Improving End-Users Utility in Software-Defined Wide Area Network Systems

Kshira Sagar Sahoo, *Student Member, IEEE*, Pritish Mishra, Mayank Tiwary, Somula Ramasubbareddy, Balamurugan Balusamy, and Amir H. Gandomi, *Senior Member, IEEE*

***Abstract*—Software Defined Networks (SDNs) has brought a new form of network architecture that simplifies network management through innovations and programmability. But, the distributed control plane of SD-Wide Area Network is challenged by load imbalance problem due to the dynamic change of the traffic pattern. The *packet_in* messages are one of the major contributors of the control's load. When such packet rate exceeds a certain threshold limit, the response time for control request increases non-linearly. In order to achieve better end-user experience, most of the previous works considered the optimal switch to controller association with an objective to minimize the response time on LAN environment but ignores the consequence of large scale network. In this regard, the proposed work realizes the necessity of layer-2 and layer-3 controller in LAN and WAN environment separately. A load prediction based alertness approach has been introduced to reduce the burden of the controllers. This approach may create an additional delay for the initial packets of the flow entry that lead to more prediction error. However, the proposed method reduces the error by selecting an optimal timeout value of the flow. Further, minimization of the response time between router to the controller has been taken care of. An extensive simulation shows the efficacy of the proposed scheme.**

***Index Terms*—Software defined networks, load balancing, OpenFLow device (OFD), load prediction, flow timeout.**

I. Introduction

SOFTWARE Defined Network paradigm has been flourishing in industries in recent years and has become the most useful network architecture for various working environments [1], [4], [5]. SDN can control and manage the underlying network via physically distributed, but logically centralized set of controllers to meet the Service Level Agreements (SLA) required by cloud service providers [3], [6]. With the increasing use of SDN, there is an essentiality to manage the load among controllers [2], [7], [8]. In this work, *packet_in* requests to the controllers have considered as load because the processing of *packet_in* is a computationally intensive task which increases the overall CPU utilization of the controllers [9]. To reduce the burden of the controller, some authors have suggested device migration technique, which transfers the excess loads from the current controller domain to another controller domain. If efficient algorithms are not implemented, the migration process may hamper the application and service performance [10]. To address the above issue, this work proposed a mechanism that depends on (i) used Network topology and (ii) controller placements in both LAN and WAN environment. The controller which is placed in a local area network is termed as the layer-2 (L2) controller, whereas the controller placed in Wide Area Location is termed as layer-3 (L3) controller. For better readability, in the rest of the paper, the term L3 and L2 controller have been used for the same.

In last few years, a sizable works focus on SDN load balancing especially for LAN environment [2], [6], [9]. Few authors have adopted device migration strategy for load balancing in distributed controller environment [28], [29], [31]. But to manage WAN traffic, the controller placement in layer-3 should not be ignored as the WAN delay acts as a bottleneck for the device migration. Whenever the controller is placed in a large network like WAN, the *packet_in*s generated from layer-2 OpenFlow Devices (OFD) suffer network delay. In another way, if the controllers are placed at every point of the WAN topology, it becomes difficult to meet the challenges of a centralized controller. Therefore, in this approach, we propose a controller placement strategy where the controllers are placed at each LAN point and to manage these L2 controllers, a few L3 controllers are located in the topology graph. In this regard, the L2 controllers are responsible for managing the *packet_in*s from the L2 devices. And the L3 controllers are responsible for handling the *packet_in*s from L3 devices.

An absolute timeout value is associated with each flow present in the flow table. If there are no matching for certain duration, the flow is removed from the flow table. The proposed work focuses on evaluating the flow entry timeout value such that if a flow miss occurs, the L2 controller can perform analytic on the generated *packet_in* and inform to the L3 controller (in case the traffic flow causes the router to generate a *packet_in* for the L3 controller). The timeout evaluation is versatile in nature and can even handle traffic flows, which show high dynamism against the number of packets. Further, the timeout evaluation scheme helps to minimize the error in

Manuscript received May 5, 2019; revised September 10, 2019; accepted November 4, 2019. Date of publication November 18, 2019; date of current version June 10, 2020. The associate editor coordinating the review of this article and approving it for publication was Q. Ling. *(Corresponding author: Kshira Sagar Sahoo.)*

K. S. Sahoo and S. Ramasubbareddy are with the Department of Information Technology, VNR VJIET, Hyderabad 500090, India (e-mail: kshirasagar12@gmail.com; svramasubbareddy1219@gmail.com).

P. Mishra and M. Tiwary are with Core Cloud Platform, SAP Labs Bangalore, Bengaluru 560066, India (e-mail: pritish.mishra@sap.com; mayank.tiwary@sap.com).

B. Balusamy is with the School of Computer Science and Engineering, Galgotias University, Greater Noida 203201, India (e-mail: kadavulai@gmail.com).

A. H. Gandomi is with the Faculty of Engineering and Information Technology, University of Technology Sydney, Ultimo, NSW 2007, Australia (e-mail: gandomi@uts.edu.au).

Digital Object Identifier 10.1109/TNSM.2019.2953621

load prediction, which has discussed in Section IV. The L2 controller alerts the L3 controller either through asynchronous or synchronous mode which we have described more precisely later. Lastly, it has observed that the load prediction helps the L3 controller to take effective migration decisions before the controller is actually loaded. Once the device migration is initiated, the new controller to router association is made by solving a response time minimization problem, which depends upon the controllers WAN link bandwidth and CPU utilization.

A. Motivation

Authors in [6] monitored the real-time traffic of China Education Network and observed that the maximum number of flows which traverses across the network is 3 million per second. It implies that if the flow table of the devices is empty, then it can generate 3 million *packet_in*s to the controller layer. Further, in [9], authors observed, when the *packet_in* per second exceeds a certain threshold, the response time of the control event rises exponentially. In order to virtualize the LAN and WAN resources, the cloud service providers will deploy the SDN application on the ISPs' OpenFlow controllers. With the increase in number of cloud service providers, there is an increasing difficulty of application state maintenance for the deployed SDN applications. Each of the SDN application needs to perform computations based on the requirements of the SLAs. Further, each cloud service provider has strict network SLAs against ISP. The problem of load balancing (which leads to maximization of end-users' utility) in SDN also depends on the selection of an appropriate device for migration so that the load on the controller can be reduced. But the optimal selection of the OFD based on the existing functionality of the OpenFlow 1.3 seems to be a difficult task [11]. To solve this problem, a load prediction strategy has been proposed, which can help in the selection of the devices to migrate with consideration of end-users utility, preferences, and priorities.

B. Contributions

To solve the above issues in real-time traffic, in this work a load prediction has been proposed. The proposed control architecture realizes the requirements of both L3 and L2 controllers separately. With the prior knowledge of traffic and with computational details about the processing of the respective *packet_in*s, the L2 controller can predict the load for WAN controllers. L3 controllers, in turn, utilizes this information and initiates the device migrations before the controller gets loaded.

The major contributions of this work can be *summarized* as follows:

- Propose a flow entry timeout evaluation method, which can handle the high dynamism of traffic flows in terms of number of packets. The proposed timeout strategy effectively minimizes the error in load prediction. This also includes a single-objective optimization problem for solving the problem of error while predicting the load.
- Selection of routers and target controllers is carried out solely based on the traffic priorities and response time, which aims at maximization of end-users' utility.
- Further, the evaluation of response time depends upon the current CPU utilization of controller, the computational complexity of the deployed applications, and WAN link bandwidth.

The organization of the paper is described as follows. Section II describes the literature survey relevant to above issues. Section III contains the proposed system model for Wide Area Software Defined Networks, placement of the controllers and load prediction. Further, the same section contains the details of load prediction followed by the proposed prediction algorithms. Section IV contains an overview of the proposed scenario. Section V gives the experimental setup details containing performance matrices and results discussion. Finally Section VI summarizes the proposed work with suggestions for future directions.

