Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2020.DOI

Controller Load Balancing in SDN based on Traffic Load Prediction

ABHA KUMARI[1], DELCIN MARIA KOILRAJ[2], JOYDEEP CHANDRA[3] AND ASHOK SINGH SAIRAM[4], (Senior Member, IEEE)
[1]Indian Institute of Technology Patna, Patna, India - 800013 (e-mail: abha.pcs16@iitp.ac.in)
[2]Indian Institute of Technology Guwahati, Guwahati, India - 781039 (e-mail: delcin@iitg.ac.in)
[3]Indian Institute of Technology Patna, Patna, India - 800013 (e-mail: joydeep@iitp.ac.in)
[4]Indian Institute of Technology Guwahati, Guwahati, India - 781039 (e-mail: ashok@iitg.ac.in)

Corresponding author: Ashok Singh Sairam (e-mail: ashok@iitg.ac.in).

ABSTRACT In a distributed software defined networks (SDN) architecture, a switch is typically assigned to a controller. However, a static switch to controller mapping cannot adapt to real networks where the traffic condition is highly dynamic. In this paper, we propose a dynamic controller to switch association scheme with an aim to minimize flow setup time. We perform a detailed analysis of the controller load to understand its ideal load. We investigate different switch reassignment strategies and highlight the reason behind their performances. An efficient switch reassignment strategy must judiciously choose the switch for migration. We propose the likelihood of predicting the controller load in future iterations and accordingly chose the switches for migration. Experimental results confirm the conclusions of our analytical analysis.

INDEX TERMS software defined networking, load balancing, flow setup time, switch oscillation.

I. INTRODUCTION

Software defined networking (SDN) is not only deployed in enterprise but it is also widely prevalent to build *virtual infrastructures*, such as server and storage virtualization. It is one of the enabling technology for 5G cellular networks. In short, SDN has transformed networks in a way unseen in decades.

In the SDN architecture, a controller serves as a central command. All the forwarding decisions are flow-based. The controller on receipt of a *flow-request* from a switch, computes the route of the flow. The flow setup time depends on processing speed of the controller and the rate of flow-requests from the switches. To improve the flow setup delay as well as for reasons of availability, a key design goal is to deploy multiple SDN controllers. Thus the SDN architecture although logically centralized is physically a distributed architecture.

In a distributed controller architecture, a switch is associated with one of the controllers. The controller-switch association can be static or dynamic. In the *static switch assignment* approach ([9], [11]), switches direct *flow-requests* to controllers based on pre-installed packet handling rules. The problem is generally modelled as a capacitated controller problem [18]; that is load on a controller should not exceed its processing capacity while optimizing flow setup time. However, once a controller becomes overloaded, its response time will suffer. Consequently, the problem of reducing flow setup time is reduced to one of load balancing and synchronization in a multi-controller SDN environment.

A major drawback of a static stitch assignment strategy is that it cannot adapt to spatial and temporal variation in the network traffic. In the *dynamic switch assignment* approach ([8], [14]), the controller-switch mappings are altered so as to adapt to changing traffic conditions. To maintain balance, we shift load from an overloaded controller to a lightly loaded one. The granularity at which the loads can be reassigned is at the switch level. An alternative approach to dynamic assignment is to direct flows from the switch to a load balancer or a virtualized network platform [4]. The load balancer dynamically assigns flow request to controllers. Although the use of a dedicated load balancer allow fine-grained management of the flow-requests, they are a single point of failure and concern of potential bottleneck. In this paper, we adopt the dynamic controller-switch mapping strategy. It has been shown that, it is possible to reassign switches without disruption to ongoing flow-requests from applications [6].

The problem of switch migration has been shown to be NP-complete [13]. The existing solution techniques are, therefore, typically heuristic. In general, the dynamic switch migration technique involve three broad steps. The first step is identification of overloaded controllers based on a threshold. The threshold is either assumed to be the average controller load or it is determined empirically. The second step is to settle on the switches for migration (so called *victim* switches) from the overloaded controllers. The third step involve selection of *target* controllers, where the victim switches can be migrated. In this work, we provide theoretical bounds on the ideal load of a controller. These bounds facilitate a more systematic approach to determine a threshold on the controller load. We perform a detailed analysis of the different possible mechanisms to select the victim switch. We show that an ordered scheduling of the switches yield better load balancing but result in large switch reassignments. We establish that *switch oscillations*, where a switch alternately migrate between a pair of controllers, mainly contribute to the large number of reassignments. Switch oscillations are primarily due improper switch selection. We propose an extension to the ordered scheduling approach called look ahead load balancing (LALB) where we maximize the likelihood of predicting controller load. The contributions of the paper can thus be summarized as follows:

- The problem of minimizing flow setup time in dynamic SDN network traffic conditions is formulated as a load balancing problem.
- To facilitate determining the threshold of a controller, we define an upper and lower bound on the ideal load of a controller.
- A detailed analysis is performed of the different strategies possible to select victim switches for reassignment. We conclude that ordered scheduling provides superior load balancing but with high number of reassignments.
- We propose to lower the reassignments using a twin strategy of identifying and inhibiting switches that frequently oscillate as well as exploit machine learning models to predict the load of a switch.
- Extensive simulations are used to corroborate our theoretical results. Our proposed scheme is capable of reducing the assignments by 90% without adversely affecting load balancing.

The remainder of the paper is organized as follows. The related work is discussed in Section II. The problem definition and an analysis of the controller load are presented in Section III and Section IV respectively. The switch migration models are described in Section V. The experimental results are presented in Section VI. Finally the concluding remarks are given in Section VII.

