Computing (2020) 102:2073–2098
https://doi.org/10.1007/s00607-019-00751-2

An energy-efficient load distribution framework for SDN controllers

Madhukrishna Priyadarsini[1] · Shailesh Kumar[1] · Padmalochan Bera[1] · Mohammad Ashiqur Rahman[2]

Received: 16 March 2019 / Accepted: 17 August 2019 / Published online: 23 August 2019

Abstract
Software-defined networking (SDN) has evolved as an effective platform for future Internet due to its capability of configuring the network dynamically with varying requirements. It has been observed that the load and energy requirement of SDN devices increase significantly with the growth of communication networks. Therefore, there is a need for efficient modeling of SDN controller that can balance the load as well as optimize the energy consumption by the devices. In this paper, we present an energy-efficient load distribution framework; controller system model for efficient load distribution and routing of traffic that objectively optimize the energy consumption in the network. Our model balances load according to the heterogeneous traffic demands as well as reduces energy consumption by introducing energy-efficient routing algorithm selection procedure. The load balancing scheme is drifted by switch migration technique for multiple controllers simultaneously, whereas the novelty of energy-efficient routing lies on sleep and active mode of network devices. We present interaction between load-balancing scheme and energy-efficient routing towards the network's performance enhancement. The efficacy of our proposed controller system model is justified with extensive simulation results that show approximately 25% reduction of energy consumption and approximately 20% performance increment. Our proposed model is applicable to real-life network environment satisfying the standards of green communication.

Keywords OpenFlow · Controller · Load distribution · Energy efficiency · Performance

Mathematics Subject Classification 68M10

✉ Madhukrishna Priyadarsini
mp18@iitbbs.ac.in

Extended author information available on the last page of the article

1 Introduction

The large-scale digitization of data and the growth of the Internet in various applications of enterprise domains, call significant changes in the communication technologies and network platforms [1]. Software-defined networking (SDN) is an emerging platform, unlike the traditional network, effectively manages heterogeneous traffic depending on changes in requirements [2].

1.1 SDN overview

One of the important aspects of SDN is that it provides a structured software environment for evolving network-wide abstractions while potentially making the data plane simple [3,4]. In SDN, a logically centralized control plane functionally manages the network and abstracts the underlying network infrastructure to the applications [5]. SDN controller communicates with the application layer using the northbound protocol to implement the application-level requirements, and southbound protocols generate the flow tables [6,7]. Then the generated flow tables are assigned to the OpenFlow switches for efficient and effective distribution of traffic in SDN [8]. By acquiring network topology information, the SDN controller provides a global network view for OpenFlow switches and implements the flexible network configuration with network management functions [9].

1.2 Traditional network versus SDN

In comparison to the traditional network, the SDN platform can effectively solve robustness by network function virtualization, and redundant implementation of the network functions in the controller [10,11]. The potentiality of SDN lies in its capability of reconfiguring the network with changes in network protocols, the inclusion of new services and applications. Despite its advantages, there exist few major challenges in SDN such as dynamic traffic load distribution [12], optimizing energy consumption [13], and performance enhancement [14].

1.3 Problem discussion

With the growth and usages of network applications; there is a need for managing heterogeneous traffic in the network [15]. Therefore, one of the major challenges in SDN is to distribute traffic among the controllers appropriately [16]. In other words, it is necessary to do load balancing effectively among the controllers. We use, load balancing and load distribution interchangeably in the rest of the paper. The problem of effective load distribution in the SDN controllers can be solved using load balancing techniques with other objectives such as efficient resource utilization [17], maximizing the throughput [18], reducing the delay [19]. A highly loaded controller in a network consumes more energy resulting in degradation of performance [20]. Also, in-efficient route selection process may lead to the routing of traffic through the route that consumes high energy. The state-of-art research on load balancing techniques is mainly on centralized [17], distributed [16], load informative based design [12], the trade-off between switch migration cost and load balancing rate [18], and multiple controller-switch migration [19]. However, none of these techniques describes the mapping of an overloaded switch to the target controller under high load condition and about the adjustment of the threshold value.

On the other hand, the researches on energy efficiency in SDN focuses on the reduction of energy consumption in OpenFlow switches and link paths [21]. However, they do not consider the energy consumption of the controllers, which is one of the significant factors in calculating overall energy consumption in a network.

It has also been reported that network devices consume 50% of the total energy consumption when the traffic is low [22]. The network, as a crucial component of data center infrastructure, consumes a significant part of total energy (up to 20 %). Therefore, energy-optimization has become a key challenge in performance evaluation of SDN.