II. Related Work

With the separated control plane and data plane, the Software Defined Network paradigm enables ample freedom to manage and program the network. But, it is a challenging task to manage the properties of controllers like responsiveness, reliability and scalability [10], [12]–[14]. For efficient load management, two properties need to be considered: (i) load balancing in control plane and (ii) controller placement (CP) in a WAN environment. For better network management and resiliency, multiple controllers architecture came into existence, which was physically distributed but logically centralized in nature [5]. In [15], the author uses the spectral clustering algorithm which divides large network into smaller domains of networks. Though there are improvements in terms of throughput and latency, the paper ignores the minimization of WAN propagation delay when layer-2 traffic has to traverse through layer-3. He *et al.* model the CP problem as an offline optimization problem for minimizing the total cost produced by the flow setup performance and the controller adaptation. Further, this offline problem was solved in an online fashion with the help of simulated annealing technique [22]. DALB [2] has given a distributed load distribution decision making architecture instead of central decision maker for LAN environment where load collection, adjustment of threshold in load collection, decision making in the migration of switches were the functions of the controller. Works in Elasticon [9] is well suited for elastic control architecture for LAN environment, but there is no consideration of state distribution overhead which will be produced when controllers are placed in the network. For load balancing in control plane, both Onix [5] and Hyperflow [4] architectures consider a static association between switches and controllers. Onix used the standard distributed system to design the architecture with fixed binding of controllers and devices. On the other hand, in Hyperflow the local controller serves local request without contacting any remote controller. This helped in minimizing the flow setup time, but frequent changes in flow variations are the constraints to these architectures. Adaptive resource management was implemented in [19], which supports both static and dynamic resource and control management. For handling high traffic dynamism, new control plane model was introduced

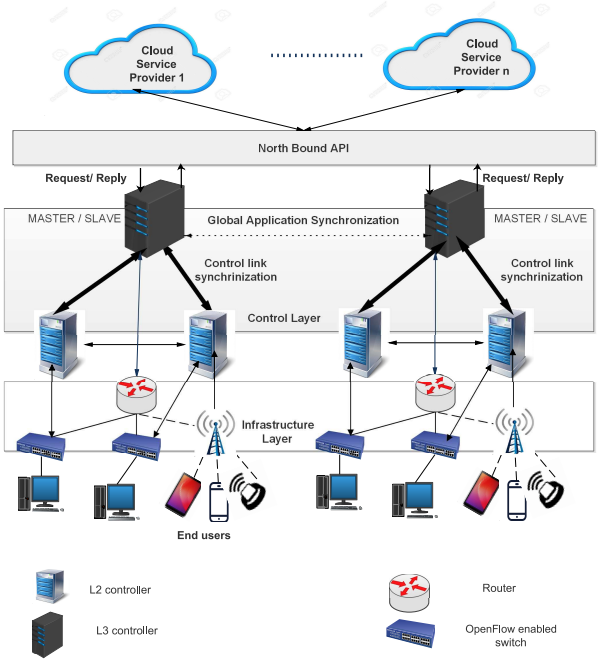


Fig. 1. Proposed Control Plane Architecture.

in DCPP [20]. In this article, the authors proposed an architecture where in response to changing traffic conditions, the number and position of controllers are dynamically changes to reduce flow setup time and communication overhead. In [9], the proposed scheme is an elastic distributed controller architecture in which the controller pool dynamically grows and shrinks based on a maximum and minimum threshold set for CPU utilization level. For better management of the load, the switch which generates maximum load on the controller gets migrated to the other SDN controller domain. But the response time of controller increases non-linearly after a certain threshold value. All these works focus on mapping of devices to the controller when the load parameters surpass a threshold. Also, all these works focus more on LAN scenario. Although several methods were proposed to solve CPP [16], [17], authors in [18], argued that no single rule could be applied to find the optimal number of controllers and their respective positions in WAN. To the best of our knowledge, no work has been introduced for WAN device to WAN controller mapping. In this respect, the WAN propagation delay for layer-2 traffic remained un-addressed. Our proposed work uses the alertness scheme for effective migration decision and improving end-users QoS in Software-Defined Wide Area Network (SDWAN).

III. System Model

In this section, an adaptive load balancing method has props for wide area software defined controllers which takes the migration decision before the controller actually gets loaded. Let us assume a WAN topology is represented by $G(V, E, W)$ as shown in Fig. 1, $V = (V_1, V_2, V_3, \ldots, V_i)$ represents the set of layer-3 OFD, $E = (E_1, E_2, E_3, \ldots, E_j)$ denotes the edges connecting these nodes in full duplex mode and $W = (W_1, W_2, W_3, \ldots, W_j)$ represents the bandwidth for respective edges. Let, $C = (C_1, C_2, C_3, \ldots, C_x)$ be the set of L3 controllers, where $C \subset V$, i.e., controllers have been deployed at some routing points in WAN environment. Each V_i in graph $G(V, E, W)$ represents a LAN environment, which can be denoted by the graph $G' = (V', E')$ where $V' = (V'_1, V'_2, V'_3, \ldots, V'_m)$ denotes layer-2 OpenFlow Devices and $E' = (E'_1, E'_2, E'_3, \ldots, E'_k)$ indicates the edges connecting these OFD in full duplex mode. Further, a set of controllers $C' = (C'_1, C'_2, C'_3, \ldots, C'_i)$ are deployed at each vertices V_i of $G(V, E, W)$. Set of controllers C is responsible for managing the L3 traffic whereas a set of controllers C' directs L2 traffic. Both C and C' are synchronized with each other so as to maintain network integrity and SDN application synchronization. A typical LAN network model is a collection of computing device such as workstations and servers connected to the switch, similarly a series of LANs linked by bridge or router forms a WAN model. If the controllers are placed at selected WAN points, the layer-2 (LAN) *packet_in*s will suffer WAN delay, on the other hand if the controllers are placed at every WAN point then it becomes difficult to maintain the application state synchronization of the controllers present in the network. With this motivation, the proposed model has been built which is illustrated in Fig. 1. In Fig. 1 the routers are attached with the L3 controller and not with L2 controller. Sometimes the L2 controller needs to access the flow table of the router. The L2 controller cannot directly access the routers flow table for which the L2 controller makes a request to L3 controller, in turn after reads the flow table it replies to L2 controller. The following assumptions have been held for this work.

Assumption 1: The Cloud Service Providers deploy the SDN applications over both L3 controllers (ISP controllers) and L2 controllers (user site controllers).

Assumption 2: This deployment architecture bears analogy with the current Internet Service Provider (ISP) WAN architecture, where ISP will have its own set of network controllers and these controllers will synchronize in a timely manner with the local user site controllers.

Assumption 3: The L3 controllers synchronize with the L2 controllers for maintaining constant SDN application state network-wide.

Assumption 4: The characteristics of the L3 controller is to manage WAN traffic whereas LAN traffic requirements are managed by L2 controllers.

Assumption 5: L2 controllers are deployed at every LAN point whereas L3 controllers are deployed at selected places in the network topology.

Assumption 6: At a particular time instance ΔT, the entire control plane cannot be overloaded.

Assumption 7: The router is connected with both L2 as well as L3 controller.

Assumption 8: L3 controller acts as both *MASTER* and *SLAVE* role for the router and it can access and modify it's flow table.

Assumption 9: L3 controller informs L2 controller about the flow entry of the routers based on an event-based propagation system. It informs only when there is a change in flow entries in the flow table of the router.

The management utility generally includes the state synchronizations of the SDN applications. If there are more than one WAN controllers, the WAN device has one WAN controller as *MASTER* and others act as in *EQUAL* mode.