II. RELATED WORK

In this section, we survey the state of the art of dynamic controller-switch assignment in SDN. Wang et al. [15] proposed a framework *Switch Migration-based Decision-Making* (SMDM) where they compute *load diversity*, a ratio of controller loads. Switch migration takes place in case the load diversity between two controllers exceeds a threshold. The threshold is assumed to be the aggregate controller load. The switches selected for migration are those with less load and higher efficiency.

Hu et al. [8] proposed *Efficiency-Aware Switch Migration* (EASM), where they measure the degree of load balancing using the normalized load variance of the controller load. In the event of the load difference matrix exceeding a threshold, the controller load is presumed to be imbalanced. The threshold is a function of the difference between the maximum and minimum controller load.

Filali et al. [7] use the ARIMA time series model to predict switch's load. The forecasting allows finding the time step at which a controller will become overloaded and accordingly schedule a switch migration in advance. The authors use a predetermined threshold to identify overloaded controllers.

The work by Zhou et al. [19] are among those few who consider the problem of *load oscillation* due to inappropriate switch migration. The issue of load oscillation is where underloaded controllers used to offload the load of overloaded controllers, themselves become quickly overloaded. The authors' reason that the problem is because the overall network status is not considered while selecting the target switches. Ul Haque et al. [14] propose to handle the variation in the controller load by employing a controller module, which is a set of controllers. Their approach periodically predicts the number of flows that will be generated by the switches and accordingly activates the number of required controllers.

Chen et al. [5] use a game-theoretic approach to solve the problem of controller load balancing. The underloaded controllers are modelled as players who compete for switches from overloaded controllers. The payoffs are determined when an underloaded controller is selected as the master controller of a victim switch.

Our survey reveals that although most of the controller load balancing strategies are threshold-based, there is no clear strategy to determine the threshold. Further, these approaches have not performed a detailed analysis of their victim switch selection policy.

III. SYSTEM MODEL

In SDN, the control plane is separated from the data forwarding layer, resulting in a highly scalable and programmable control framework. The SDN controller functions as the central command of the control plane. It communicates forwarding decisions to the switches and routers. In SDN, all the forwarding decisions are flow-based. The network devices have one or more flow tables populated with forwarding rules by the controller. The main job of the SDN controller is setting up the flow rules.

Definition 1 (Flow). *From a SDN perspective, a flow f is a sequence of packets exchanged between network devices.*

The flow setup process begins with the arrival of a packet at a forwarding device. The device looks up its flow table for a matching entry. In the event, the arriving packet does not match any entry of the table; it is a new flow. The network device use the packet's header fields to construct a

PACKET_IN message and forwards it to the controller for exception handling. We define this control message as *flow-request*.

Definition 2 (Flow-request)**.** *A flow-request is the control message sent by a switch to a controller to setup flow rules.*

The controller on receipt of a flow-request runs an instance of the routing protocol to determine the route. It subsequently sends *FLOW_MOD* command message to populate flow table of the affected network devices. It also sends a *PACKET_OUT* message to the network device that initiated the flow-request, signalling the device to begin the flow. The total time taken to setup a new flow is the flow setup time.

Definition 3 (Flow setup time)**.** *For a given flow-request, the time required to setup forwarding rules in all the involved switches as well as the time taken to forward the first packet along the data plane.*

Figure 1, shows a scenario with two controllers and ten switches. Each controller is associated with five switches. A new flow arrives at switch $s2$, and it is destined to $s6$. The flow triggers an exception, and a *PACKET_IN* control packet is forwarded to the controller $C1$. The controller processes the flow-requests and computes the path of the flow. Let us assume the computed path as $s2 \rightarrow s4 \rightarrow s10 \rightarrow s6$. The *FLOW_MOD PACKET_OUT* messages from the controller is sent to these affected switches while *PACKET_OUT* message from the controller is sent to source switch as shown.

A. PROBLEM DEFINITION

Initially, we formulate the problem as one of minimizing flow setup time. The problem fundamentally involves multiple components. We rework the problem as a multi-objective problem of balancing controller load, minimizing switch migrations and node to controller latency. Let the network comprise of a set of n switches, $\mathcal{S} = \{s_1 \dots s_n\}$, *and* k controller, $\mathcal{C} = \{c_1 \dots c_k\}$, where $k \leq n$. The controllers are assumed to be co-located with the switches [9]. Table 1 presents the symbols and variables used in the problem formulation.

TABLE 1: List of important symbols

Symbol	Description
$\mathcal{S}$	Set of switches, $\mid S \mid = n$
$\mathcal{C}$	Set of controllers, $\mid C \mid = k$
$\zeta_s(t)$	Flow-request rate of switch s at time t
ξ_c	Execution rate of controller c
$\delta_{s,c}^{prop}$	Propagation delay between nodes s and c,
δ_c^{proc}	Processing delay of c, $c \in \mathcal{C}$
$\delta_{s,c}^{f}$	Flow setup delay of switch s assigned to controller c
$\Phi_c(t)$	Instantaneous load of controller c at time t
$\Phi_c^{\mathrm{I}}(t)$	Ideal load of controller c at time t

1) Flow Setup Time

The flow setup time consists of three components. The time taken for the *PACKET_IN* message to travel from the network device to its associated controller. The processing time of the controller. The third component is the latency encountered by the control messages to travel from the controller to the corresponding network devices. The flow setup time of a flow f arriving at a switch s assigned to the controller c is computed as follows

$$\delta_{s,c}^{f} = \delta_{s,c}^{prop} + \delta_{c}^{proc} + \max_{s' \in \mathcal{S}'}\{\delta_{c,s}^{prop}, \delta_{c,s'}^{prop}\} \tag{1}$$

where $\mathcal{S}'$ denote the set of switches affected by the flow. The third term of the equation, $\delta_{c,s}^{prop}$ and $\delta_{c,s'}^{prop}$ denote the propagation delay of $FLOW\_MOD$ and $PACKET\_OUT$ messages respectively.

