Cooperative Load Balancing for Hierarchical SDN Controllers

Hakan Selvi, Gürkan Gür, and Fatih Alagöz
SATLAB, Dep. of Computer Engineering, Bogazici University, Istanbul, Turkey
Email: {hakan.selvi, gurgurka, fatih.alagoz}@boun.edu.tr

***Abstract*—Software-defined networking (SDN) has been proposed as a new networking paradigm which separates the control and data forwarding planes. The controllers deployed as centralized network control entities are crucial for meeting the performance and efficiency requirements of these systems. The placement of these network entities, switch assignment and their load-aware operation are challenging research questions. Although various load-balancing schemes are proposed in the literature, this research topic is yet to be explored adequately. In this work, we discuss load-balancing issues in SDN and propose a load-balancing scheme, namely *Cooperative Load Balancing Scheme For Hierarchical SDN Controllers (COLBAS)*, for hierarchical controller configurations. Moreover, we evaluate the performance of our scheme and investigate important performance factors.**

I. Introduction

Today's networks have become exceedingly complex with increasing number of distributed protocols and specialized forwarding elements using closed and certified software stacks [1]. This traditional networking architecture lacks the inherent traits for scalability, flexibility, and programmability. To address these issues and facilitate Future Internet services, a new networking paradigm, called Software-Defined Networking (SDN), has been proposed [2]. SDN is devised to simplify network management and enable innovation through network programmability. In the SDN architecture, the control and data planes are decoupled and the network intelligence is logically centralized in software-based controllers. An SDN controller provides a programmatic interface to the network, where applications can be written to perform management tasks and offer new functionalities. An SDN forwarding device contains one or more flow tables consisting of flow entries, each of which determines how a packet performs [1] and the controller updates these flow tables and instructs the switches on which actions they should take via a programmatic interface called Southbound interface [3] as shown in Fig. 1. For instance, an OpenFlow Controller, being the controller type for the most common SDN protocol, sets up OpenFlow devices in the network, maintains topology information and monitors the network status [4].

SDN paradigm provides not only facilitation of network evolution via centralized control and programmability by enabling deployment of third-party applications, but also elimination of middle-boxes and rapid depreciation of network devices [5]. The controller related aspects of the software-defined network are paramount for addressing the emerging intricacies of SDN-based systems [1]. Therefore, although SDN paradigm will facilitate new degrees of freedom for traffic and resource management, it will also bring forth profound issues such as security, system complexity, and scalability. In that respect, the controller load-balancing related challenges also emerge as critical factors on the feasibility of SDN [6]. In this work, we elaborate on load-balancing issues in SDN and propose a load-balancing scheme, namely *Cooperative Load Balancing Scheme For Hierarchical SDN Controllers (COLBAS)*, for hierarchical controller configurations.

The paper is organized as follows: The next section describes the multiple controller architecture and load balancing in SDN. Moreover, we briefly introduce related works. The design and implementation of COLBAS is explained in Section III. In Section IV, we present experimental results and investigate effects of varying parameters on system mechanism. Finally, we conclude the paper in Section V.

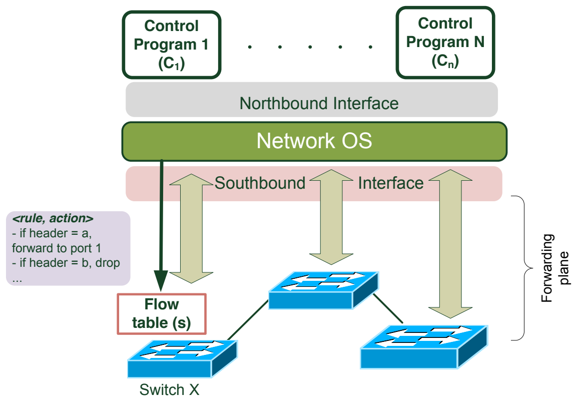


Fig. 1. SDN control and Rule/Action approach.