A. OFDevice (OFD) Migration

Whenever a controller gets overloaded, the connected OFD suffer from the delay in response for processing of *packet_in*s. To avoid this, generally the OFDs are migrated to another connected controller during overload conditions. For the safe migration, the standard migration properties such as liveness, seriallizability, and safety ought to follow [9]. To meet the above criteria, and to follow the same migration strategy, OpenFlow 1.3 and higher versions provide *barrier_request* and *barrier_reply* features. Once an OFD receives a *barrier_request*, it starts clearing the buffer of already received control requests, and once it is removed, it replies back with *barrier_reply*. It means the response time for *barrier_reply* is directly proportional to application layer logic complexity. This delay causes service disruption and decreases the end-users QoS. The Algorithm 3 aims at selecting such OFDs for migrations such that least end-users traffic will be affected.

The Assumption-6, implicates that in a distributed control plane framework (with appropriate state synchronization mechanism), not all controllers are busy at any given instance of time. This idea arises from the fact that, if all controllers are busy at any given point of time, it implicates that the network manager would deploy a new controller in the network. The busyness of a controller can be the measure in terms of closeness of the current average response time to the target response time set by the network manager. In other words, if other controllers can share the load, it does not make sense to install a new controller in the network for that time period. So we need a mechanism to transfer a part of the load from a heavily loaded controller to a lightly loaded controller so that all the controllers work within their threshold load.

In SDWAN, the cloud service providers deploy their own SDN application over WAN controllers, where a single received *packet_in* has to be processed by one or a set of SDN applications deployed by multiple cloud service providers. This can even cause more computational complexity. Therefore, to model the computational complexity a calibrated number between 0 to 10 is assigned to every *packet_in* based on the variation of CPU utilization, memory utilization and the number of SDWAN applications, which process the *packet_in*. The calibrated number is termed as τ_i where *i* is the *packet_in* identifier. The calibrated number assignment is accomplished by executing the *packet_in* in ideal conditions. The current CPU utilization level approaches to 0% specify the ideal condition for the controller. Each device always has a specific set of *packet_in* to be sent to the controller. This set is represented by $G = \{\tau_1, \tau_2, \tau_3, \ldots, \tau_n\}$, where for every OFD *i*, there exists τ_i such that $\tau_{min} \leq \tau_i \leq \tau_{max}$.

B. Response Time Minimization

In this work, CPU utilization for controllers *C*, is represented by $\mathcal{P}_C(\cdot)$ and it depends on the value of τ_{max} for $r \in G$. $\mathcal{P}_C$ is mathematically represented as follows:

$$\mathcal{P}_C(\cdot) = \mu - \sum_{i=1}^{\mathcal{A}(t)} \mathcal{F}(\tau_i)\phi_i \tag{1}$$

where, $\mathcal{F}(\tau_i)$ is defined as:

$$\mathcal{F}(\tau_i) = \frac{\tau_i}{\sum_{x=1}^{S} \tau_x} \tag{2}$$

Equation (2) represents the percentage of load caused by a specific τ_i from a router with respect to all τ in the set *G* for a specific OFDs. In Equation (1), $\mathcal{A}(t)$ represents the total number of active devices at time slot *t* connected to the WAN controllers *C*. The μ denotes the average processing rate of the controller *C* and τ_{max} represents the maximum value of *packet_in* which can be sent by the router, ϕ_i is the job arrival rate. The average processing rate μ depends on controller's machine configuration and tasks which are currently being executed. Hence, the value of μ is always taken from the last time slot, i.e., $t - 1$. To obtain the value of ϕ_i in real time scenario for a time slot *t*, for a specific number of days the packets sending by a router have been averaged and then use these values for estimating the same.

For load balancing, the aim of the router *r* is to select such a controller; so that the response time would be minimal even after sending τ_{max} to the controller. To calculate the response time, the router needs to have $\sum_{i=1}^{\mathcal{A}(t)} \mathcal{F}(\tau_r)\phi_i$ value.

Definition 1: The load status of the controller is defined by $L_F(\cdot)$.

$$L_F(\cdot) = \sum_{i=1}^{\mathcal{A}(t)} \mathcal{F}(\tau_r)\phi_i \tag{3}$$

To calculate the response time, each router needs to have the current load status of the other controllers. Each controller announces its $L_F(\cdot)$ at the start of the time slot *t*. Using this load factor, each router evaluates its response time for a given controller set and selects the optimal controller. The problem of load distribution can be described as an optimization problem which can minimize the response time. This optimization problem can be formulated as follows:

$$Minimize\text{: } Response(\mathcal{F}(\tau_{max})) = \frac{\mathcal{F}(\tau_{max})}{\mathcal{P}_C(\cdot)} \tag{4}$$

$$\text{Subject to constraints: } \mathcal{P}_C(S, \mu) > 0 \tag{5}$$

$$\sum \mathcal{F}(\tau_i) = 1 \tag{6}$$

$$\sum \mathcal{F}(\tau_r)\phi_i > \mu \tag{7}$$

The constraints represent in Equation (5) and Equation (6) denote positivity and conservation, respectively. Similarly, Equation (7) represents stability. In the above optimization problem, each router evaluates the response time $Response(\mathcal{F}(\tau_{max}))$ of its τ_{max} and selects the controller which processes the τ_{max} in minimum time.

IV. Load Prediction for WAN Controllers

In this section, a load prediction approach has been devised where layer-2 controller predicts the load for layer-3 controllers and further, this load information is used for initiating the migration modules.

A. Overview of Proposed Scenario

We have discussed earlier that, whenever there is a flow miss by an L2 device, the device sends a *packet_in* to the L2 controller for control decision of the respective traffic flow. The L2 controllers then analyze the traffic flow against the *packet_in* and checks whether the traffic flow will cross through the L3 WAN gateway devices. Then the L2 controller checks the presence of flow entry into the router (WAN device) against the same traffic flow. When there is no flow entry in router for respective flow generated from L2, it signifies that the same traffic flow will cause generation of *packet_in* from the router. If no rule is found, L2 controller checks the current load of L3 controller. In case, the L3 controllers' current resource utilization exceeds a threshold value, before sending a $packet_{in}$, L2 controller notifies it to L3 controller.

B. Estimating Increase in CPU Utilization of the Controller

The L3 controller informs L2 controller about its CPU utilization using an event based propagation system. L3 controller informs about its CPU Utilization level only when there is a marginal change in it. For simplicity, the CPU utilization of the controller is divided into L levels where $L \in \{0, 100\}$.

For prediction, the L2 controller requires the current CPU utilization level of the L3 controller and the flow table entries of the L3 gateway device. Whenever, there is change in the CPU utilization level of the L3 controller, the L3 controller broadcasts the change to L2 controllers. The L2 controllers cannot directly access the flow table entries of the L3 gateway device because L3 controller acts as *MASTER* for the gateway device. The L2 controller accesses the flow entries of the gateway device through L3 controller. This mirroring process is based on a specific event for example, whenever a change in the flow table occurs then only L3 controller notify to L2 controller as shown in Fig. 2. L2 controller informs L3 controller only when the following equation is satisfied:

$$CPU_{util} + 10\tau_t \geq L_i * \frac{100}{L} \tag{8}$$

where, CPU_{util} represents the last informed CPU utilization of L3 controller to L2 controller.

C. Load Detection

In the previous works ([6], [23] and [9]), the load is checked in time slots $t \in \mathcal{T}$, where $\mathcal{T}$ is the set of time slots in a day. The work in Elasticon [9] focuses on judging the CPU Utilization on the controller because the controller's response depends on processing *packet_in* requests rate and current CPU utilization level of the controller.

Let, the controller has CPU Cores from $C_i = 1$ to c where each core is responsible for processing of *packet_in* requests.

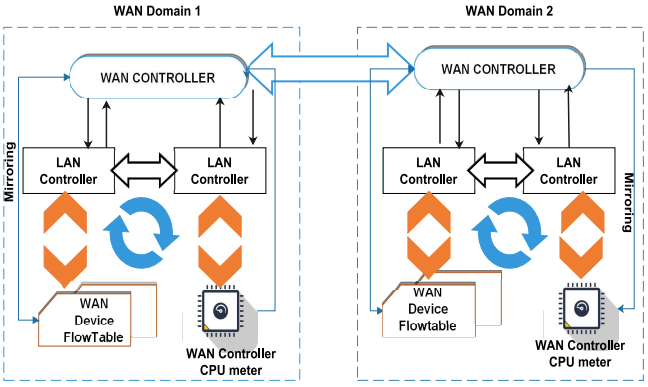


Fig. 2. Mirroring of CPU utilization and flow table in proposed control plane.