B. MULTI-OBJECTIVE OPTIMIZATION

The flow setup time (Equation 1) is a function of the processing speed of controllers and propagation delay between switches and controllers. An overloaded controller will take more time to process flow-requests. Thus, the goal of optimizing flow setup latency reduces to limiting flow-requests to a controller within its processing capacity. The plan is to distribute requests among controllers such that the instantaneous load of a controller is as *balanced* as possible. The *imbalance metric* of a controller load is the difference between the ideal and instantaneous load of the controller. The *ideal controller load* (Φ_c^{I}) is defined as the traffic load, such that the processing capacity of the controller is not exceeded. Our first objective is to minimize the imbalance metric.

$$f1 : min|\Phi_{c_j}^{\mathrm{I}} - \Phi_{c_j}(t)|, \ \forall c_j \in C \tag{2}$$

Let the variable x_{ij} is set to 1, if switch s_i is assigned to controller c_j, else it has the value 0. This ensures that a switch is assigned to only one controller at any time instant.

$$\sum_{i=1}^{n}\sum_{j=1}^{k} x_{ij}(t) \leq 1$$

Due to temporal variations in the network traffic, to keep the imbalance metric contained, one needs to alter the switch-controller mapping. However, switch migration involves overhead in terms of control messages exchanged as well as delay encountered by flows affected by the migration. Our second objective is, therefore, to minimize the switch reassignments.

$$f2 : min \sum_{j=1}^{k}\sum_{i=1}^{n} |x_{ij}(t) - x_{ij}(t-1)| \tag{3}$$

The process of switch migration involves choosing a *victim* switch s_i for migration from amongst the switches assigned to the overloaded controller. Further, let $C_u \subset C$, denote the set of underloaded controllers to which the victim switch can be re-assigned. We aim to select an underloaded controller c_j with minimum node to controller latency, that is

$$f3 : \min_{c_j \in C_u} \delta_{s_i,c_j}^{prop} \tag{4}$$

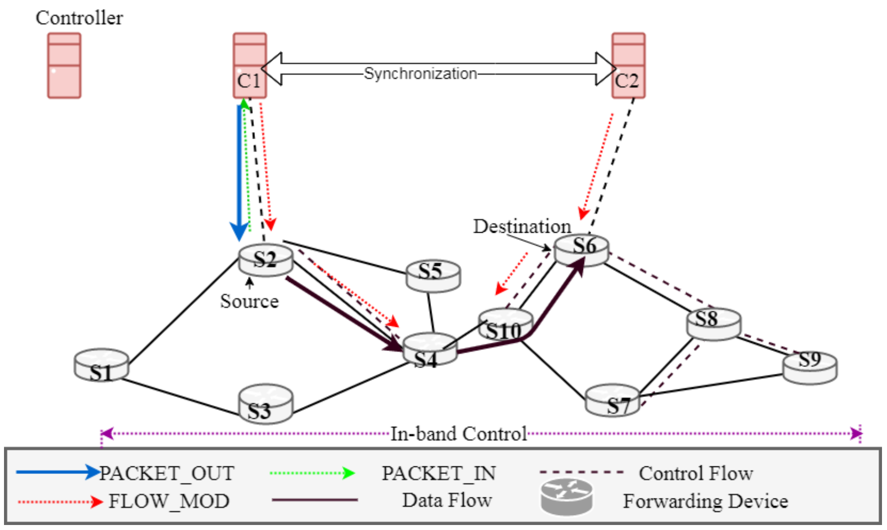

FIGURE 1: Flow setup in SDN

IV. CONTROLLER LOAD

In a threshold-based load balancing algorithm, a controller is considered congested when its load exceeds a pre-determined threshold. The threshold is generally assumed to be mean of the controller load [19], which is restrictive. We show that the mean is a lower bound. A more adaptable approach is to presume the threshold value to be a range based on double thresholds [17]. However, to the best of our knowledge, there is no standard procedure to estimate these threshold bounds. In this section, we analytically define an upper and lower bound on the controller load. Let $\zeta_s(t)$ denote the number of new flow arrivals at switch s in time t. We define the load of a controller as the sum total of flows that hit the controller. The instantaneous load of a controller c_j at time t can be computed as follows

$$\Phi_{c_j}(t) = \sum_{i=1}^{n} \zeta_{s_i}(t) x_{ij}(t) \tag{5}$$

The processing delay of a controller depends on its execution rate. Let ξ_c is the request execution rate of controller c. We assume that the total flow-requests in the network does not exceed the combined processing capacity of the controllers. Based on this assumption, if flow-requests are assigned to controllers commensurate to their processing capacity, a controller will not get overloaded. That is

$$\Phi^{\mathrm{I}}_{c_j} \geq \frac{\sum_{i=1}^{k} \Phi_{c_i}(t)}{\sum_{i=1}^{k} \xi_i} \xi_{c_j}$$

Such a distribution of the flow request would have been possible if we adopted a per flow-request controller assignment. Although per flow-request assignment allows fine-grained management of flows, it requires an additional device (load balancer) to manage the flows. The load balancer themselves can become a source of bottleneck and single point of failure. We, therefore, assume the granularity of load assignment is at the switch level. Let ζ_{max} and ζ_{min} represent the maximum and minimum flow-request rate from a switch. Further, for simplicity in representation, we assume homogeneous controllers.