Energy-efficiency is a seamless integral part of load balancing. Because during load balancing, the controllers communicate with each other through message passing. This, in turn, makes the devices and link paths active, causing energy consumption. No state-of-art research in this domain highlighted the effect of load on energy consumption and how they are related to green communication. The modern standards of green communications mainly motivate our work of inducing energy efficiency in the communication network for less carbon dioxide emission [14,23]. This motivates us to formulate our current research problem with the following constraints.

(1) Generation of flow rules for heterogeneous real-time traffic versus performance
(2) An in-efficient distribution of load versus routing delays
(3) The routing of traffic under the load balancing process versus energy consumption versus performance and resource utilization

1.4 Contributions

The main objective of this paper is to provide a controller system model (CSM) that integrates load balancing and energy efficiency in SDN. The major contributions of our work are as follows:

(1) An efficient load balancing scheme that effectively distributes the heterogeneous traffic load among the controllers.
(2) (a) Implementation of sleep-active mode mechanism in devices for reducing energy consumption.
    (b) A heuristic-based routing algorithm called EERAS (energy-efficient route selection) for efficient route selection with optimized energy consumption.
(3) Integration of load balancing and energy-efficient routing. The integration finds a trade-off between performance and energy-optimization in the network.
(4) An extensive experimentation with varying network size to identify the efficacy and usability of our proposed controller system model.

The rest of the paper is organized as follows. Section 2 presents the related works on energy efficiency and load balancing of the SDN environment. In Sect. 3, we define the problem and present the motivation behind our work. We describe our proposed controller system model, including load balancing and energy-efficient routing framework in Sect. 4. Section 5 presents the performance evaluation of our controller system model with experimental analysis. We conclude with future work in Sect. 6.

2 Related work

In this section, we highlighted some important state-of-art works in both load balancing and energy efficiency domains of SDN. The first subsection explains existing works in SDN control plane load balancing, and the second subsection explains energy efficiency issues and their proposed solutions.

2.1 SDN control plane load balancing

There has been certain research work to address different issues in control plane load balancing, starting from a centralized load balancing approach to distributed load balancing and later on load informative strategy for load balancing. Some of them are mentioned in this section.

In centralized load balancing approach [17], a single controller is responsible for balancing the load. It periodically collects load from other controllers, informs the overloaded controller to transfer some of its load to a lightly loaded controller. Then it sends the load migration command to the target controller, followed by the switch's load migration to the controller with minimum load. The key issue in this approach is latency, which includes the time elapsed since a controller gets highly loaded, due to load collection by the master controller. Load on a controller changes dynamically, and this load balancing strategy lags behind the real load on the target controller.

In the distributed load balancing approach, DALB [16] each controller balances its load. Here, a threshold is defined for each controller, and no load balancing is required until that value. When the load on a controller increases its threshold, it does load collection, and then it may go for load balancing. In this case, most of the controllers would collect load continuously because of their load more than the threshold. This would consume bandwidth available for the data plane as a large number of controller O(n^2) message exchange between controllers.

In load informative strategy LI [12], each controller broadcasts its load, so one particular controller does not require to do the load collection, which saves time and does load balancing as quickly as possible. The load scale from 0 to threshold is segmented into many pieces ($V_0 = 0, V_1, V_2, \ldots V_n =$ Threshold), where segment size decreases gradually that is $|V_i - V_{i-1}| < |V_{i-1} - V_{i-2}|$. The load would be informed only if previously informed load, and current load lies in two different segments.

SMDM based load balancing [18] proposed a trade-off between switch migration cost and load balancing rate. However, the migration at the source controller leads to another load migration at the target one, which is the vital reason for performance degradation. In SMCLBRT load balancing scheme [19] multiple controllers migrate their switches simultaneously depending on the response time of the target controller.

One set contains the highly loaded controllers and another lightly loaded controllers. In the first run of this scheme, controller-switch pairs are found, and in the second phase, all the selected switches are migrated. This scheme consumes high network bandwidth and introduces congestion due to simultaneous switch migration. A load-balancing scheme on switch groups [20] proposed migration of a group of switches to reduce the number of decisions. However, it increases load balancing time and migration cost, which effectively reduces the performance of the network.