Load can be measured by the following equations:

$$L_m(t) = \mathcal{K}_L + \sum_{core=1}^{c} \left(\tau_j * \mathcal{P}_j\right) \tag{9}$$

In Equation (9), $\mathcal{K}_L$ is the initial load on the controller due to current processes going on the controller. $\mathcal{P}_j$ represents the number of packets received individually in the buffer of same type j. The available amount of CPU resource left $\mathcal{R}_{ym}$, is given as:

$$\mathcal{R}_{ym}(\cdot) = L_{th} - L_m(\cdot) \tag{10}$$

In Equation (10), L_{th} represents the threshold CPU load after which response time for *packet_in* (reply of *packet_out*) increases exponentially. The condition for load imbalance is stated as:

$$L_m(t) \geq L_{th}. \tag{11}$$

D. Overall Flow Management

Whenever the CPU utilization of the L3 controller crosses a specific threshold, the response time and device migration time increases non-linearly. In this work, we consider two threshold values, i.e., first level L_i and then final level L_{th}. The L_{th} is the final threshold, after which the response time and time of devices increase non-linearly. The first level L_i is the initial level after which the L2 controllers enter the load monitoring state. In load monitoring state, for every incoming *packet_in* the L2 controller checks whether this is going to create a load on the L3 controller. This monitoring is done by effective timeout mechanism as discussed in proposed Algorithm-1. The L2 controller's associates a timeout value for the flow entries which may cause a generation of *packet_in*, once the traffic flow arrives at L3 device. Once the L2 controller detects that specific traffic flow is going to create a load on the L3 device, it communicates this information to L3 controller (using proposed Algorithm-2).

Once the L3 controllers make migration decisions, it broadcasts this information to all other L3 controllers. In turn, other L3 controllers reply back by the Load Factor $L_F(\cdot)$. This helps the source controller to evaluates the response time. In the process of minimization of response time, the link bandwidth

from router to controller is another important factor. The load factor ($L_F(\cdot)$) is announced, if the link bandwidth from the current controller to router is more than the bandwidth from the newly selected controller. Otherwise, the new controller announces the following load factor to the router.

$$L_{Fnew}(\cdot) = L_F(\cdot) * \left\{ tan^{-1}(e^{\frac{\lambda}{10}}) - \gamma \right\} \tag{12}$$

Here, γ is a predefined constant and λ can be expressed as follows:

$$\lambda = \frac{W_{new} - W_{current}}{W_{current}} * 100 \tag{13}$$

In Equation (13), W_{new} denotes the bandwidth (or channel limit) from a new controller to the router and $W_{current}$ denotes the current link bandwidth. The load factor increases when the bandwidth from the controller to the router is more and vice versa. We observed that $tan^{-1}(e^{\frac{\lambda}{10}} - \gamma)$ returns a value between 0.7 to 1.5, which further linearly depends upon the value of λ. This function helps the load factor to be dependent on the link bandwidth, which further impacts the decision process. Finally, the L3 controller decides L3 devices to migrate to target L3 controllers using Algorithm 3.

E. Limitations of Load Predictions Module

Whenever a flow entry expires, and a new *packet_in* request for the same traffic-flow arrives at L2 controller, the controller analyzes *packet_in* request against creating a load for the L3 controller; then the L2 controller informs L3 controller synchronously or asynchronously. But, the waiting time results, delay in response for initial packets of the flow. Authors in [27] point out that while OFD latencies are in microseconds, a single round trip to controller causes additional latency of around 10-20 milliseconds. The L2 controller informs the L3 controller either using synchronized methods where it waits for the response from L3 controller to send a *packet_out* or using asynchronous methods where it does not wait for the response of the L3 controller. Both asynchronous and synchronous methods have their own limitations. The synchronous methods allow the L3 controller to take action while at the same time creating delay for the initial packets of the flow, which affects the end user experience. The asynchronous methods seem to reduce the delay for the initial packets of the flow, whereas, leaves very less time for L3 controller to take action as compared to the synchronized methods. In the result analysis section, the proposed model has evaluated using both asynchronous and synchronous methods.

F. Traffic Analysis

In this work, we analyzed layer 3 traffic from university data centers in [24] and [25]. The data-set holds the Layer 3 traffic traces for data center providing system backup services, E-mail services, distributed file system services, Web application services, and video streaming services.

This work focuses on solving the problem of predicting the load for L3 controllers using the L2 controllers by analyzing the traffic flow misses from the OF routers. The L2 controllers play the most vital role, as the controllers can directly analyze the end-users traffic. In order to analyze the end-users' traffic, the L2 controller sets the timeout values for the Layer 3 flow entries that are not present in the OF routers. In order to assign the flow timeout values, the L2 controller should know about the number of *packet_in* events caused due to end-users' traffic flow. In real time scenario, finding out the number of *packet_in* traffic flow is a very challenging task. Further, if the timeout value is wrongly calculated, the end-users traffic may suffer degradation of QoS parameters. Therefore, to find out the number of packets in a L3, we perform traffic analysis.

In [26], authors classified the flows as small, medium and large traffic flows. The small traffic flows usually have 1-2 packets, the medium traffic flows usually have 2-10 packets and large traffic flows having more than 10 packets. Moreover, from the analysis, it is clear that about 15% of flows have just 1-2 packets.

G. Timeout Computation

The delay due to processing the packets, against a particular flow in an OFD is given by:

$$F(t_i) = \frac{\mathcal{S}(i)}{m_{speed}(i,j)} \tag{14}$$

In Equation (14), $\mathcal{S}(i)$ denotes the number of packets against i^{th} flow and $m_{speed}(i,j)$ is the processing capacity (Flow Forwarding Rate) in mpps (million packets processed per second) for i^{th} flow on j^{th} OFD.

In this work, we consider the load prediction error metric and try to minimize this error.

Definition 2: The load prediction error is given by the following equation:

$$E(L) = \frac{F(t_i)}{L} \tag{15}$$

In Equation (15), *L* represents the time out value defined by the controller and $F(t_i)$ is the optimal timeout value. The error function shows two cases; one in which the time-out set by the controller *L* is less than the actual optimal timeout value $F(t_i)$ which is known as overhead error and in another case, the *L* value is more than the actual optimal time value known as the prediction error.

The overhead error occurs only when the controllers' time-out value *L* is equal to or less than the half of the actual time out value $F(t_i)$. The reason is obvious, if the flow entry already exists in the flow table and flow entry time out value is more than half of the actual time out value, then only one *packet_in* will be generated for the traffic flow and which is sufficient for the L2 controller to predict the load for the L3 controller.

In the second case, when the timeout value *L* of the traffic flow exceeds two times of the optimal processing delay ($F(t_i)$), packet processing of the corresponding flow might have completed. If the flow exists in the flow table, another traffic flow may pass undetected during that period before it expires. The timeout value has assigned as $n.F(t_i)$ (*n* usually 2 to 3 times), to ensure that the flow entries with fewer occurrences, can be taken care.The reason is obvious; if the controllers' time out value is less than two times of the optimal

timeout value, meanwhile if a flow passes before the expiry of the flow entry, then it might expires before every packet of the flow is processed. Hence the device will generate a *packet_in* to the L2 controller which is sufficient for the controller to detect the load of the L3 controller. Further, we propose a minimization of the error function, which is defined as follows:

$$Min\ E(L) \tag{16}$$

such that

$$0 < L < \infty \tag{17}$$

$$E(L) > 0.5 \tag{18}$$

$$E(L) < 2 \tag{19}$$

In Equation (17) and (19), the *E*(*L*) value has been selected more than 0.5 to avoid the overhead error and less than 2 to avoid prediction error.