Lemma 1. *A lower bound of the ideal controller load is defined as*

$$\Phi^{I}_{L}(t) = max\left\{\zeta_{max}, \left\lceil \frac{|S|}{k} \right\rceil \zeta_{min}\right\}$$

Proof. Let us examine each term of the equation. The first term is due to the generalized load balancing problem, which states that the optimal load on a machine is at least the size of the maximum job. We do not include the average controller load, since ζ_{max} will always be larger. The second term is due to the pigeon-hole principle when the number of switches is greater than the number of controllers. It defines the minimum number of switches that must be assigned to a controller. □

A tight upper bound on the ideal controller load will be achieved when the maximum load of any controller is minimized. The problem is equivalent to minimizing the *makespan*, the maximum load on a machine, which is shown to be NP-complete [10].

We consider an *ordered scheduling* of the switches to define a loose upper bound. The switches are assumed to be sorted in ascending order of their flow-request rate. They are assigned iteratively to the controller with least load. Given this scenario, we prove an upper bound on the controller load.

Lemma 2. *An upper bound on the ideal controller load is defined as*

$$\Phi^{I}_{U}(t) = \Phi^{I}_{L}(t) + \zeta^{*}_{j}$$

where ζ^{*}_{j} is the final switch load assigned to the controller with maximum load.

Proof. Let c_j be the controller with largest load and $\Phi^*_{c_j}$ be its load before the last switch is assigned. This means load of the controller is

$$\Phi_{c_j}(t) = \Phi^*_{c_j} + \zeta^*_j$$

Based on our assumption of *ordered scheduling*, $\Phi^*_{c_j}$ is the controller with the least load at that instant. The smallest controller load at a given time instant cannot exceed the ideal load, Φ^{I}_L. Thus the above equation can be written as

$$\Phi_{c_j}(t) \le \Phi^{\mathrm{I}}_L(t) + \zeta^*_j$$

□

The load ζ^*_j can be approximated as one-half of the lower bound [10]. Thus we can approximate the result of Lemma 2 as $\Phi^{\mathrm{I}}_U(t) \le 1.5\Phi^{\mathrm{I}}_L(t)$ for practical purpose. Combining Lemma 1 and 2, we provide a bound on the controller load.

Theorem 1. *The bound on the ideal controller load at time instant t is*

$$\Phi^I_L(t) \le \Phi^I(t) \le \Phi^I_U(t)$$

V. SWITCH MIGRATION MODELS

In this section, we propose two schemes to load balance the controllers by dynamically assigning switches to controllers,

A. ORDERED SCHEDULING

The proposition work in two stages. Initially, switches are assigned to controllers either randomly or using a clustering approach. We compute the instantaneous load of a controller as well as it's lower and upper bound. The *underloaded* and *overloaded* controllers are identified. We also identify the candidate switches to be reassigned (S_v). The procedure is given in Algorithm 1.

In the next phase, the greedy module (Algorithm 2) pick the switch to be reassigned from S_v, the so-called victim switch as well as its target controller. The set of victim switches is sorted in ascending order in terms of their flow-requests rate. These switches are reassigned, starting with the first switch, one at a time to a target controller. An underloaded controller with the minimum switch to controller latency is chosen as the target controller. The reassignment process continues until the load of the overloaded controller reduces within the permissible limit. During switch reassignment, we also ensure that the target controller does not get overloaded.

B. ANALYSIS OF GREEDY SCHEDULING

The victim switches chosen for reassignment can be selected either by ordering them or randomly. In this section, we perform a detailed analysis of these various strategies. We show that the load balancing achieved by selecting victim switches in ascending order of their load is at least as good or better than selecting switches in any other manner. To corroborate our claim, we extend our definition of the set of victim switches as S^{mode}_v, where *mode* defines the order in which the victim switches are selected. Let S^a_v, S^d_v and S^r_v be the set of victim switches when selected in ascending, descending and random order respectively.

Algorithm 1 Identify overloaded, underloaded controllers and victim switches

```
Require: Flows of switch at time $t$, $Flows_s(t)$ where $s \in S$
Ensure: Victim switch and its target controller
1: Initialize: ▷ Compute ideal and instantaneous controller load
       Compute $\Phi^{\mathrm{I}}_L(t)$ using Lemma 1
       Compute $\Phi^{\mathrm{I}}_U(t)$ using Lemma 2
       Compute $\Phi_{c_j}(t)$ using Equation 5
2: $C_T \leftarrow \{c_j : \Phi_{c_j}(t) < \Phi^{\mathrm{I}}_L(t), \forall c_j \in C\}$
3: $C_O \leftarrow \{c_j : \Phi_{c_j}(t) > \Phi^{\mathrm{I}}_U(t), \forall c_j \in C\}$
4: for all $c_j \in C_O$ do
5:     $S_v(c_j) \leftarrow \{s_i : x_{ij} == 1, \forall s_i \in S\}$
6: end for
7: LoadBalGreedy($S_v$, $C_T$, $C_O$)
```

Algorithm 2 Ordered switch scheduling (ascending)

```
1: function LoadBalGreedy($S_v$, $C_T$, $C_O$)
2:     for all $c_j \in C_O$ do
3:         while $\Phi_{c_j}(t) > \Phi^{\mathrm{I}}_U(t)$ do
4:             $S^{sorted}_v \leftarrow sort(S_v(c_j))$ ▷ Sort victim switches in ascending order
5:             $s_i \leftarrow S^{sorted}_v[1]$ ▷ Pop the first switch
6:             $\Phi_{c_j}(t) \leftarrow \Phi_{c_j}(t) - \zeta_{s_i}$
7:             $c_u \leftarrow min\{d(c', s_i), \forall c' \in C_T\}$ ▷ Minimum latency target controller
8:             if $\Phi_{c_u}(t) + \zeta_{s_i} \le \Phi^{\mathrm{I}}_U(t)$ then
9:                 $x_{ij} \leftarrow 0$; $x_{iu} \leftarrow 1$
10:                $\Phi_{c_u}(t) \leftarrow \Phi_{c_u}(t) + \zeta_{s_i}$
11:            else <GOTO 6 and get next $c_u$>
12:            end if
13:        end while
14:    end for
15: end function
```