There exist some different research aspects of controller load balancing, such as controller placement in SDN networks, load distribution over multi-path networks, flow-assignment, and packet scheduling. Lange et al. [24] presented a framework which shows operators with Pareto optimal placements concerning different performance metrics and one heuristic approach, which helps in the analysis of the resulting trade-off between time and accuracy. It also solves virtual functions placement problems in the context of Network Functions Virtualization (NFV). Wu et al. [25] proposed an adaptive flow assignment and packet scheduling framework which integrates the channel resources in heterogeneous wireless networks to maximize the aggregate goodput and spreads out the packet's departures over multiple communication paths within the delay constraint to mitigate burst losses.

However, each of the state-of-art techniques experiences a high network bandwidth requirement and congestion, which leads to performance degradation.

2.2 Energy efficiency in SDN

Wei et al. [26] presented the energy-efficient traffic engineering problem in hybrid SDN/IP networks. They formulated a mathematical model considering SDN/IP hybrid routing mode and proposed one algorithm to solve the energy-efficient traffic engineering problem. Their algorithm considers the IP routers which perform the shortest path routing using distribute OSPF link weight optimization based on neighboring region search and split the traffic at the SDN enabled switches by the global controller.

Bolla et al. [27] proposed energy management primitives in the context of the emerging Software Defined Networking. It increases networking flexibility, the OpenFlow Protocol to integrate the energy-aware capabilities offered by the Green Abstraction Layer (GAL). It also proposes an analytic model for the management of a network with these capabilities. In Nam et al. [28] a new energy-saving scheme that can flexibly control and route traffic relying on the difference of network device's energy-profile is discussed. The energy-saving scheme using OpenFlow makes use of the energy profile of network devices and switch port under various link rates.

Cruz et al. [13] focused on energy consumption optimization in SDN switches and links, their associated rates, the number of flow entries at each SDN switch. Alberto et al. [29] proposed an energy-aware and policy-based system-oriented SDN paradigm, which allows managing the mobile network dynamically at run time and on-demand through policies. In their work, they reduced energy consumption by switching off the unused devices.

From the above-related works, we infer the following key observations:

(1) The current SDN state-of-art energy consumption reduction mechanism does not address energy-optimization in controllers which decides flows of the SDN switches and routers.
(2) Also, the existing energy-optimization solutions do not consider the effect of routing algorithms on energy consumption calculation. In real-time traffic scenario, at the time of route generation for any packet, the controller may consider existing routing algorithms or create its routing algorithm. For executing each routing algorithm, some energy is required, and that affects the energy consumption by a controller and thus by SDN.

In summary, none of those mentioned works considers energy consumption reduction through balancing controller load and selecting an efficient routing algorithm. In the next section, we present the problem statement of this paper with the motivation behind our work.

3 Motivation and problem definition

Here in the first subsection, we describe the motivation of our work, and in the second subsection, we present the problem statement.

3.1 Motivation

Today, a large number of heterogeneous applications are being executed in the backbone network of any organization or publicly accessible network [30]. Increase of traffic with varying requirements trivially cause load balancing issues in the controllers, which leads to performance degradation. To maintain and enhance network's performance, energy-efficient load balancing in SDN controllers, switches and links in live networks with varying input is necessary. Here, we provide motivating examples to demonstrate the significance of our research.

(1) Infrastructure less network for disaster: In the disaster scenario, public infrastructure network such as cellular and WiMAX network cannot work. However, one or more local network elements like a mobile node can establish communication channels for other nodes. In such a case, it is beneficial to use a real-time application such as VoIP to establish node-node communication. Therefore, there is a need for continuous VoIP communication in the affected area. In this context, the key requirements include Quality of Service (QoS), energy efficiency, robustness, and reliability. Software-defined network with optimized energy consumption may be a potential solution in this scenario.
(2) Data center network: Increasing amount of dynamic heterogeneous traffic creates load imbalance and consume a high amount of electricity in data centers every year to provide reliable and stable services (shown in Table 1, where the growth of data centers are reducing). High energy cost has become one of the vital concerns for large-scale data centers. The proportion is even rising due to the

Table 1 Energy consumption by global data centers

Year	Energy consumption (unit Twh)	Growth proportion (%)
2014	7202.9	16.8
2015	7998.3	14.3
2016	8842.7	13.8
2017	9102.9	12.6

rapid development of energy conservation technologies on servers and cooling systems. Thus the energy-optimization framework is beneficial.

(3) Green communication: With the exponential growth of the Internet use, many organizations use a large amount of energy to operate and control the cooling system in backbone network infrastructures and thus produce a significant amount of carbon waste. The Global e-Sustainability Initiative (GeSI) 2010 estimated that network energy requirement in European telecom operator was about 21.4 terawatt-hours (TWh) and forecasted a figure of 35.8 TWh in 2020 if no energy-saving initiatives are adopted. There is a need for energy consumption reduction to control carbon emission and make the environment green.