II. Multiple Controllers and Load Balancing in SDN

Since the control is centralized and applications are written as if the network is a single system, policy enforcement and management tasks are simplified in SDN [7]. The outcome of experiments in [8] shows that a single controller has capability to manage excessive number of new flow requests in an unexpected manner. However, in a large-scale network deployment, this centralized approach has some limitations related to the interaction of control and forwarding elements, response

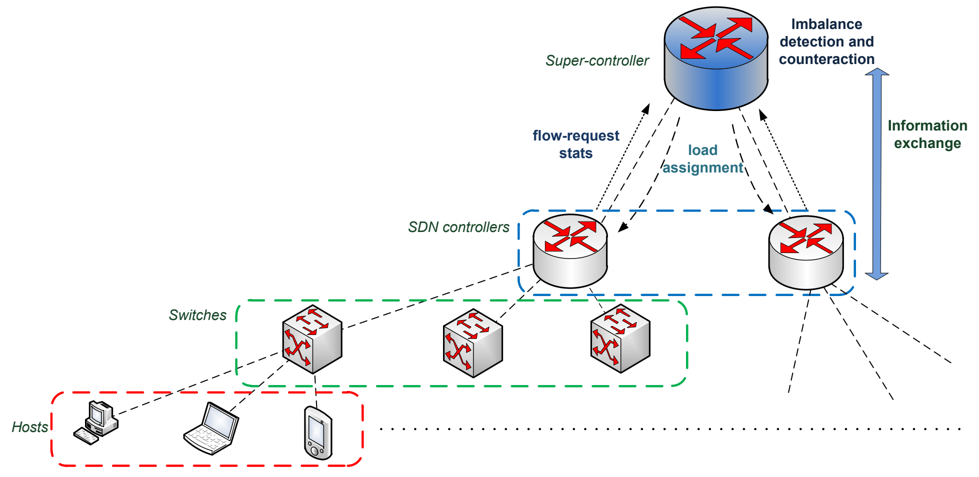


Fig. 2. System view.

time, scalability, infrastructure support and availability [9]. Typically, large amount of network flows originating from all infrastructure nodes cannot be handled by a single controller because of the limited capacity and resource constraints [10]. Another work by Voellmy et al. supports this claim and shows multiple controllers ensure high fault tolerant networks with reduced latency [11].

For networks with multiple controllers, a controller may become overloaded if switches mapped to this controller have large number of flows. However, other controllers may remain underutilized. It is better to shift load across controllers over time depending on the temporal and spatial variation in traffic conditions. For instance, if the nodes are assigned to nearest controller using latency as a metric or shortest path distance between node and controller, there may be times that some controllers are overloaded due to traffic flow. There can be an imbalance in the number of nodes per controller in the network. The higher the number of nodes attached to a controller, the greater the load on that controller. The increase in number of node-to-controller requests in the network induces additional delay due to queuing at the controller system. Therefore, one must determine measures against excessive controller loads and the network must be reconfigured before an overload or failure situation occurs.

Load balancing in computer and network systems can be traced back to the very early computing research since the utilization and response characteristics of different computing and communicating machinery were to be optimized for different workloads in a spatiotemporal setting. Load-balancing in SDN networks can be achieved by utilizing different SDN concepts. To handle most of the traffic in data plane is one approach as opposed to making decisions on control plane, which is more suitable to the nature of SDN. Yu et al. propose DIFANE which is a scalable and efficient solution that keeps all traffic in the data plane. It selectively directs packets through intermediate switches that store the necessary rules [12] and thus assigns the controller to the less complex task of partitioning these rules over the switches. Similarly, DevoFlow is a modification of the OpenFlow model which breaks the coupling between control and global visibility, in a way that maintains a useful amount of visibility without imposing unnecessary costs [13]. In that work, Curtis et al. find that DevoFlow uses drastically fewer control messages and fewer flow table entries at an average switch.