Let c_o be an overloaded controller from which victim switches are selected and reassigned. Let s_i be the last switch selected for reassignment in the i^{th} iteration. This means after the switch is reassigned, the controller load will be less than or equal to the upper bound of the ideal controller load, that is,

$$\Phi_{c_o}(t) - s_i \le \Phi^I_U(t) \tag{6}$$

Theorem 2. *A tight upper bound of Equation 6 is achieved if $s_i \in S^a_v$.*

Proof. The equation can be rewritten as $\Phi_{c_o}(t) \le \Phi^I_U(t) + s_i$. The switch s_i can be selected in the i^{th} iteration in three possible ways, $s^a_i \in S^a_v$, $s^d_i \in S^d_v$ and $s^r_i \in S^r_v$. From definition, we have $s^a_i \le s^r_i \le s^d_i$. Further if $S^a_v \cap S^d_v \cap S^r_v = \phi$, then the inequality will definitely hold. Thus we get the most tight upper bound when $s_i = s^a_i$. □

The statement of Theorem 2 can also be understood intuitively. When we offload a controller by selecting switches sorted in ascending order, we unload the controller in the smallest possible steps. Although the proposed greedy approach gives better load balancing, the switch reassignments are inflated.

Theorem 3. *Ordered scheduling of switches, sorted in ascending order, result in maximum switch reassignments.*

Proof. Consider an instance of the greedy algorithm. Without loss of generality, let $c_1, \ldots, c_r$ be the overloaded controllers. Let us call the reassignment performed by the greedy algorithm as R. The initial space available for reassignments is

$$SR_0 = \sum_{j=1}^{r} (\Phi_{c_j}(t) - \Phi_U^{\mathrm{I}})$$

Let SR_i denote the space available for reassignment after the i^{th} switch has been transferred. Let $R^{'}$ be the reassignment performed in such a way that at the i^{th} iteration, it chooses a switch different from R. Since R always chooses the switch with least load, it implies, $SR_i > SR'_i$. The space available for R' is less than that of R. Hence the number of reassignments required for R' will be less than or equal to that of R. □

The greedy reassignments focus only on minimizing the instantaneous imbalance metric without considering the overall network load scenario. Due to the dynamic nature of Internet traffic, the greedy reassignments induce *controller load oscillations* [19]. It is a state where controllers oscillate between overload and underload in cycles. The controller load oscillations are linked to *switch oscillations*, switch alternate between controllers. These unwarranted oscillations form a significant component of the switch reassignments.

C. LOOK AHEAD LOAD BALANCING(LALB)

We propose two changes to address the issue of switch reassignments in the ordered scheduling approach. The first refinement is to reassign switches based on their predicted load. We compute the likelihood of a controller becoming overloaded or underloaded, given its instantaneous flow load. Notwithstanding the switch reassignments based on their predictive load, switches with a persistently low flow-request rate are selected more often as victim switch. This result in switch oscillations. The second modification, therefore, is to dampen the switch oscillations. We introduce a metric, *switch oscillation ratio* to measure the extent of a switch alternating between controllers. Switches with high oscillation ratio are prohibited from being reassigned.

1) Look Ahead

In the look ahead phase, the aim is to predict the number of flows that a switch will generate in the next iteration by relating it to past values. The flows generated by a switch is sampled to convert the data points into a univariant time series. For a given time period T and sampling interval $\Delta\tau$, the time series is represented as $TS\left[1 : \lceil T/\Delta\tau \rceil\right]$ With input as time-series data, we use support vector regression (SVR) to estimate the switch load in future iterations [12]. Using the probability distribution of the number of flows generated by a switch, we compute the likelihood of the controller load. A switch will be withdrawn from a controller only if it is expected to be overloaded in future iterations.

2) Damping Switch Oscillation

We define *switch oscillation ratio* as a ratio of the number of times a switch has been reassigned to the total number of iterations. Let, $P_m(s_i)$ denote the switch oscillation ratio of switch s_i at the current iteration m. The switch oscillation ratio at the $m+1$ iteration is defined as

$$P_{m+1}(s_i) = \frac{m}{m+1} P_m(s_i) + \frac{x^i_{m+1}}{m+1} \tag{7}$$

where $x^i_{m+1} = 1$, if s_i will be chosen as victim at the $(m+1)^{th}$ iteration or 0 otherwise.

In LALB, the switches are sorted in ascending order of their predictive load. Like ordered scheduling, the switch with the least load is selected as a candidate for reassignment. However, the switch is eligible for reassignment only if its oscillation ratio is less than a threshold α, the *damping* factor. The range of α is (0,1), and its value is determined empirically. It is important to note that this ratio will be higher for a switch which consistently has a low flow-requests rate. This additional check ensures that switches with higher flow rate are also chosen for relocation resulting in lesser reassignments as well as oscillations.

VI. PERFORMANCE EVALUATION

We evaluate the performance of the proposed schemes using python-based simulations. The network topology used is LambdaNet from the Internet Topology Zoo [3]. In our evaluation, we are primarily concerned with the controller load and do not emulate the data plane overhead or the actual transmission of packets through switches. The parameters used in our experiment are tabulated in Table 2.

We used three datasets to simulate the traffic generated by the switches - two datasets from CAIDA [1] and a university [2] dataset. These datasets give us a flavor of backbone as well as enterprise network traffic. The traffic is decomposed into flows. A flow is defined as a four tuple consisting of source and destination IP addresses and port numbers. We sample the data in a time interval so as to convert the N data points into a univariant time series [12].

