector with the controller's load information identifies the candidate controllers to receive a migration.

Then, algorithm 2 of Figure 5, refines the choice by determining the appropriate switch. If the switch has migrated to the target controller (successful migration process) Zookeeper will return 1. Otherwise, it will raise an exception.

Finally, after completing the migration, the controller updates its load information and informs the other controllers via the load information component. Finally, after completing the migration, the controller updates its load information and informs the others. With the help of Zookeeper, we can change the role of the master node and the slave node. To ensure high availability and lossless message delivery during migration, we use Highly Available TCP (HA-TCP). The migrating switch module is assisted in its task by Zookeeper. After completing the migration, the controller updates its load information and reports to the other controllers via the load information component. We can note than, the time complexity of algorithm 2 is O (n^2). In the following section we evaluate our queueing model with different parameters by numerical analysis. Suppose packets arrive at all the switches with same rate λ, the rate at which the switches forward packets to the controller and output ports is 32 K packets per second and 64 K packets per second.

Switch Migration Algorithm: **SwitchMigration( )**

Input

Γ_k : k=1,2,.....M ; the average load of the controller k.
Collect( ) : switch load and switch loads, sorted in ascending order
Statut Controller ( ) : Indicates whether the switch is normal, overloaded, or underloaded
SwitchMigration ( ) : Migration function of a switch to a less loaded controller
SwitchElection ( ) : eligible Switch to migrate

Output

```
For any controller k=1, 2, ..., m
  For any switch i=1, 2, ..., n
      If Statut switch(k) = -1  and Statut switch (k) = 1  Then
          If  min_i(Γ_j + ω_i^k) ≤ Γ_th  Then
              i = SwitchElection (k)
               SwitchMigration (ω_i^k, j)
            Else
              Collect(Δ_k)      /* Can not migrate a switch
          EndIf
      EndIf
  EndFor
EndFor
```

Figure 5. Algorithm 2: load balancing.

5. Results Analysis

The implementation of the balancing prototype is based on the Floodlight controller [17]. It is a Java based OpenFlow controller of the Beacon SDN controller developed by Stanford. That controller is an open-source software manufactured by Apache, supported by a community of software developers. It offers a modular architecture, easy to expand and improve. We add the TCP-HA protocol and the storage module for the synchronous state fault detection.

Figure 6 shows that the number of switch supported by a controller depends on its load ρ. Cbench application is a performance measurement tool designed to compare packages processed by OpenFlow controllers. It can simulate packets from OpenFlow controllers. Mininet is an OpenSources tool for emulating SDN networks. The mapping table has been stored on the Zookeeper servers. It can support, store services with strong consistency so that the result of the choices of switch to migrate is consistent with the expected results. During the experiment were used three distributed Floodlight controllers likewise having a charge rate ρ of 0, 8. The modules for load measurement, load collection, decision and switch migration are implemented on each controller. All controllers are implemented so that as soon as they reach 80% of their maximum load, an alert is issued by the decision component of the load balancing module.

Figure 7 shows that there is switch migration at controller level (C1) towards (C3). In fact from 4.4 ms the curve (C1) decreases while the curve (C3) takes more load than expected. Some problems may occur during the experiment:

1) all controllers reached the 80% threshold of Γ_{max};