HyperFlow is the first example of the distributed event-based control plane application which is implemented for the original OpenFlow controller, NOX, by minor modifications [14]. In network-wide controller deployments, since all the controllers need a consistent network-wide view and run as if they are controlling the whole network, it has limited use for local control applications. Kandoo [15] and BalanceFlow [16] consist of two controller layers distinct from an ordinary OpenFlow network. In the former, most of the frequent events are handled on lower layer controllers of the hierarchical structure and higher layer maintains network-wide state. In the latter, the super-controller processes the flow information collected from controllers and detects load imbalances, then acts upon it.

III. Cooperative Load Balancing Scheme For Hierarchical SDN Controllers (COLBAS)

COLBAS is a controller load-balancing scheme for hierarchical networks. It relies on controller cooperation via cross-controller communication. It assigns one of the controllers as a super controller which can flexibly manage the flow requests handled by each controller. It is located at the network node which is closest to the geographical center of the topology. As it is illustrated in Fig. 2, all controllers work with their own load information and publish it periodically through a cross-controller communication system. When traffic conditions (e.g. load surge) change, super controller reassigns different flow setups to proper controllers and installs allocation rules on switches for load balancing.

The SDN environment that we consider is shown in Fig. 2. It has hosts connected to switches located at nodes on the modified Abilene topology shown in Fig. 3. The switches are

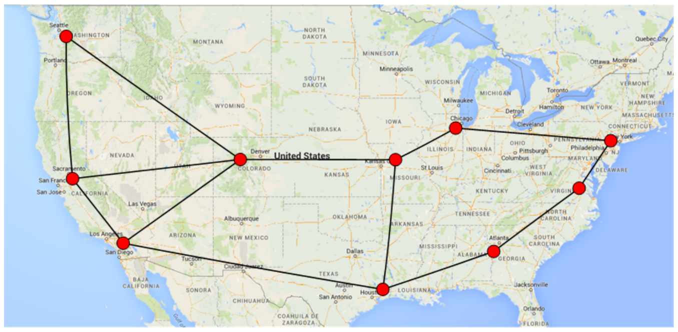


Fig. 3. Modified Abilene topology.

connected to controllers which serve as SDN controllers for these nodes. There is a *super-controller* which is connected to the lower-tier controllers. This hierarchical structure works in a cooperative manner which entails information exchange for load metrics and partitioning as shown in the figure. The hosts connected to switches generate flow requests according to a Poisson process where packet interarrival times are exponentially distributed. The packet sizes are assumed to be identical. The flows between switch pairs constitute flow-request information. For these data, each controller stores an $S \times S$ matrix, where S is the number of switches in the network, i.e. $S = \| \mathbb{S} \|$. The element in the ith row and jth column $\widetilde{B}_{ij}$ corresponds to the average number of flow requests from switch i to switch j. It is calculated according to

$$\widetilde{B}_{ij} = z\widetilde{B}_{ij}^{(-2)} + w\widetilde{B}_{ij}^{(-1)} + (1 - w - z)R_{ij} \tag{1}$$

where w and z are weight coefficients to avoid sharp up and downs by taking into consideration past values (smoothing), and R_{ij} is the number of flow requests from switch i to switch j in the last T_c, which denotes the controller period. The super controller collects the calculated matrices from all controllers and tries to detect imbalance if average number of flow requests handled by any controller exceeds a mutual threshold of the flow requests rate per controller pair, namely L_{upper}, in the network. Once the system is imbalance state, COLBAS redistributes the flows to reach a lower threshold L_{lower}. This counteraction continues until the load on overloaded controller falls below L_{lower}. When that threshold is crossed towards lower values, the system is out of imbalance state. The mechanism again waits until L_{upper} is exceeded in order to activate. These upper and lower threshold parameters must be tunable according to the performance of the super controller, the number of controllers and the network environment. This operation is illustrated in Fig. 4.