A. PERFORMANCE METRICS

Our objective is to load balance the controllers, reduce switch reassignments as well as the cost involved in the reassignments. Accordingly, the performance metrics used are load balancing rate, switch reassignment rate and migration cost.

- Load divergence rate: The load divergence rate compute the variance [8] of the controller load from its ideal value.

Parameter	Value	Parameter	Value
# Controllers	4	# Switches	41
Topology	LambdaNet	D* factor (α)	0.7
Dataset: CAIDA1 [1]			
Data size	1065GB	# Flows	67M+ (1hr)
Flow-req. rate	18.7k/s	Flow size	30 sec
Dataset: CAIDA2 [1]			
Data size	1463GB	# Flows	63M+ (1hr)
Flow-req. rate	17.6k/s	Flow size	30 sec
Dataset: University [2]			
Data size	97.17GB	No. of Flows	0.43M+(56mins)
Flow-req. rate	110k/s	Flow size	30 sec

* Damping, +Million

TABLE 2: Experimental Settings

$$\frac{\sum_{j=1}^{k}(|\Phi_{c_j}(t)-\Phi^{I}_{c_j}(t)|)}{k}$$

- Switch reassignment rate: This metric reflect the permanency of the reassignments. It is measured as the number of switch reassignments which return to their *original* controller of the previous epoch. In case the reassignments are stable, none of the switches will return to their previous controller and ideally the value of the metric should be 0.

$$\frac{\sum_{s_i\in S_v}\sum_{c_j\in C}\neg(x_{ij}(t)-x_{ij}(t-1))}{|S_v|}$$

- Migration cost: The migration cost [15] gives a measure of the delay that will be encountered due to migration. It consists of two components: (i) the cost of exchanging messages to effect the switch migration and (ii) increase in propagation delay of the affected flows. It has been shown that the number of messages exchanged to move a switch from one controller to another is 6 times the round trip time (RTT) [8]. The cost of moving a switch s_v from controller c_o to c_t is thus given as

$$6*(2*\delta^{prop}_{c_o,c_t})+\zeta_{s_v}(t)(\delta^{prop}_{c_t,s_v}-\delta^{prop}_{c_o,s_v})$$

The second term will be set to 0, if $\delta^{prop}_{c_o,s_v}>\delta^{prop}_{c_t,s_v}$.

B. EXPERIMENTAL RESULTS

We compare LALB with two other possible scheduling of the victim switches - *ordered scheduling (ascending)* and *random*. We label the ordered scheduling approach, described in Section V-A, as Greedy. In the random approach, the victim switches are selected randomly.

The experiments are run for 3000 iterations, each iteration of 30 seconds duration. We pool the output in groups of 10 and compute their average. The same process is repeated 10 times and their average is computed.

1) Instantaneous Controller Load

The initial assignment of the switches to controllers is achieved using K-means [16]. A scatter plot of the instantaneous load of the controllers is shown in Figure 2, keeping the switch to controller assignment static. For clarity, we show the traffic load of only three controllers. The load on the controllers vary over a wide range. It can be seen that the load of a controller fluctuate between overloaded and underloaded. On average, the average load of the controllers vary from 15% to 80%. This validate our stance that the network traffic is highly dynamic and the need of load balancing to minimize response time.

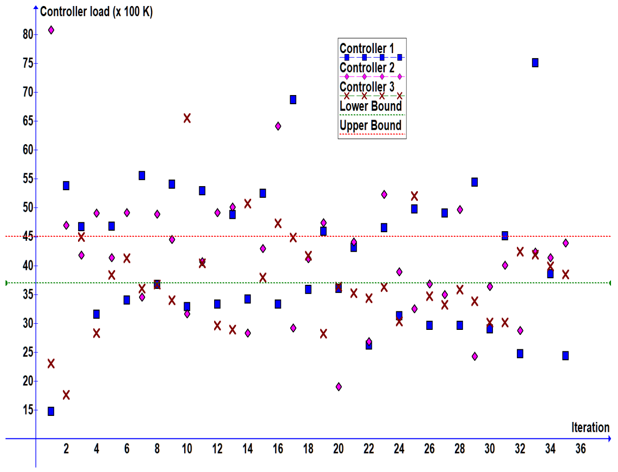

FIGURE 2: Load on Controllers without load balancing

2) Impact of Threshold value

In this experiment, we examine the effect of threshold value on the load divergence rate and the number of reassignments. Initially, the threshold value is set to its lower bound. Further, in each run of the experiment, the threshold value is incremented by one step. The step size is taken as the difference between the lower and upper threshold bound divided by the number of steps. For our experiment, we take the number of steps as 7.

The average load divergence rate of the controllers and the switch reassignment rate for each threshold value is tabulated in Table 3. As we increase the threshold value, the load divergence rate degrades while reassignment rate improves. The load divergence is best when the threshold value is set to lower bound but the number of reassignments is highest. The choice of the threshold value will be a trade-off between these two performance metrics. In the remaining experiments, we consider a step size of two as the threshold value.

Threshold step	Load divergence rate ($\times10^5$)	Reassignment rate ($\times10^{-1}$)
0	29.56	2.99
1	35.4	2.81
2	35.43	2.41
3	35.49	2.39
4	41.38	1.41
5	42.82	1.30
6	45.11	0.98

TABLE 3: Load divergence and switch reassignment rate for different threshold value

3) Switch Reassignment Rate

A comparison of the number of reassignments of the three approaches is shown in Figure 3. In all the iterations, the number of reassignment is least for LALB and highest for greedy. The reassignments of greedy on an average is 5 times more than that of LALB. The average reassignments of LALB, greedy and random are 1.787, 8.6 and 2.705 respectively. The experimental results corroborate our theoretical analysis (Theorem 3), where we have shown that the greedy approach will result in maximum reassignments. The reason behind LALB's superior performance is its reduction in switch oscillations. The random approach chooses victim switches with larger load. Thus, it is also able to achieve load balancing with relatively fewer reassignments.