Considering the aforementioned application and deployment scenarios, the main objective of our work is to develop an optimized energy-efficient load balancing framework for SDN, which optimizes energy consumption by intelligently distributing the load.

3.2 Problem definition

In any SDN topology, let C_i as the ith controller with the corresponding load CL_i, which is defined as the sum of incoming packets from the connected switches. Given M switches set $S = \{S_1, S_2, \ldots, S_M\}$ with their load SL_i and N controllers set $C = \{C_1, C_2, \ldots, C_N\}$. The objective is to uniformly distribute the load among all the controllers present in the network with optimized energy consumption by the devices, which leads to better performance and reduces the time complexity.

Every controller C_i has a potential switch set, and it can only control the flows from that set of switches. To maintain the performance and manage the network, each controller should finally have almost the same amount of load. Otherwise, some controllers load increase and they will not be able to manage the switches from their respective switch set. To more precisely define the problem we take base threshold BT of each controller as our load limit, current load as CL and mathematically define the load balancing condition in each controller as $CL_i < BT$. If traffic flow increases as the system are running, the load of the controller (CL_i) may increase rapidly and make the previously mentioned condition as false, i.e., $CL_i > BT$.

Here we need to migrate some highly loaded controllers load to other lightly loaded controllers, to reduce its load and keep the total network traffic balanced. On the other hand, if simultaneously load increases in most of the controllers, then increase of BT value provides the best solution to accommodate more load. However, this condition should be applicable up to a certain extent, depending upon the hardware limit of the

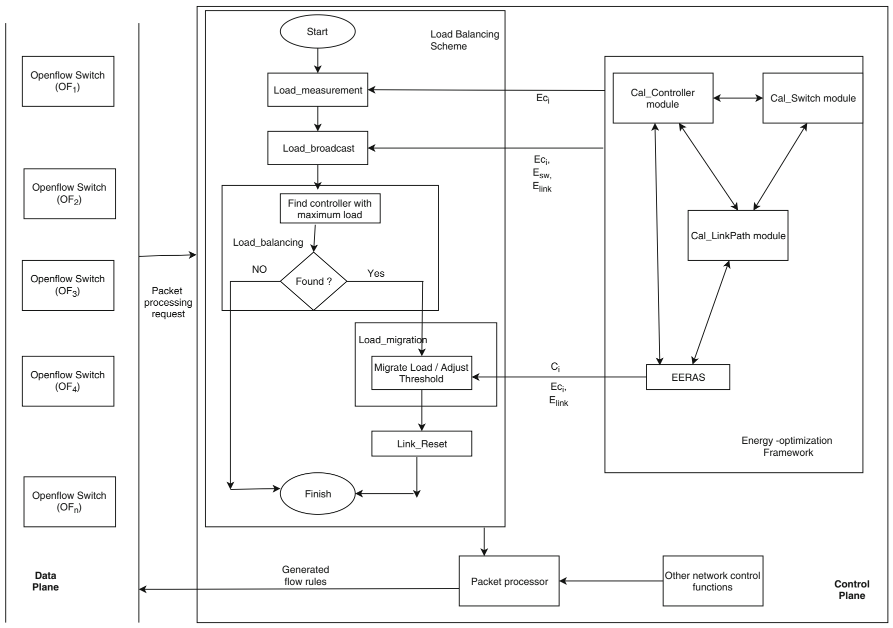


Fig. 1 Functional architecture of proposed controller system model

controller. Also, the load on SDN controllers are dynamic and application-specific, which leads to more energy consumption by the network devices. This leads to more CO_2 emission to the environment and affects the greenhouse gases.

To provide energy-efficient load distribution in the SDN environment, we need to design a framework which not only balances network load but also provides minimum energy consumption by the devices with suitable routing algorithm selection. In the next section, we describe our proposed controller system model in detail.

4 Proposed controller system model

In this section, we present the proposed controller system model (CSM), including a load-balancing scheme and energy-optimization framework.

Our proposed model runs inside each controller present in the network. Whenever the load is high on each controller, our model balances load using proposed self-adaptive load balancing scheme (SALB). During load balancing, it needs to migrate some of the switches from one controller to another. The migration requires efficient routing algorithm selection and less energy consumption for better QoS and reliability, the stability of the network. Our proposed energy computation model and EERAS algorithm guarantee energy-efficient routing during the load balancing process. After migration, the network is lightly loaded, consumes less energy, provides high performance. Figure 1 shows the overall architecture of our proposed controller system model with the flow between different load balancing components and energy-optimization framework.