For reassignment and allocation, we need to keep the system performance high and the reassigning cost low. Therefore, we use a cost-based greedy algorithm after imbalance detection similar to [16]. The objective is to keep the total number of flow requests handled by any controller below the given threshold. After the switch pairs are sorted in descending according to their $\widetilde{B}_{ij}$ values, the cost function is calculated for each pair as follows:

$$C_s = \overline{Q_s} + P_{ij}^s + \alpha v_M D_{is} \tag{2}$$

where $\overline{Q_s}$ is the average number of flow requests already handled by controller s in last two T_s periods, P_{ij}^s is the number of flow requests that is about to be handled by that controller and calculated by projecting $\widetilde{B}_{ij}$ on to the period of super controller's period, and D_{is} is the delay from switch i to controller s. $\overline{Q_s}$ and P_{ij}^s are calculated according to following formulas:

$$\overline{Q_s} = \frac{Q_s^{(-1)} + Q_s^{(0)}}{2} \tag{3}$$

$$P_{ij}^s = \widetilde{B}_{ij}\frac{T_s}{T_c} \tag{4}$$

Parameter v_M is a variable that makes controllers' load and propagation delays comparable, and $v_M = R_{total}/D_{avg}$ where R_{total} is the total number of flow requests in the network, and D_{avg} is the minimum average node-to-controller and controller processing latency and calculated as

$$D_{avg} = D_{min}^t + D_{min}^p \tag{5}$$

Parameter α is the variable to adjust the weights between controllers' load and propagation delays in the cost function. A "too small" α represents that the benefit of reducing latencies is ignored and related pair is reassigned to a controller regardless of delays. We can adjust it larger if we are more focused on packet delivery time than load-balancing. The delay D_{is} consists of propagation or transmission delay D_{is}^t and processing delay D_s^p:

$$D_{is} = D_{is}^t + D_s^p \tag{6}$$

The processing delay D_s^p for a flow packet in controller is calculated as the sum of the time spent in the controller processing and queue wait-time.

The load balancing algorithm sorts the switch pair flows in descending order if imbalance is detected and reassigns the flows to the lowest-cost controller till the lower threshold is expected to be met. After the decision step, the next action is to actuate the new flow rules in the switches. They are represented as low-level rules and sent to the associated switches as in [16]. Normal flow rules has priority over new allocation rules. If a flow-request does not match any normal flow entries, it tries to match an allocation rule entry to take action. Subsequently, the switches operate under these new flow rules leading to a better distribution of load.

IV. Performance Evaluation

This section explains the experiments and measurements done using the setup explained in Sec. III and discusses the findings. We developed a discrete-time event simulator based on SimPy [17]. SimPy is a process-based discrete-event simulation framework based on Python programming language. We have extended this environment with SDN related entities such as controllers and algorithmic components such as our load distribution mechanism.