H. Utility Function for End-Users

The limitations of the proposed method is that, it degrades the end-users' QoS parameters by creating additional latency for the initial packets of the traffic flow. On the other-hand, this scheme improves the TCAM (Ternary Content Addressable Memory) utilization by quickly removing the flow entries and creates the space for other flow entries. A utility function measures user's relative preference over different levels of decision metric values. Here, the objective is to maximize the end-users utility by minimizing the effect of migration on QoS metrics. The utility function depends on the value of the load factor announced by the controller. The utility of end-user is given as Equation (20).

$$\frac{\partial \mathcal{U}(L)}{\partial L} < 0 \tag{20}$$

The value less than 0 indicates, if the timeout value increases, the end users traffic flow will not suffer any extra latency for *packet_in*. At the same time, this will reduce the TCAM memory utilization.

$$\frac{\partial^2 \mathcal{U}(L)}{\partial L^2} < 0 \tag{21}$$

Equation (21) shows the marginal utility of end-users. The marginal utility represents the end-users' utility at extreme conditions. The marginal utility of the end-users increases with increase in the value of *L*. The Algorithm 1 describes the evaluation of the flow entry timeout value.

The flow timeout computation algorithm aims for computing optimal timeout value for a flow entry, such that when the traffic flow has traversed the device, the flow entry should expire from device flow table. The input parameters to Algorithm 1 is a particular flow entry *F*, OFD processing rate m_{speed}, total packets against the flow entry *S*(*F*), in turn, the algorithm returns the timeout value *L*. In Step 1, the timeout value is maintained as NULL. Then the algorithm checks the occurrence of the flow entry in terms of flow misses. If the flow miss is for the first time, then the router flow table is checked against the same flow entry as discuss in Step 3. If the flow entry exists in the router, then the flow entry is assigned a high timeout value, else the flow entry is assigned the timeout value as depicted in Step 6 where *K* is a constant value. The value of *K* is chosen such that the timeout value satisfies all subject to constraints as given in Equation (17)-(19). The function *Flow_Occurrence* (*F*) returns the number of flow miss for the flow entry *F* in a given interval of time t_i and the function *Flow_Present_Router*(*F*) returns true if the corresponding flow entry is found in Routers flow table. If the flow miss is not the first time and less than *X*, at the same time a corresponding flow entry is not found in the router (Step 8), the timeout value is assigned as discussed in Step 9. After the flow misses exceed the value *X*, the algorithm checks for the corresponding flow entry in the router. If the flow entry is present with router, a high timeout value is assigned as shown in Step 13; else L3 controller is updated for the particular flow *F* for which a high timeout value is assigned to the flow entry. In this case, L3 controller adds the flow entry in the routers' flow table. The value of *X* is chosen by the administrator based on the number of flow entries for the L2 device. The high value of timeout is computed based on multiple of $F(t_i)$, i.e., $(n.F(t_i))$. The value of *n* is decided by the administrator, where a high value of *n* determines low TCAM memory utilization, and at the same time, high flow misses and vice versa. The value of $n.F(t_i)$ should always be less than the current time slot t_i. In the above algorithm, the values of *n* and *x* can also be different for different OFDs and end-users. If a high priority users traffic is considered, the value of *X* has to be less and a value of *n* has to be high. Algorithm 1, addresses the problem of traffic flows with unknown or highly dynamic number of packets, i.e., *S*(*F*). If the number of packets in a flow entry is very dynamic, then high misses will occur against the corresponding flow entry and ultimately the number of misses exceeds *X*. When the miss exceeds the value of *X*, the L2 controller updates the L3 controller for adding the same flow entry to the router. If the router's TCAM is full, the router has to evict some existing flows to add the new flow entry.

Algorithm 1: Flow Entry Timeout Computation

Inputs : $Flow\_Entry - (F)$
Device Processing Speed-m_{speed}
$FlowPackets - S(F)$
Outputs: $Flow\_Timeout - L$

1 $L \leftarrow NULL$
2 **if** $Flow\_Occurrence(F) == 0$ **then**
3 **if** *Flow_Present_Router(F)* **then**
4 $L = HighValue$
5 **else**
6 $L = F(t_i) + K$
7 **else if** *(Flow_Occurrence(F)≤X &*
8 *Flow_Present_Router(F)==False* **then**
9 $L = 2 \cdot F(t_i)$
10 **else**
11 **if** *Flow_Present_Router(F)* **then**
12 $L = HighValue$
13 **else**
14 $Update(L3\_Controller)$
15 $L = HighValue$
16 Return L

Algorithm 2: Asynchronous Load Prediction

Inputs : $packet\_in(\tau)$
Router $\mathcal{R}$
L_i(Current CPU Utilization Level)
Outputs: $Boolean\_Variable$
$packet\_out$

1 $Boolean\_Variable \leftarrow False$
2 $packet\_out \leftarrow NULL$
3 **if** $Flow\_Entry(\mathcal{R}, \tau)$ **then**
4 $\quad Boolean\_Variable \leftarrow False$
5 $\quad packet\_out \leftarrow Process(packet\_in)$
6 **else**
7 $\quad$ **if** $CPU_{util} + 10\tau_t \geq L_i * \frac{100}{L}$ **then**
8 $\quad\quad Boolean\_Variable \leftarrow True$
9 $\quad\quad packet\_out \leftarrow Process(packet\_in)$
10 $\quad\quad Inform\_L3\_Controller()$
11 $\quad$ **else**
12 $\quad\quad Boolean\_Variable \leftarrow False$
13 $\quad\quad packet\_out \leftarrow Process(packet\_in)$

Algorithm 3: Effective Migration Decision

Inputs : Route_Set $\mathcal{R}$
$L3$(L3 Controller Set)
Outputs: $Target\_Controller$

1 **if** $Real\_Time\_Load(L_i,) \geq Th$ **then**
2 $\quad$ High_Priorities_Routers = Check_Traffic_Stats($\mathcal{R}_i$, H)
3 $\quad$ Low_Priorities_Routers = Check_Traffic_Stats($\mathcal{R}_i$, L)
4 $\quad Migrated\_Devices = 0$
5 $\quad$ **while** $CPU[\mathcal{R}_i] \leq L_i$ **do**
6 $\quad\quad$ **if** $Migrated\_Devices \leq \frac{3}{4} \cdot |\mathcal{R}|$ **then**
7 $\quad\quad\quad$ Router = Select_Any_Router(Low_Priorities_Routers);
8 $\quad\quad\quad$ New_Controller(Router)=Get_Lest_Loaded _Controller(Router[τ_{max}]);
9 $\quad\quad\quad Migrated\_Devices++$;
10 $\quad\quad$ **else**
11 $\quad\quad\quad$ Router = Select_Any_Router(High_Priorities_Routers);
12 $\quad\quad\quad$ New_Controller(Router)=Get_Lest_Loaded _Controller(Router[τ_{max}]);
13 $\quad\quad\quad Migrated\_Devices++$;
14 **else**
15 $\quad$ Wait();
16 $\quad$ Return $Target\_Controller$;

For the synchronous load prediction, L2 controller has to wait for the action of the L3 controller. In this mechanism, the L2 controller has to inform L3 controller but will not wait to receive an acknowledgment from the controller. Simultaneously it processes the *packet_in* event to generate *packet_out* message. The asynchronous method of load prediction has summarized in Algorithm 2. In Step 3, flow entry corresponding to a *packet_in* in L2, is checked in the flow table of the router. If the flow entry is found then the algorithm returns *Boolean_Variable* as false and a *packet_out* is generated for processing $packet_{in}$ in Step 5. But, if the flow entry is not found then overload condition is checked by using Equation (8) and if this flow request will overload controller then *Boolean_Variable* becomes true and information of the upcoming load is passed on to L3 controller (Step 9). The L2 controller waits for the acknowledgment from L3 controller and then generates *packet_out* for the respective *packet_in* requests as in Step 11. If the controller doesn't get overloaded then forwarding rules are responded for respective packet requests. In Step 10, the algorithm causes a delay in response to *packet_in* requests and its effects are evaluated on the predicted load in the result reported in Section V.