The key components of our load balancing scheme (SALB) includes: $Load\_measurement$, $Load\_broadcast$, $Load\_balancing$, $Load\_migration$, and $Link\_reset$. On the other hand, our energy-optimization framework consists of the following two components:(1) Energy computation model covering three modules namely, $Cal\_Controller$, Cal_Switch, $Cal\_LinkPath$, and (2) EERAS algorithm for optimizing energy consumption by the controllers and the links for efficient routing of traffic. One of the key features of our optimized energy consumption model lies in the introduction of two important modes of operation, namely, $Sleep$ and $Active$. In $Active$ mode, the network devices receive OpenFlow messages without any delay and subsequently process them. On the other hand, in $Sleep$ mode, the devices store the incoming packets in a queue and process them after a random interval of time. In general, energy computation considers all the $Active$ state devices only.

In the proposed controller system model, we establish an interaction between optimized energy consumption framework and the three components of the load balancing scheme, i.e., $Load\_measurement$, $Load\_broadcast$, and $Load\_migration$. The energy-optimization framework is triggered by load measurement component to calculate the energy consumption of the controller. Also, the $load\_broadcast$ component activates the energy-optimization framework to calculate the energy consumption of the switches, controllers and the link paths. Afterward, the $Load\_migration$ component sends the selected controller-switch pair to the energy-optimization framework. Then, the $EERAS$ algorithm finds the energy optimized route between the source and target controllers based on the results from its different modules, e.g., $Cal\_Controller$, Cal_Switch, $Cal\_LinkPath$. This energy-efficient route is communicated to the $load\_migration$ module. Finally the $Link\_reset$ component (LR) resets the network state to its initial topology. This component runs periodically once in a day to bring the network to its original state after a long run of the load balancing process. Our proposed controller system model provides efficient traffic management with the reduction of energy consumption by the network devices. The parameters used throughout the paper are summarized in Table 2. The next subsection describes our proposed load balancing scheme.

4.1 Self-adaptive load balancing scheme

In this subsection, we present our proposed self-adaptive load balancing scheme SALB in detail with the functionality of all the components. Firstly, we introduce the basic terminology used for the proposed scheme and their estimation process. Then we describe the five components of the load-balancing scheme with flow diagrams and algorithms. In our scheme, we model the base threshold of a controller as a function of three parameters namely $clock$, $inBW$, $outBW$ and described as $f(clock, inBW, outBW)$ where $clock$ is the time required by the controller to process one packet request, $inBW$ is the packet-in bandwidth to the controller and $outBW$ is the packet-out bandwidth from the controller in an average [31]. We derived the BT value for each controller by varying the $clock$, $inBW$, and $outBW$. We model current threshold CT as $\alpha * BT$, where α measured as the ratio between CL and BT. The α is set to 1, and it varies

Table 2 Notations used in controller system model

Notation	Description
CL	Current load of controller
BT	Base threshold
CT	Current threshold
α	Parameter used to calculate current threshold = CL/BT
δ	Maximum load error = 10% of BT
ρ	Parameter used for threshold reset condition = 20% of BT
β	Parameter used for representation of high load
N	No. of controllers
LLC	Set of lightly loaded controllers
HLC	Set of highly loaded controllers
U	Set of switches under highly loaded controller
SL	Switch load
k_1, k_2, k_3	Weight parameters varies with network and converges after certain limit
LT	Latency of switch
E_{link}	Active links energy consumption
E_{swi}	Energy consumption by ith switch
E_{ci}	Energy consumption by ith controller
$\lambda * T$	Arrival rate of packets
Q_{size}	Overall queue size of the network
E_{sw}^{A}	Energy consumption by switches when it wake up itself
E_{sw}^{N}	Energy consumption by switches when it is activated by neighbor
E_{SA}	Conversion energy from sleep to active mode
E_s	Sensing energy
W	Weight of packet
E_{beacon}	Energy of beacon message; which uses (1−3)% of device battery
E_{oi}	Energy consumption by active port i
n	no. of ports
α' and β' =	$\begin{cases} 1; switchisactive \\ 0; switchisinsleepmode \end{cases}$
CPU_{util}	CPU utilization by controller due to different types of packets
$f(T_i)$	$f(T_{i-1} + \delta)$
$f(T_0)$	$\begin{cases} 1; receive(p) \geq 0 \\ 0; otherwise \end{cases}$
receive(p)	Function which receives no. of packets

Table 2 continued

Notation	Description
expected time for δ is $E(\delta)$	$\delta * p_r(t = \delta)$ initial δ= setup time
$p_r(t = \delta)$	Probability of selecting time t = $\frac{1}{N}$ and N is the sample space
x_i	Total traffic of link i
c_i	Capacity of link i

with the load on the controller. The operational flow of our proposed load balancing scheme with complete functionality of each component is shown in Fig. 2.