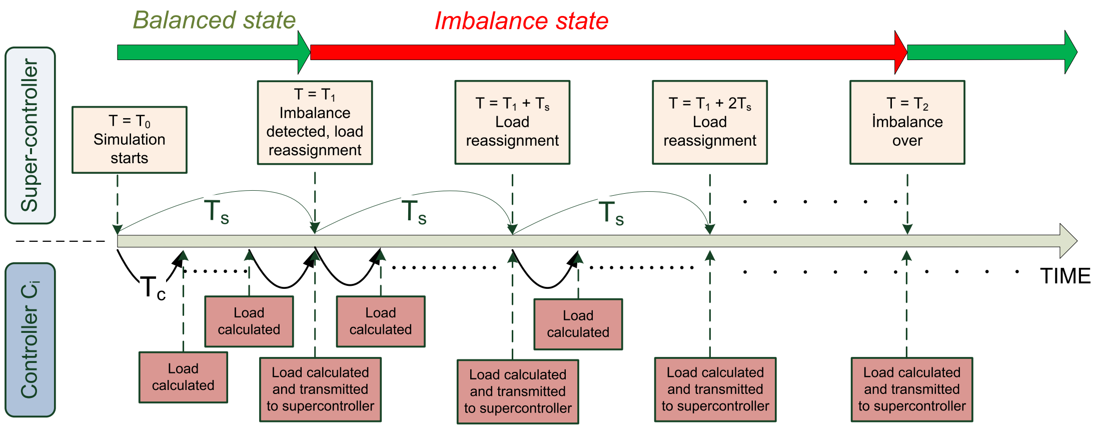

Balanced state
Imbalance state
Super-controller
Controller Ci
T = T0 Simulation starts
T = T1 Imbalance detected, load reassignment
T = T1 + Ts Load reassignment
T = T1 + 2Ts Load reassignment
T = T2 Imbalance over
Ts
Tc
TIME
Load calculated
Load calculated and transmitted to supercontroller

Fig. 4. The operation of super-controller and controllers for COLBAS operation. The node-based calculations and decisions/actions are depicted for different epochs. The imbalance occurs between T_1 and T_2.

TABLE I
PARAMETER VALUES

Parameter	Value
number of nodes (N_n)	10
number of hosts (N_h)	10
number of controllers (N_c)	3
number of switches (N_s)	10
controller period (T_c)	10 msec
super controller period (T_s)	1000 msec
timeout period for flow entry in rule table	6 msec
host traffic generation rate (exponential (λ))	50
weight of past B_{ij} at t=-1 (w)	0.3
weight of past B_{ij} at t=-2 (z)	0.2
coefficient for cost function (α)	0.05
imbalance detection upper threshold (L_{upper})	1
imbalance detection lower threshold (L_{lower})	0.7

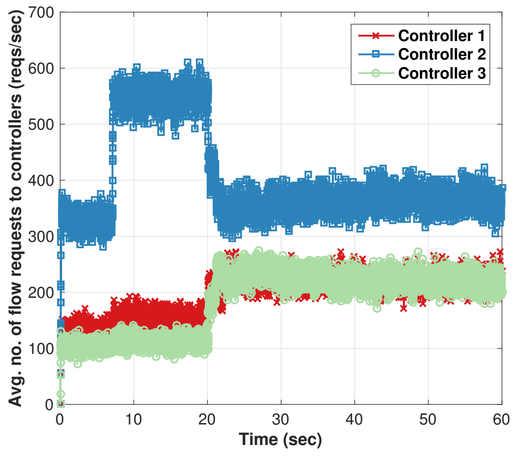


Fig. 5. Baseline results for parameter values in Table I.

A. Baseline System

In our experiments, we first set system and simulation parameters to the values shown in Table I and then we assign different values to L_{upper} and L_{lower}, delay sensivity parameter α, w and z weight coefficients of past values to investigate their effects. Finally, we compare COLBAS with two other algorithms, namely *random reassignment algorithm (RND)* which allocates switch pairs without considering cost of assignment or controller capacities and *nearest-neighbor algorithm (NEN)* which assigns related switch pairs to the nearest controller to mitigate imbalanced state.

The simulations are executed for a period which corresponds to 60 seconds in network operation time and each scenario is tested 45 times. There are three controllers (C1, C2, and C3) and a supercontroller in the network. C1 and C3 are each connected to three switches while C2 is connected to four switches. C2 also has two times more capacity than C1 and C3. Therefore, the number of flow requests passing through C2 is greater compared to other controllers as expected. At t = 7 sec.s, we suddenly increase the load of C2 by increasing host traffic generation rate as $\lambda = 20$. Then, to see how COLBAS detects load imbalance and acts to decrease this overload, we start supercontroller after the activation time (t = 20 sec.s). The super controller checks the loads of controllers each second. After it detects imbalance, it redistributes the load of C2 to other two controllers and achieves a balance state as shown in Figure 5. Overall, if one of the controller's load exceeds the determined threshold L_{upper}, its load is diverted until the excessive load decreases to the lower threshold L_{lower}.