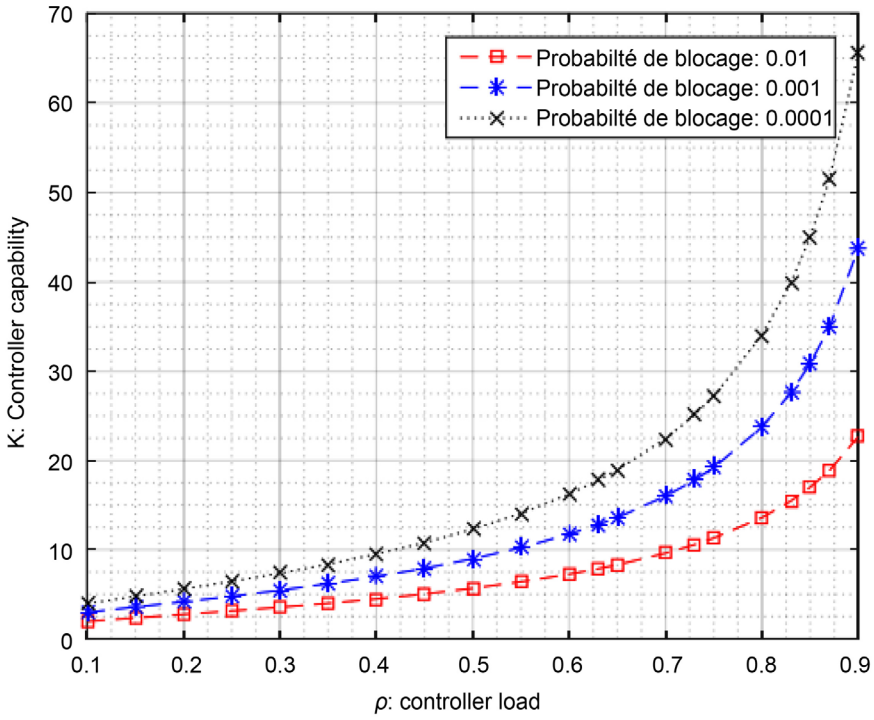

Figure 6. Average number of packets in the queue.

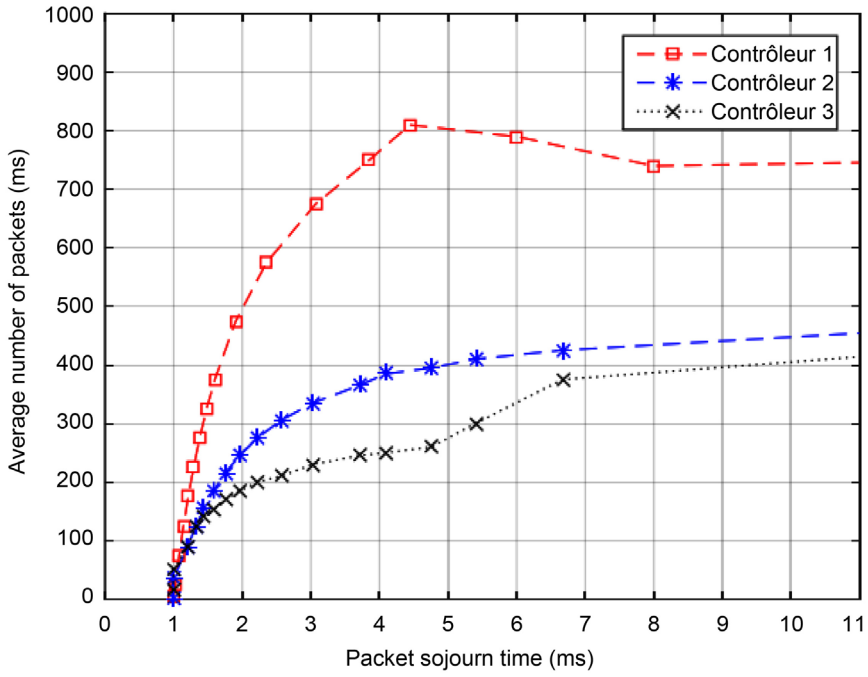

Figure 7. Balancing test result.

2) two controllers that reached the 80% max threshold could migrate their switches to the same controller. This will cause saturation of the target controller.

In case (1), it is assumed that all controllers have reached the fixed threshold. In this case, it is necessary to anticipate a Floodlight controller without any load beforehand which will be operational as soon as all the controllers exceed the 80% of their load.

In case (2) we could choose in algorithm 2, the choice to migrate in priority the switch of the controller with the highest load. Thus, when two controllers have exceeded the 80% threshold, the one with a maximum load will be prioritized for switch migration.

We notice that in our model, the controllers store a lot of information related to the other SDN controller. However, the above results show the performance of proposed load balancing algorithm by using analytic model. Thus, we got a load balanced overall costs of the network.

6. Conclusions

In this article, we propose a load balancing strategy for the SDN controller based on distributed decision. An analytical queue model that responds to this strategy is exposed. We describe each component of the SDN architecture, in particular the load balancing module. The uneven distribution of controllers' load is an inevitable problem. We proposed a load balancing algorithm. The results of the evaluation showed that our mechanism can achieve two objectives: to anticipate controllers' overloads on the one hand and to balance the load of the system

controllers on the other hand.

Our future direction is to compare the results obtained from the proposed model with hardware implementations which will lead to interesting results.