4.1.1 Load_measurement

This component periodically measures the load of each controller. We use packets arrival rate from the connected switches to the controller as a measure of the load. This is calculated as the cumulative sum of the received packets for each switch connected to the controller. $CL_i = \sum_j SL_j$.

4.1.2 Load_broadcast

It broadcasts the load to other controllers so that each controller have global knowledge about the network. The question here is how often we should broadcast the load? The periodic broadcast would not be a good solution because it consumes bandwidth available for the data plane. On the other hand, the high broadcasting period may lead to a lack of updated information in the controllers. A controller broadcasts the current load if and only if there is enough deviation from the previously informed load (PL). The parameter δ indicates maximum load deviation, which is set as 10% of BT. Our SALB scheme triggers load broadcast process if the deviation of CL and $PL > \delta$. We choose δ value as 10% of BT to avoid a large gap in broadcast time as well as frequent broadcast message transfer between controllers. This is dynamically captured during the selection of the switches to be migrated to the target controller so that it does not trigger another load migration at the target controller. The highly loaded controllers use a time out t_w for receiving load broadcast information from other controllers before executing the load balancing process. During this time period, if the highly loaded controllers do not receive any broadcast message from specific controllers due to packet drops, it discards the respective controller in the load balancing process. Here, t_w is set dynamically as the last sampled round trip time.

4.1.3 Load_balancing

After the $Load\_broadcast$ process as presented earlier, each controller has the information about the current load (CL) and base threshold (BT) of other controllers. Using these values, each controller calculates the current threshold (CT) for all other