B. Impact of Imbalance Detection Threshold ($L_{lower/upper}$)

Since the threshold levels play a significant role in the imbalance detection, we must be attentive to determine its optimal value for our network structure. Especially, in the case that all controllers have different capacities, we might need to assign different loads on different controllers and the threshold value is one of the key parameters to adjust these allocations. In this analysis, we focus on investigating how different thresholds affect the load distribution rather than determining optimal values for them. For Fig. 6, we decrease L_{lower} and L_{upper} to 0.5 and 0.8, respectively. Compared to Fig. 5, we observe that if we set thresholds lower, while time

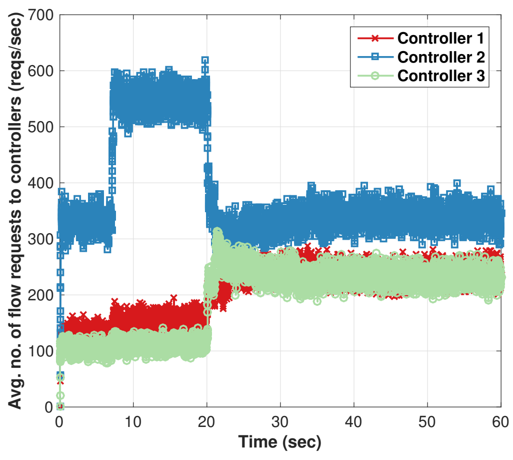


Fig. 6. Results for L_{upper}=0.8 and L_{lower}=0.5.

spent in imbalance state is longer, the loads of controllers converge closer. If we want to assign more burden on a specific controller (e.g. the one with higher capacity), we should configure these parameters appropriately.

C. *Impact of Weight Coefficients of Past Values* (w, z)

The flow-request information stored in each controller represents the load factor and is calculated by taking into account data from previous two periods as stated in (1). If we increase the contribution of past values by increasing w and z, we can achieve smoother transitions between load values. Fig. 7 illustrates the effect of these variables on the loads of controllers explicitly. As it is shown, loads in more retrospective systems are steadier and less volatile.

D. *Impact of Delay Sensivity Parameter* (α)

We change the α parameter for investigating flexibility to serve delay-sensitive traffic. As described above, α adjusts the weights between controllers' load and propagation delays in the cost function given in (2). If we do not consider the latencies critical and concentrate on balancing the controller loads, we must select small α values. If it is set too large, in the case of imbalance, any node will not change its current controller with another one and load balancing becomes less effective. To test the effect of this coefficient, we set it to three different values, specifically 0.5, 0.05, and 0.005. In that case, average propagation delay becomes 180.71, 209.35, and 264.27 mseconds, respectively. Briefly, we observe that the bigger the delay sensivity coefficient, the smaller packet delay and the more time spent in imbalanced state.

E. *Comparison with Other Reassignment Algorithms*

We compare COLBAS with the random (RND) and the nearest-neighbor (NEN) reassignment counterparts as baseline algorithms. Since any switch pair causing overload on any controller is reallocated to another randomly selected controller and to the physically closest controller without regarding its load and capacity in these algorithms, COLBAS is expected to perform better. Our tests verify this expectation.

If RND algorithm is applied, the network stays in imbalanced state almost ten times more in comparison to when COLBAS is used. Moreover, the load distribution is altered without considering the individual capacities of controllers. Although the load figures in Fig. 8(a) may seem like a good balancing situation, the load on C2 should stay higher since it has more capacity compared to C1 and C3. If the reassignment is applied with NEN algorithm, staying in imbalanced state decreases compared to random algorithm but it is still almost six times more compared to COLBAS. Since there are three controllers in our topology and C1 is closer to C2, the system burden oscillates between these two controllers. However, because of its high capacity, C2 can serve more load than C1. With these results, COLBAS performs better than RND and NEN algorithms.