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

References

[1] Benamrane, F., Ben Mamoun, M. and Benaini, R. (2015) Performances of Openflow-Based Sofware Defned Networks: An Overview. *Journal of Networks*, **10**, 329-337. https://doi.org/10.4304/jnw.10.6.329-337

[2] Liu, H.H., Wu, X., Zhang, M., Yuan, L., Wattenhofer, R. and Maltz, D. (2013) Updating Data Center Networks With Zero Loss. *ACM SIGCOMM Computer Communication Review*, **43**, 411-422. https://doi.org/10.1145/2534169.2486005

[3] Mahmood, Chilwan, A., Østerbø, O. and Jarschel, M. (2014) Modelling of OpenFlow-Based Software Defined Networks: The Multiple Node Case, Kashif. *IET Journal*.

[4] Open Networking Foundation (ONF) (2015) OpenFlow Switch Specification; Version 1.5.1 (Protocol Version 0x06), ONF TS-025. https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf

[5] Hock, D., Gebert, S. and Hartmann, M. (2014) POCO-Framework for Pareto-Optimal Resilient Controller Placement in SDN-Based Core Networks. *IEEE Network Operations and Management Symposium* (*NOMS*), *IEEE Xplore*, 1-2.

[6] Mattos, D.M.F., Duarte, O.C.M.B. and Pujolle, G. (2016) A Resilient Distributed Controller for Software Defined Networking, Communications (ICC). *IEEE*.

[7] Nunes, B.A., Mendonca, M. and Nguyen, X.-N. (2014) A Survey of Sofware-Defned Networking: Past, Present, and Future of Programmable Networks. *IEEE Communication Survey and Tutorial,* **16**, 1617-1634. https://doi.org/10.1109/SURV.2014.012214.00180

[8] Ros, F.J. and Ruiz, P.M. (2016) Reliable Controller Placements in Sofware-Defned Networks. *Computer Communications*, *Sciencedirect*, **77**, 41-51.

[9] Abdelaziz, A., Fong, A.T. and Gani, A. (2017) Distributed Controller Clustering in Software Defined Networks. *PLOS ONE*. https://doi.org/10.1371/journal.pone.0174715

[10] Gardner, K., Harchol-Balter, M., Scheller-Wolf, A., Velednitsky, M. and Zbarsky, S. (2017) Redundancy-D: The Power of D Choices for Redundancy. *Operations Research*. https://doi.org/10.1287/opre.2016.1582

[11] Paris, S., Destounis, A., Maggi, L., Paschos, G. and Leguay, J. (2016) Controlling Flow Reconfigurations in SDN. *The* 35*th Annual IEEE International Conference on Computer Communications*, San Francisco, 10-14 April 2016. https://doi.org/10.1109/INFOCOM.2016.7524330

[12] Guo, Z.H., Su, M., Xu, Y., Duan, Z.M., Wang, L., Hui, S.F. and Chao, H.J. (2014) Improving the Performance TF Load Balancing in Software-Defined Networks through Load Variance-Based Synchronization. *Computer Networks*, **68**, 95-109. https://doi.org/10.1016/j.comnet.2013.12.004

[13] Shen, D., Yan, W., Peng, Y., Fu, Y. and Deng, Q. (2018) Congestion Control and Traffic Scheduling for Collaborative Crowdsourcing in SDN Enabled Mobile Wireless Networks. *Wireless Communications and Mobile Computing*, **2018**, Article ID: 9821946. https://doi.org/10.1155/2018/9821946

[14] Yu, J., Wang, Y., Pei, K., Zhang, S. and Li, J. (2016) A Load Balancing Mechanism for Multiple SDN Controllers Based on Load Informing Strategy. *The* 18*th Asia-Pacific Network Operations and Management Symposium*, Kanazawa, 5-7 October 2016.

[15] Wang, K.-Y., Kao, S.-J. and Kao, M.-T. (2018) An Efficient Load Adjustment for Balancing Multiple Controllers in Reliable SDN Systems. *IEEE International Conference on Applied System Invention*, Chiba, 13-17 April 2018, 593-596.

[16] Jackson, J.R. (1957) Networks of Waiting Lines. *Operations Research*, **5**, 518-521. https://doi.org/10.1287/opre.5.4.518

[17] Bholebawa, I.Z. and Dalal, U.D. (2017) Performance Analysis of SDN/OpenFlow, Controllers: POX versus Floodlight, Wireless Personal Communications. Springer Science + Business Media, Berlin.