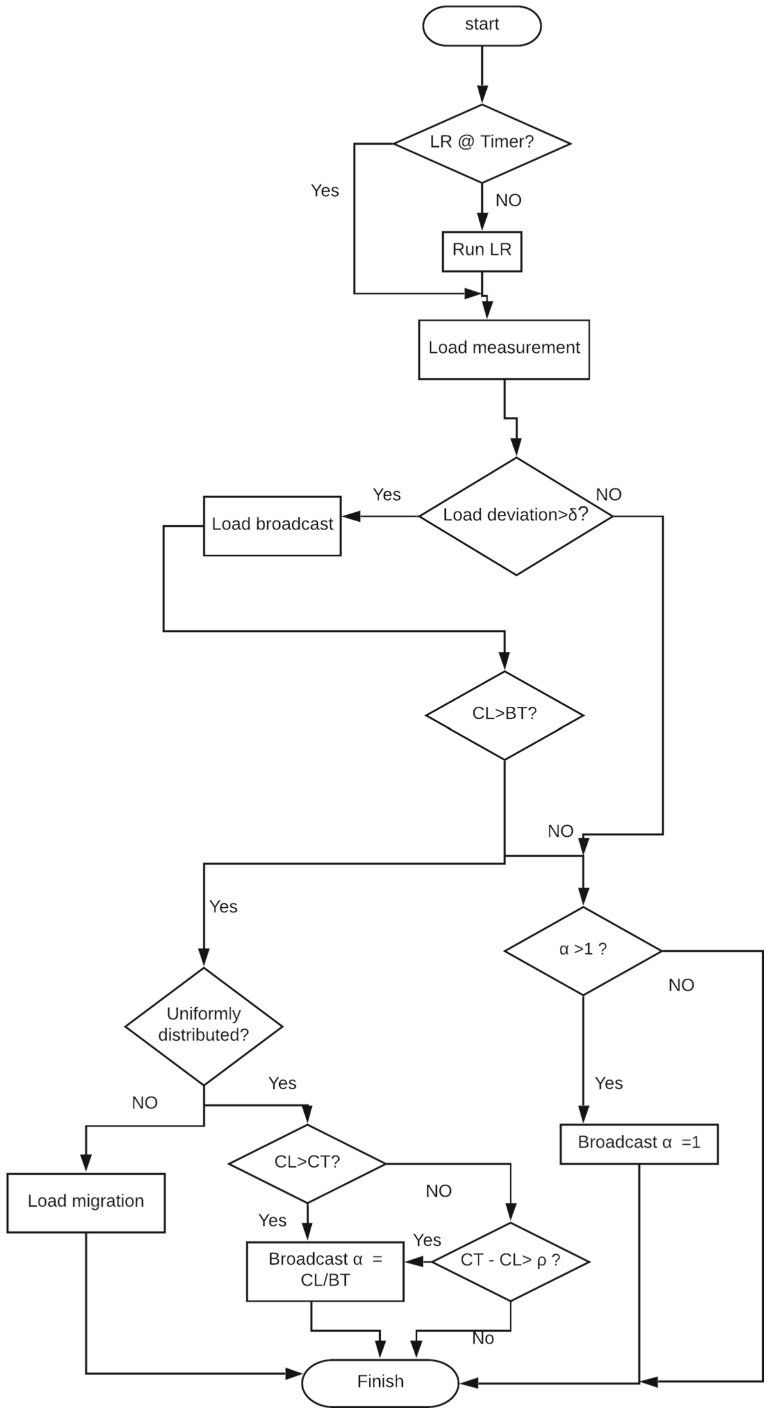


Fig. 2 The operational flow of the proposed load balancing scheme

controllers. The controllers possess high load to execute the subsequent tasks of the load balancing process. Now, there arise two possible cases. In the first case, the load in the network is uniformly distributed across all the controllers. This occurs when almost all the controllers in the network are heavily loaded. In the second case, the load in the network is non-uniformly distributed. This essentially indicates that there are some lightly loaded controllers in the network, which may be the potential target for the load balancing process. Here, we consider a parameter β_i for controller i which is representative of high load and is calculated as CL_i/BT_i. The controllers having high load execute the following tasks under these cases.

Case-1 In our approach, we dynamically increase the value of α and hence increase the current threshold of a controller in high load condition. In case the load is uniformly distributed, our algorithm calculates the difference between CL and CT. If the difference is greater than ρ, then it sets new α as CL/BT. This is to accommodate more load in the controller. On the other hand, if the difference is less than ρ, it is not possible to perform load balancing. It is trivial that we perform load balancing up to a certain level, where the distribution is uniform. This is because a higher ρ or higher threshold for all the controllers may lead to an unstable condition. Thus we choose ρ to 20% of BT. This is derived experimentally considering the hardware capacity of the controller and effective adjustment of CT. On the other hand, we reduce the value of α under the lightly loaded condition and set it as CL/BT. Accordingly, other controllers modify their α value. Again, our algorithm checks the value of α when CL is less than BT and subsequently set it to 1. Then the controller broadcasts this α to all other controllers. As our scheme, increases the current threshold to accommodate more load and decreases it when the load is low on the network, it clearly explains the self-adaptiveness nature of the scheme.

Case-2 When the highly loaded controllers determine the load is non-uniformly distributed in the network, they procedurally select suitable controller-switch pairs and subsequently trigger the migration of the load from source to target controller. Our proposed load balancing process is described in Algorithm 1. The controller-switch pair selection is one of the key tasks in the load balancing process, which is presented in Algorithm 2. It iteratively compares the current load of all lightly loaded controllers and objectively returns a controller-switch pair that possesses minimum load and shortest route between the switch and target controller. The controller-switch pair selection process maintains a list of lightly loaded controllers. The selection process depends on different factors as listed below:

(a) The load of the migrated switch should not exceed CT of the target controller. This is to avoid consecutive load migration at the target controller.
(b) The distance between the selected switch for migration and the target controller should be minimized.

To achieve the requirement in (a) we have introduced one constraint in our proposed switch-controller selection process which are presented as follows:

$$(CL_{i'} + \delta + SL_k)/CT_{i'} < \beta_i \tag{1}$$

where $CL_{i'}$ is target controller's load, SL_k is load of selected switch and $CT_{i'}$ is target controller's current threshold. Here Eq. 1 ensures that new β of target controller is less than the present β of source controller, which may lead to another load migration and introduces instability in the network. The load informed by target controllers may have deviation maximum up to δ due to delay; hence, this has been taken into consideration while the source controller calculates the new load of the target controller. There may exist many switch-controller pairs for migration, but we need to select those pairs which satisfy the constraint mentioned in (b). We record each mapping representing the difference of hop count from the switch to the target controller and hop count from the switch to the current controller. The difference in respective recorded hop counts is captured, sorted in increasing order. A smaller or negative value of hop count difference means the switch is closer to the target controller than the current controller. If there exists more than one mapping having the same hop count difference, we choose the mapping with maximum switch load. Briefly, the switch migration from the source controller to target does not initiate another load migration at the target controller, and distance of the selected switch should be geographically nearer to the target ones.