V. Conclusions

The controllers which are centralized network control entities are crucial for software-defined networks satisfying the stringent performance and efficiency requirements imposed by surging data traffic. The controller-switch assignment and their load-aware operation are challenging research questions for these systems. In this paper, we discuss load-balancing issues in SDN and propose COLBAS consisting of hierarchical and heterogeneous SDN controllers for load balancing of control traffic. Moreover, we evaluate the performance of this scheme and investigate important trade-offs and effects of system parameters.

As future work, additional dynamic factors can be considered such as flow table size and queue sizes in network entities. We may also consider diversity of flow requests: some may be easy to process while some are difficult. Moreover, we may take link/node failure and more complex imbalance possibilities, i.e. multiple overloaded controllers, into account to consider more realistic scenarios. As an additional solution, a three- or more-layered controller structure might be formed in the case that the two-layer structure cannot satisfy the load requirements in the network.

VI. Acknowledgment

This work is supported by the Celtic-Plus and TÜBİTAK TEYDEB (project no. 9130038) project SIGMONA.

References

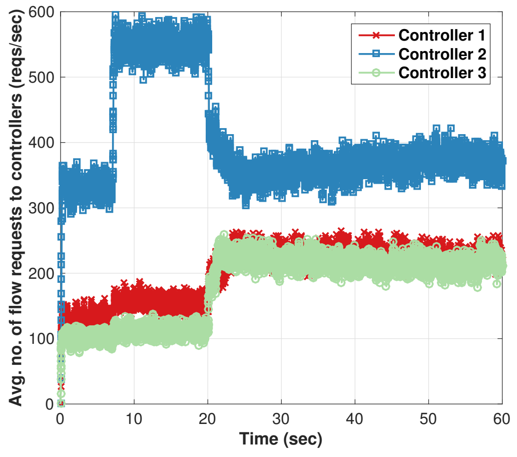
[1] B. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, and T. Turletti, “A survey of software-defined networking: Past, present, and future of programmable networks,” *Communications Surveys Tutorials, IEEE*, vol. 16, no. 3, pp. 1617–1634, Third 2014.

[2] Open Networking Foundation (ONF), “Openflow specifications,” 2014. [Online]. Available: https://www.opennetworking.org/sdn-resources/openflow/, January 2014.

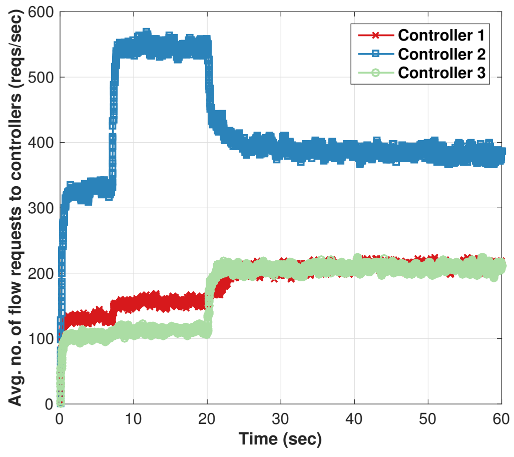
[3] NEC, “Ten things to look for in an SDN controller,” 2014. [Online]. Available: https://www.necam.com/Docs/?id=23865bd4-f10a-49f7-b6be-a17c61ad6fff, September, 2014

[4] M. Fernandez, “Comparing OpenFlow controller paradigms scalability: Reactive and proactive,” in *Advanced Information Networking and Applications (AINA), IEEE 27th Int. Conf. on*, 2013, pp. 1009–1016.

[5] T. A. Limoncelli, “OpenFlow: A radical new idea in networking,” *Queue*, vol. 10, no. 6, pp. 40–46, Jun. 2012.