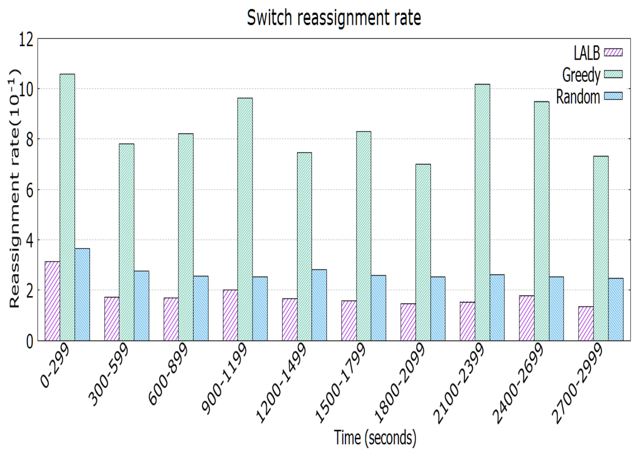

FIGURE 3: Comparison of reassignments

4) Switch Oscillations

To understand the switch reassignments, we also measure the switch oscillations, as shown in Figure 4. The number of switch oscillations is highest for the greedy approach which translates to high number of switch reassignments, as observed in Figure 3. LALB although a variant of the greedy approach, employ measures to dampen switch oscillations. As a result its oscillations is considerably less than that of greedy, about 90% less. Interestingly, the random approach also has a low number of switch oscillations. The random approach uses uniform distribution to select the victim switch. Therefore, the probability that a switch is selected for reassignment is $\frac{1}{p}$, assuming there are p candidate switches. For switch oscillation to occur, the same switch must be selected again for reassignment. Assuming q is the number of candidate switches in the second iteration, the probability of oscillation will be $\frac{1}{pq}$. This probability of oscillation will be low for $p,q > 1$.

5) Load Divergence

We measure the load divergence rate of the controllers, as depicted in Figure 5. The divergence rate of greedy is the least. This result is in agreement with our theoretical

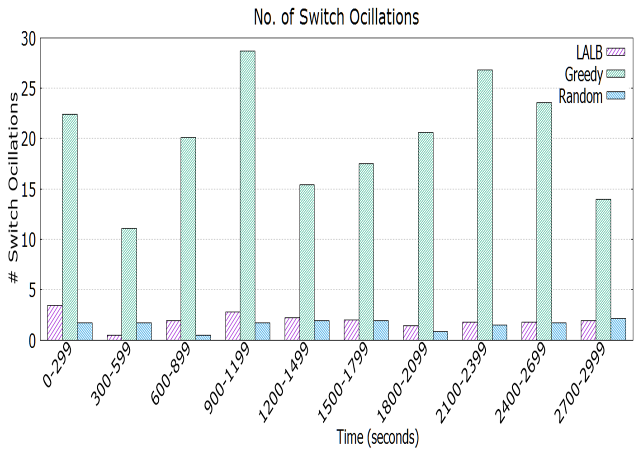

FIGURE 4: Comparison of switch oscillations

analysis. We have shown, using Theorem 2, that the best load balancing of controllers is achieved when victim switches are scheduled in ascending order of their load. The performance of LALB in terms of load balancing is comparable to that of greedy. LALB is able to achieve a good load balancing performances, in spite of its low number of reassignments, since it can predict the controller load and accordingly schedule the switches. The average load divergence rate of LALB, greedy and random approach are 3.63, 3.28 and 5.29 respectively. The random approach arbitrarily select switches which can result in reassignment of traffic load larger than necessary. Hence, it has a higher load divergence.

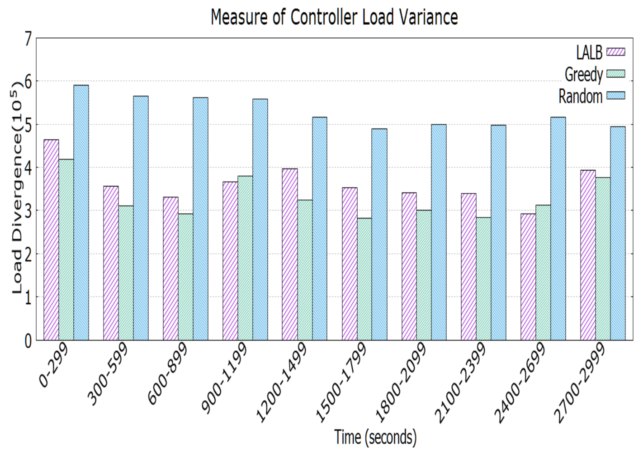

FIGURE 5: Comparison of load divergence rate

6) Migration Cost

The migration cost computed for the three approaches is shown in Figure 6. The migration cost depends on two factors –: choice of target controller and load of the victim switch. The process of choosing the target controller is same for all the three approaches. Thus the variation in migration cost that we observe is due to their different approaches of choosing the victim switch. A general trend that is observed is a higher migration cost in the initial iterations. This indicates that initially there is a large readjustment of traffic to load balance. Thereafter, the adaptation is to counter the traffic volatility which is relatively less. LALB has the least migration cost while random approach has about 40% higher cost on average. The performance of greedy is comparable to that of LALB.