I. Router and Controller Selection

The main aim of load prediction or adaptive load balancing is to create a traffic aware load balancing mechanism for L3 controllers. The major bottleneck in load balancing is the constant time which occurs during device migration. Therefore, during the device migrations, the end-users suffer degradation of QoS. In the proposed approach, the L2 controller analyzes each traffic flow and predicts the load. During the analysis phase, L2 controller can view the packet headers of each traffic flow. Using this information, L2 controllers can help to L3 controllers in the selection of optimal OFD for migration such that high priorities users will not suffer degraded QoS during device migrations.

In Algorithm 3, L3 controller is notified for load change information from all L2 controllers as shown in Step 1. Then L3 controller performs traffic statistics on the L2 devices, where L2 controller collects flow-based statistics and passes to L3 controller. These statistics include bandwidth consumption for different flow entries (by sending FLOW_STATS_REQUEST). Based on the flow entries, L3 controller identifies the high priority end-user traffic using the priority assigned to each flow entry. If in a time slot t_i a high priority traffic flow entry has high bandwidth, the same router is marked as high priority router as shown in Step 2 and 3. Then based on the value of τ_{max} for a router a suitable controller is selected based on the controllers current CPU utilization level as described in Steps 4 - 11. The *Check_Traffic_Stats*() takes Router set $\mathcal{R}_i$ as input and indicator *H* or *L*. This function returns all routers in order of their flow rules and priorities. The function checks the traffic statistics against the flow entry rules and uses the priority field of the flow entry for prioritizing the routers. If the indicator passed is *H*, the function returns the routers in descending order and vice versa. Finally, we control the total number of migrated devices using the *Migrated_Devices* variable. The while loop continues until the CPU utilization of the source L3 controller comes below L_i level. The function $Get\_Least\_LoadedController(\cdot)$ returns a new controller which has minimum response time for the control packets for the router. This function evaluates the response time for all available L3 controllers based on their announced load factors value.

V. Performance Evaluation

In order to implement the proposed work, 12 extreme switches (Extreme Summit X440-24p) have been used, with OpenFlow support 1.3v. For simulating the control plane, Java-based Floodlight [11] controller has been used. We created 30 dynamic flow-entries and 160 static flow entries (ARP entries) with infinite time out values. The controllers communicated among themselves using Java's socket programming. In the

topological ordering, 3 switches were deployed in LAN environment and 4 switches are deployed in WAN environment. To simulate the WAN link, the queues have created which were bandwidth limited. The switches in WAN were deployed and rate limiting was accomplished using bandwidth limited queues so that the OF switches can simulate as routers. The WAN link bandwidth used during performance evaluation has fixed to 54 Mbps and the delay is 15 ms.

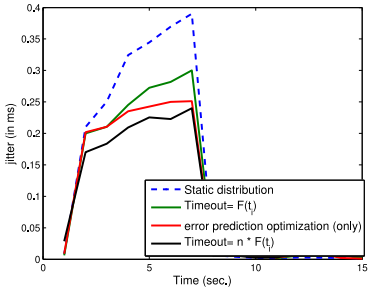
For the performance evaluation, the following metrics have been considered:

- *QoS Parameters:* The whole process of load prediction causes degradation of end-users traffic in terms of QoS parameters. For instance jitter and packet loss parameter has been considered.
- *Prediction Error:* The overall goal of the proposed work is load prediction. The prediction rate depends on the traffic dynamism and optimal solution of the flow entry timeout variable.
- *End-Users' Utility:* The improvement of end-users' utility is also considered in the proposed work based against QoS parameters. The end-users' utility is measured by averaging the drop in different QoS parameters on the percentage scale.
- *Controller Communication:* Controller communication plays an important role in the improvement of end-users utility. We evaluate L3 controller to L2 controller communication using two methods, such as: synchronous and asynchronous methods.

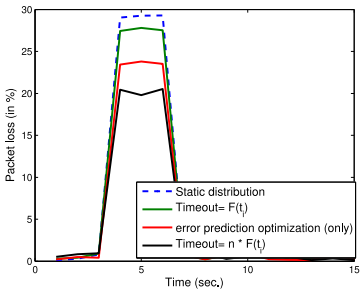
For further evaluation,the proposed work has compared with the following conditions:

- *Timeout With F(t_i):* The evaluation of timeout using $F(t_i)$ produces the optimal value of timeout, whereas this strategy is not suitable for high flow dynamism.
- *Timeout With N.F(t_i):* The evaluation of timeout using $N.F(t_i)$ does not gives the optimal value. But this strategy can handle high traffic dynamism and at the same time can decrease the TCAM memory utilization.
- *With Error Optimization:* Here, the optimization only depends on the error prediction minimization as in Equations (8)-(11).
- *Static Distribution:* The flow entry time out values and a minimum number of flow entries are assigned statically which remain same for every time slot.

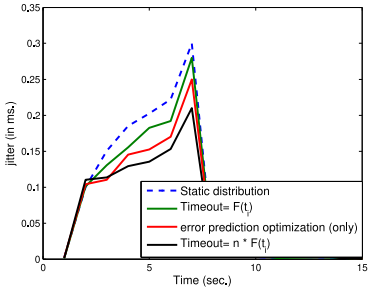
The proposed work evaluates the flow entry timeout using the packets in the traffic flow and device processing rate. The device processing rate is measured in Million Packets Processed per Seconds (Mpps). The device processing rate for X440 is 72.5 Mpps. Further finding out the number of packets *S*(*F*) for a given flow is achieved by observing the traffic statistics. As mentioned in the above traffic analysis in Section IV, the number of packets against a flow entry can be determined statistically. But there can be some traffic flows, where the flow packets vary dynamically. The proposed Algorithm 1 takes care of such traffic flows by adding the flow entry in the router flow table. The time slot t_i is evaluated using maximum length traffic flow ($t_i = (N+2)F(t_i)$, where $F(t_i)$ evaluates the optimal flow time for traffic flow with maximum length).



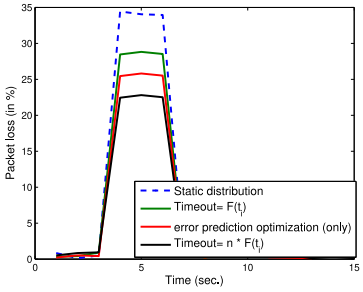
(a) Jitter (synchronous)



(b) Packet loss (synchronous)



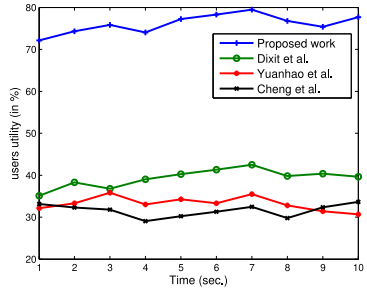
(c) Jitter (asynchronous)



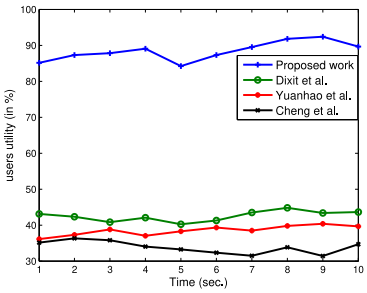
(d) Packet loss (asynchronous)

Fig. 3. Impact of QoS during i) synchronous ii) asynchronous mode of communication between L3 and L2 controller.

Based on the proposed algorithm, at the end of a time slot, t_i the miss history of the flow entries is flushed out. The graphs in Fig. 3 represent the variations of QoS metric during the communication between L3 and L2 controller. Both asynchronous and synchronous methods are evaluated against only TCP traffic. The Fig. 3(b) and 3(d) represent the variation of packet loss for synchronous and asynchronous methods that are evaluated against TCP traffic. From the plots represented in Fig. 3 it can be observed that the overall QoS parameters are degraded for initial packets of the traffic flows in all cases. In static



(a) End-Users' Utility(Low Priority)



(b) End-Users' Utility(High Priority)

Fig. 4. Average end-user utility.