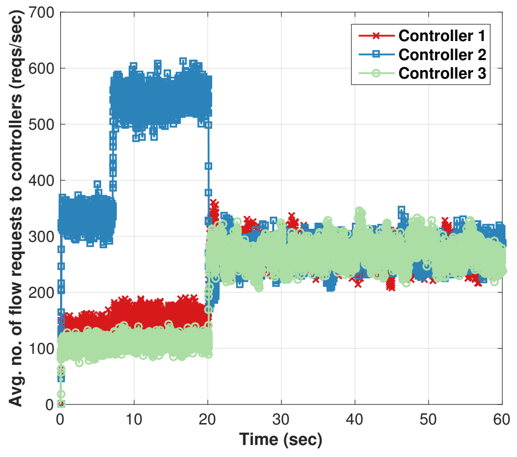


(a) w=0.15 and z=0.5

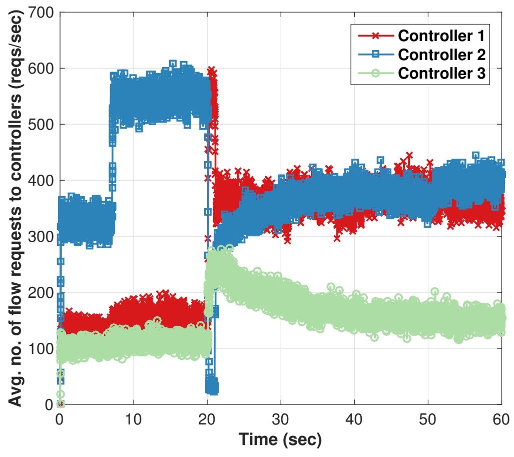


(b) w=0.3 and z=0.6

Fig. 7. Results for different w and z values.



(a) Random (RND) algorithm



(b) Nearest-Neighbor (NEN) algorithm

Fig. 8. Results for different reassignment algorithms.

[6] A. Dixit, F. Hao, S. Mukherjee, T. Lakshman, and R. Kompella, "Towards an elastic distributed SDN controller," *SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 7–12, Aug. 2013.

[7] M. Bari, A. Roy, S. Chowdhury, Q. Zhang, M. Zhani, R. Ahmed, and R. Boutaba, "Dynamic controller provisioning in software defined networks," in *Network and Service Management (CNSM), 2013 9th International Conference on*, Oct 2013, pp. 18–25.

[8] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood, "On controller performance in software-defined networks," in *Proceedings of the 2Nd USENIX Conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services*, ser. Hot-ICE'12. Berkeley, CA, USA: USENIX Association, 2012, pp. 10–10.

[9] S. Schmid and J. Suomela, "Exploiting locality in distributed SDN control," in *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN '13. New York, NY, USA: ACM, 2013, pp. 121–126.

[10] I. F. Akyildiz, A. Lee, P. Wang, M. Luo, and W. Chou, "A roadmap for traffic engineering in SDN-OpenFlow networks," *Computer Networks*, vol. 71, pp. 1 – 30, 2014.

[11] A. Voellmy and J. Wang, "Scalable software defined network controllers," *SIGCOMM Comput. Commun. Rev.*, vol. 42, no. 4, pp. 289–290, Aug. 2012.

[12] M. Yu, J. Rexford, M. J. Freedman, and J. Wang, "Scalable flow-based networking with DIFANE," in *Proceedings of the ACM SIGCOMM 2010 Conference*. New York, NY, USA: ACM, 2010, pp. 351–362.

[13] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "DevoFlow: Scaling flow management for high-performance networks," in *Proceedings of the ACM SIGCOMM 2011 Conference*, 2011, pp. 254–265.

[14] A. Tootoonchian and Y. Ganjali, "HyperFlow: A distributed control plane for OpenFlow," in *Proceedings of the 2010 Internet Network Management Conference on Research on Enterprise Networking*, ser. INM/WREN'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 3–3.

[15] S. Hassas Yeganeh and Y. Ganjali, "Kandoo: A framework for efficient and scalable offloading of control applications," in *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, ser. HotSDN '12. New York, NY, USA: ACM, 2012, pp. 19–24.

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

[17] SimPy, "SimPy discrete-event simulation framework," 2015. [Online]. Available: https://simpy.readthedocs.org/en/latest/, December 2014.