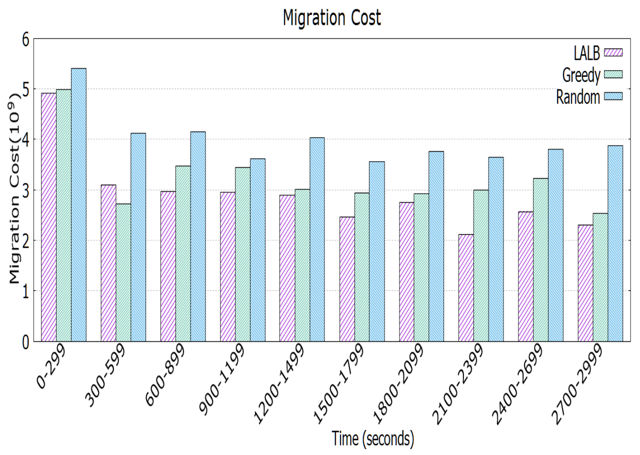

FIGURE 6: Comparison of migration cost

VII. CONCLUSION

In this work, we formulated the problem of minimizing flow setup time as a load balancing problem. We computed theoretical bounds of the controller load required to achieve load balancing. We established that a greedy approach of scheduling switches to controllers in increasing order of their traffic load, result in superior performance but with high number of reassignments. Consequently, we proposed look ahead load balancing (LALB), an extension of the greedy approach. The proposed approach effected the reassignments based on the predicted load of the switches as well as incorporate measures to reduce switch oscillations.

Extensive simulations show that the load balancing achieved by LALB is comparable to that of the greedy approach but the switch reassignments are lower by more than 90%. The migration cost of LALB is also lower. We show that the experimental results are in conformance with our theoretical analysis. We plan to employ an online prediction of the switch traffic as a future extension of this work.

REFERENCES

[1] The CAIDA UCSD Anonymized Internet Traces 2016, [Last accessed on 05/12/2019]. Available from: http://www.caida.org/data/passive/passive_2016_dataset.xml.

[2] Data Center Measurment University Data Set, [Last accessed on 05/12/2019]. Available from: http://pages.cs.wisc.edu/~tbenson/IMC10_Data.html.

[3] The internet topology zoo. http://http://www.topology-zoo.org/index.html, 2019 (accessed January, 2020).

[4] S. Bera, S. Misra, and N. Saha. Traffic-aware dynamic controller assignment in sdn. IEEE Transactions on Communications, pages 1–1, 2020.

[5] H. Chen, G. Cheng, and Z. Wang. A game-theoretic approach to elastic control in software-defined networking. China Communications, 13(5):103–109, 2016.

[6] Advait Dixit, Fang Hao, Sarit Mukherjee, TV Lakshman, and Ramana Rao Kompella. ElastiCon; an Elastic Distributed SDN Controller. In ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS), pages 17–27, 2014.

[7] A. Filali, S. Cherkaoui, and A. Kobbane. Prediction-based switch migration scheduling for sdn load balancing. In ICC 2019 - 2019 IEEE International Conference on Communications (ICC), pages 1–6, 2019.

[8] Tao Hu, Julong Lan, Jianhui Zhang, and Wei Zhao. EASM: Efficiency-Aware Switch Migration for Balancing Controller Loads in Software-Defined Networking. Peer-to-Peer Networking and Applications, 12(2):452–464, 2019.

[9] Bala Prakasa Rao Killi and Seela Veerabhadreswara Rao. Capacitated next controller placement in software defined networks. IEEE Transactions on Network and Service Management, 14(3):514–527, 2017.

[10] Jon Kleinberg and Eva Tardos. Algorithm Design. Addison-Wesley Longman Publishing Co., Inc., USA, 2005.

[11] Adlen Ksentini, Miloud Bagaa, Tarik Taleb, and Ilangko Balasingham. On using Bargaining Game for Optimal Placement of SDN Controllers. In IEEE International Conference on Communications (ICC), pages 1–6, 2016.

[12] A. Kumari, J. Chandra, and A. S. Sairam. Predictive flow modeling in software defined network. In TENCON 2019 - 2019 IEEE Region 10 Conference (TENCON), pages 1494–1498, 2019.

[13] W. Liang, X. Gao, F. Wu, G. Clien, and W. Wei. Balancing traffic load for devolved controllers in data center networks. In 2014 IEEE Global Communications Conference, pages 2258–2263, 2014.

[14] Md Tanvir Ishtaique ul Huque, Weisheng Si, Guillaume Jourjon, and Vincent Gramoli. Large-Scale Dynamic Controller Placement. IEEE Transactions on Network and Service Management, 14(1):63–76, 2017.

[15] Chuan'an Wang, Bo Hu, Shanzhi Chen, Desheng Li, and Bin Liu. A Switch Migration-Based Decision-Making Scheme for Balancing Load in SDN. IEEE Access, 5:4537–4544, 2017.

[16] Guodong Wang, Yanxiao Zhao, Jun Huang, Qiang Duan, and Jun Li. A k-means-based Network Partition Algorithm for Controller Placement in Software Defined Network. In IEEE International Conference on Communications (ICC), pages 1–6, 2016.

[17] H. Xiao, Z. Hu, and K. Li. Multi-objective vm consolidation based on thresholds and ant colony system in cloud computing. IEEE Access, 7:53441–53453, 2019.

[18] Guang Yao, Jun Bi, Yuliang Li, and Luyi Guo. On the Capacitated Controller Placement Problem in Software Defined Networks. IEEE Communications Letters, 18(8):1339–1342, 2014.

[19] Yaning Zhou, Ying Wang, Jinke Yu, Junhua Ba, and Shilei Zhang. Load Balancing for Multiple Controllers in SDN based on Switches Group. In 19th Asia-Pacific on Network Operations and Management Symposium (APNOMS), pages 227–230. IEEE, 2017.