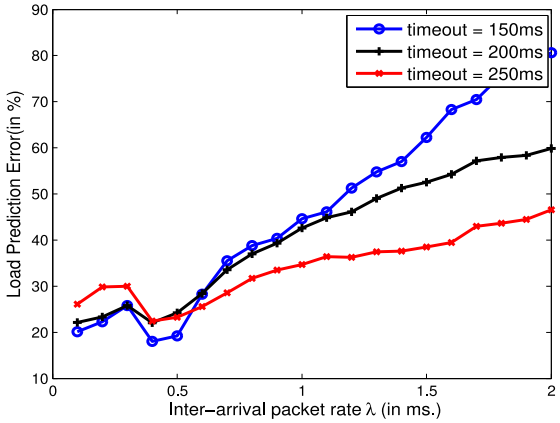


Fig. 5. Load Prediction Error for N = 100.

distribution, the QoS degradation is even more as compared to other. In static timeout distribution, the flow misses is very high as this strategy cannot well handle traffic dynamism. The time out strategy using $F(t_i)$, increases the flow misses for dynamic traffic flows. Whereas, the evaluated timeout value shows very low degradation in QoS because the flow miss is minimal.

But this strategy causes high TCAM memory utilization. Lastly, the timeout strategy using proposed Algorithm 1 has lower flow misses upto the value of *X*. In another experiment, Fig. 6(a) represents an average of end-users utility. For comparing the proposed approach, the following approaches have been considered.

- In [6], for shifting the load from one controller to other, Yuanhao *et al.* have made use of distributed decisions. To find the load on the controller, the following equation has been used by the author.

$$\rho = \frac{\frac{1}{n}\sum_{i=1}^{n} L_i}{max_{i=1}^{n} Li} \tag{22}$$

where, $\{L_1, L_2, L_3, \ldots, L_n\}$ represent the loads on various controllers. If the value of ρ approaches 1, then that would indicate that the load is uniformly distributed. If ρ approaches 0.7, then the controller can afford to have more load. For the selection process, the OFDs are checked for the following condition:

$$L_{migrate} \leq \frac{L_{overloaded} - L_{target}}{2} \tag{23}$$

The $L_{migrate}$ represents the load on the controller by the OFD's load on the controller. The $L_{overloaded}$ represents the controller which is overloaded and L_{target} represents the target controller.

- In [9], an elastic and distributed control architecture has been proposed by Dixit *et al.* In their work, CPU utilization metric has been used to check load imbalance criteria. Therefore, the load has been transferred to the controller which has minimum CPU utilization.
- In [23], optimal switch has been selected by the over loaded controller. A zero sum game is formed among switches and controllers. Switches being treated as the commodities and controllers as players. For calculation of payoff α of each player, authors proposed the following equation:

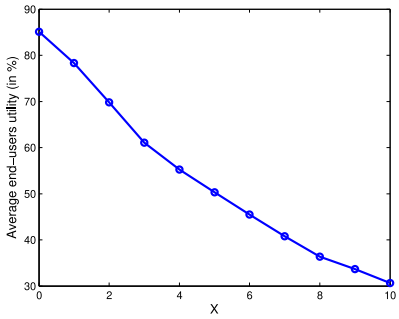
$$B_j(f') = B_j(f) + \lambda(s_i) \tag{24}$$

The f' and f are the network configurations before and after the switch (s_i) migrations. And $\lambda(s_i)$ depends upon the event count for the controllers (migration events).

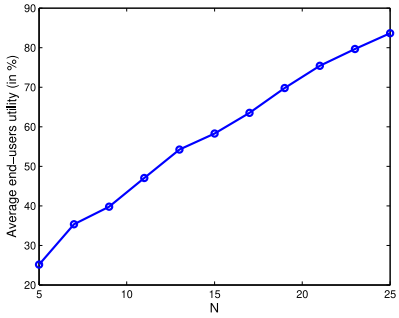
Fig. 4 shows that the proposed approach exhibits higher end-users utility due to less processing delay for both low and high priority traffic. Fig. 5 represents the load prediction error with varying inter-arrival time. With higher value of λ, the proposed approach outperforms different competing schemes. The reason is that, the proposed approach considers both prediction error minimization along with processing delay minimization.

For this experiment three different time-out values have been set. From the Fig. 5, it can be observed that the error is minimal for only a specific inter-arrival time. When the flow entry timeout is calculated by using $F(t_i)$, the error is minimum. Otherwise, the error would have significantly increased

The Fig. 6(a) and 6(b) show the average end-users' utility for different values of *X* and *N*. It can be noted that with an increasing value of *X*, the average flow misses increases, and the end-users' utility reduces. Fig. 6(a) shows a steep slope when the value of *X* is in between 1-4, and the slope decreases when the value of *X* further progresses. The reason is that, when the value of *X* is in the range of 1 to 4, the traffic flow that hits the OFD, shows less dynamism in terms of number of packets. Whereas when the value of *N* increases, the flow miss decreases, and in turn, the average end-user utility improved, as shown in Fig. 6(b).



(a) End-user utility with varying X



(b) End-user utility with varying N

Fig. 6. Average end-user utility with varying X and N.

VI. Conclusion

The proposed work focuses on a new control plane architecture, where the control decision demands are handled separately. Based on the control plane architecture, the L2 controllers predict the load for L3 controllers. Further, the load prediction approach downgrades the QoS parameters for the initial packets of the flow. To solve the initial delay, the proposed approach solves a single-objective optimization problem considering by minimizing the load prediction error. The timeout strategy also checks the high traffic dynamism and hence increases the end-users utility. We evaluated the performance of the algorithms in synchronous and asynchronous modes on real switches. The proposed work outperforms other competing schemes in optimal control load distribution because it ensures improved results in terms of QoS, prediction error, and end users' utility. In the future work, this scheme will be evaluated on a real-time topology in a heavy traffic scenario.

References

[1] A. Blenk, A. Basta, M. Reisslein, and W. Kellerer, "Survey on network virtualization hypervisors for software defined networking," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 1, pp. 655–685, 1st Quart., 2016.

[2] Y. Zhou *et al.*, "A load balancing strategy of SDN controller based on distributed decision," in *Proc. IEEE 13th Int. Conf. Trust Security Privacy Comput. Commun.*, 2014, pp. 851–856.

[3] D. Levin, A. Wundsam, B. Heller, N. Handigol, and A. Feldmann, "Logically centralized: State distribution trade-offs in software defined networks," in *Proc. 1st Workshop Hot Topics Softw. Defined Netw.*, 2012, pp. 1–6.

[4] Y. Ganjali and A. Tootoonchian, "HyperFlow: A distributed control plane for openflow," in *Proc. Internet Netw. Manag. Conf. Res. Enterprise Netw.*, vol. 3, 2010, p. 3.

[5] T. Koponen *et al.*, "Onix: A distributed control platform for large-scale production networks," in *Proc. 9th USENIX Conf. Oper. Syst. Design Implement. (OSDI)*, vol. 10, 2010, pp. 351–364.

[6] F. Yonghong, B. Jun, W. Jianping, C. Ze, W. Ke, and L. Min, "A dormant multi-controller model for software defined networking," *China Commun.*, vol. 11, no. 3, pp. 45–55, Mar. 2014.

[7] W. Yong, T. Xiaoling, H. Qian, and K. Yuwen, "A dynamic load balancing method of cloud-center based on SDN," *China Commun.* vol. 13, no. 2, pp. 130–137, Jan. 2016.

[8] G. Cheng, H. Chen, Z. Wang, and S. Chen, "DHA: Distributed decisions on the switch migration toward a scalable SDN control plane," in *Proc. IFIP Netw. Conf. (IFIP Netw.)*, 2015, pp. 1–9.

[9] A. Dixit, F. Hao, S. Mukherjee, T. V. Lakshman, and R. R. Kompella, "ElastiCon; an elastic distributed SDN controller," in *Proc. ACM/IEEE Symp. Archit. Netw. Commun. Syst. (ANCS)*, 2014, pp. 17–28.

[10] K. Lee *et al.*, "MC-SDN: Supporting mixed-criticality real-time communication using software-defined networking," *IEEE Internet Things J.*, vol. 6, no. 4, pp. 6325–6344, Aug. 2019.

[11] *Floodlight Controller OpenFlow*. Accessed: Jan. 10, 2019. [Online]. Available: http://www.projectfloodlight.org/floodlight/

[12] S. H. Yeganeh, A. Tootoonchian, and Y. Ganjali, "On scalability of software-defined networking," *IEEE Commun. Mag.*, vol. 51, no. 2, pp. 136–141, Feb. 2013.

[13] F. Benamrane, M. B. Mamoun, and R. Benaini, "Short: A case study of the performance of an openflow controller," in *Proc. Int. Conf. Netw. Syst.*, 2014, pp. 330–334.

[14] Y. Jarraya, T. Madi, and M. Debbabi, "A survey and a layered taxonomy of software-defined networking," *IEEE Commun. Surveys Tuts.*, vol. 16, no. 4, pp. 1955–1980, 4th Quart., 2014.

[15] P. Xiao, W. Qu, H. Qi, Z. Li, and Y. Xu, "The SDN controller placement problem for WAN," in *Proc. IEEE/CIC Int. Conf. Commun. China (ICCC)*, 2014, pp. 220–224.

[16] Y. Jiménez, C. Cervelló-Pastor, and A. J. García, "On the controller placement for designing a distributed SDN control layer," in *Proc. IFIP Netw. Conf.*, 2014, pp. 1–9.

[17] S. Lange *et al.*, "Heuristic approaches to the controller placement problem in large scale SDN networks," *IEEE Trans. Netw. Serv. Manag.*, vol. 12, no. 1, pp. 4–17, Mar. 2015.

[18] B. Heller, R. Sherwood, and N. McKeown, "The controller placement problem," in *Proc. 1st Workshop Hot Topics Softw. Defined Netw.*, 2012, pp. 7–12.

[19] D. Tuncer, M. Charalambides, S. Clayman, and G. Pavlou, "Adaptive resource management and control in software defined networks," *IEEE Trans. Netw. Serv. Manag.*, vol. 12, no. 1, pp. 18–33, Mar. 2015.

[20] M. T. I. U. Huque, G. Jourjon, and V. Gramoli, "Revisiting the controller placement problem," in *Proc. IEEE 40th Conf. Local Comput. Netw. (LCN)*, 2015, pp. 450–453.

[21] C. Coello, S. De Computación, and C. Zacatenco, "Twenty years of evolutionary multi-objective optimization: A historical view of the field," *IEEE Comput. Intell. Mag.*, vol. 1, no. 1, pp. 28–36, May 2006.

[22] N. Srinivas and K. Deb, "Muiltiobjective optimization using nondominated sorting in genetic algorithms," *Evol. Comput.*, vol. 2, no. 3, pp. 221–248, Sep. 1994.

[23] G. Cheng and H. Chen, "Game model for switch migrations in software-defined network," *Electron Lett.*, vol. 50, no. 23, pp. 1699–1700, Jun. 2014.

[24] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proc. 10th ACM SIGCOMM Conf. Internet Meas.*, 2010, pp. 267–280.

[25] *Data Set for IMC 2010 Data Center Measurement*. Accessed: Jan. 15, 2019. [Online]. Available: http://pages.cs.wisc.edu/~tbenson/IMC10_Data.html

[26] B. Ryu, D. Cheney, and H.-W. Braun, "Internet flow characterization: Adaptive timeout strategy and statistical modeling," in *Proc. Workshop Passive Active Meas. (PAM)*, vol. 105, 2001, p. 45.

[27] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "DevoFlow: Scaling flow management for high-performance networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 4, pp. 254–265, 2011.

[28] K. S. Sahoo and B. Sahoo, "CAMD: A switch migration based load balancing framework for software defined networks," *IET Netw.*, vol. 8, no. 4, pp. 264–271, Jul. 2019.

[29] J. Cui, Q. Lu, H. Zhong, M. Tian, and L. Liu, "A load-balancing mechanism for distributed SDN control plane using response time," *IEEE Trans. Netw. Serv. Manag.*, vol. 15, no. 4, pp. 1197–1206, Dec. 2018.

[30] H. Zhong, Y. Fang, and J. Cui, "LBBSRT: An efficient SDN load balancing scheme based on server response time," *Future Gener. Comput. Syst.*, vol. 68, pp. 183–190, Mar. 2017.
[31] J. Xie, D. Guo, C. Qian, L. Liu, B. Ren, and H. Chen, "Validation of distributed SDN control plane under uncertain failures," *IEEE/ACM Trans. Netw.*, vol. 27, no. 3, pp. 1234–1247, Jun. 2019.
[32] M. He, A. Varasteh, and W. Kellerer, "Towards a flexible design of SDN dynamic control plane: An online optimization approach," *IEEE Trans. Netw. Serv. Manag.*, to be published.

Kshira Sagar Sahoo received the M.Tech. degree in information and communication technology from IIT, Kharagpur, India, in 2014, and the Ph.D. degree in computer science and engineering from the National Institute of Technology, Rourkela, India, in 2019. He is currently working as an Assistant Professor with the Department of IT, VNR VJIET, Hyderbad, India. His research interest includes future generation network infrastructure, such as software defined networks, edge computing, and IoT. He is a Student Member of IEEE Computer Society and an Associate Member of the Institute of Engineers, India.

Pritish Mishra received the graduation degree from IIIT, Bhubaneswar, India. He is working as a Core Developer with SAP Cloud Platform, SAP Labs Bangalore, India. He has been contributing to many open-source projects related to the domain of cloud for over two years. He has been published papers in reputed journal and conferences.

Mayank Tiwary received the graduation degree from the Biju Patnaik University of Technology, Rourkela, India. He is working as a Core Developer with SAP Cloud Platform, SAP Labs Bangalore, India. He has numbers of publications in the domain of distributed and cloud computing.

Somula Ramasubbareddy received the master's degree in computer science and engineering in 2015. He is currently pursuing the Ph.D. degree in computer science with VIT University Vellore, India. His areas of interest are mobile cloud computing and big data analytics.

Balamurugan Balusamy received the B.E. degree in computer science and engineering from Bharathidasan University, Tiruchirappalli, India, in 2001, the M.E. degree in computer science and engineering from Anna University, Chennai, India, in 2005, and the Ph.D. degree in computer science and engineering from VIT University, Vellore, India, in 2015. He is a Professor with the School of Computing Science and Engineering, Galgotias University, Greater Noida, India. His current research interests include big data, network security, and cloud computing. He is a Pioneer Researcher in the areas of big data and IoT and has published over 70 papers in various top international journals.

Amir H. Gandomi (GS'14–M'15–SM'19) was an Assistant Professor with the School of Business, Stevens Institute of Technology, NJ, USA, and a Distinguished Research Fellow with BEACON Centre, Michigan State University, MI, USA. He is a Professor of data science with the Faculty of Engineering and Information Technology, University of Technology Sydney, Australia. He has published over one hundred and sixty journal papers and five books which collectively have been cited more than 13 000 times (H-index = 56). He has been named as one of the most influential scientific minds and a Highly Cited Researcher (top 1%) for three consecutive years, 2017 to 2019. Because of his efforts in genetic programming, he also ranked 19th in GP bibliography among more than 12 000 researchers. His research interests are global optimization and (big) data analytics using machine learning and evolutionary computations in particular. He has also served as an Associate Editor, an Editor, and the Guest Editor in several prestigious journals and has delivered several keynotes and invited talks.