After the selection of a suitable controller switch pair, it finds the energy-efficient routing algorithm and the link path through which the migration can happen. Using the selected energy-efficient path, switches load are migrated to the target controller, which reduces the overall network load.

Proposed SALB scheme is applicable to multiple controller switch migration at the same time. The load migration for more than one controller is possible considering the constraints mentioned above. The controllers with high load can find their suitable controller-switch pair for load migration at the same time and can migrate the overloaded switches control to the target ones. These migrations do not affect the bandwidth utilization and increment of network traffic. The Algorithm 2 also ensures that no more than one highly loaded controller chooses the same controller as target, because the selected switch for every highly loaded controller is not in the same hop count difference to the target one.

4.1.4 Load_migration

After suitable controller-switch pair selection, the controller with high load sends a $Role\_Request$ message to the target controller. The target controller replies with $Role\_Reply$ message after getting a confirmation from the selected switch and then switch migration procedure starts. On the other hand, the controller with high load first completes the unfinished work of the selected switch (if any) and then migrate the corresponding switch's load to the target controller with $End\_Migration$ command. The target controller updates its role and informs the migrated switch about its role exchange. Finally, both the controllers update their switch connection table. The load migration procedure does not create any impact in the data plane, as only switch's control is changed from one controller to another but its load remains same as previous.

Algorithm 1 *LoadBalancing* Algorithm

```
  Input: $CL, BT, CT, \rho, \alpha$
  Output: $\alpha$
for i= 1 to n do
  if $CL_i > BT$ then
    Procedure Send()
    if Load Uniformly Distributed then
      if $((CL_i - CT_i)$ or $(CT_i - CL_i) > \rho)$ then
        $\alpha_i = CL_i/BT_i$
        BroadcastNewAlpha($\alpha_i$)
      end if
    else
      GetControllerSwitchPair();
      EERAS();
      LoadMigration(controller,switch,link);
    end if
    if $\alpha_i > 1$ then
      $\alpha_i = 1$
      BroadcastNewAlpha($\alpha_i$)
    end if
    End Procedure
  else
    Procedure Receive()
    receive ($\alpha_i$)
    $\alpha_{max}$ = Findmax($\alpha_i$)
    CT= $\alpha_{max} \times BT$
    End Procedure
  end if
end for
```

4.1.5 Link_reset

This module helps in resetting the shuffled link to their initial condition (i.e, before running the load migration algorithm). After long run of load balancing scheme some switches are controlled by controllers that are not geographically close to them, even if the network has less load. This leads to performance reduction of SDN network. In order to overcome this challenge our *Link_reset* module resets the migrated switch's link to their initial condition. This module runs once in a day at a certain instance of time. Here, for simulation purpose we set this module's running time as 12 AM each day. This module runs only if finds the network is lightly loaded.

In the next section we describe our proposed energy-optimization framework with detail functionality of its modules.

4.2 Energy-optimization framework

Our proposed energy-optimization framework consists of three functional modules namely *Cal_Controller*, *Cal_Switch*, *Cal_LinkPath*. In addition, we proposed energy-efficient Routing Selection (EERAS) algorithm for efficient route selection that minimizes energy consumption by SDN devices and links. The design of these three modules functions include the concept of *Active* and *Sleep*. In this subsection, we formally describe our energy optimized framework. We calculate the total energy consumption of a network as follows:

$$E = \sum_{i=1}^{n} (e_{1i} * E_{swi} + e_{2i} * E_{ci} + e_{3i} * L) + E_{link} \tag{2}$$

Here, E_{swi} denotes energy consumption of ith switch along the selected path and E_{ci} represents the energy consumption of the ith controller. The latency of the network is denoted by the parameter L that depends on λ, T, Q_{size}. The parameter E_{link} represents the total energy consumption of all the active links in the network. We used three normalized weight parameters e_1, e_2 and e_3 which are derived through simulation run for a given time period. We consider all the active switches and controllers present in the network for calculation of the parameters E_{sw} and E_c. We model the latency, L as follows:

$$L = f(Q_{size}, \lambda, T) = \frac{\lambda * T}{Q_{size}} \tag{3}$$

Here, Q_{size} is the queue size associated to each device (switch, controller) which is a derived constant. The parameter L is calculated experimentally through simulation by observing arrival of packets in the network. Here, λ denotes the arrival rate and T is the time period. We consider latency in the energy consumption calculation, because due to introduction of $Sleep$ mode concept, the latency may increase and which degrades network's performance. The energy consumption model of individual network devices are presented as follows:

4.2.1 Switch energy consumption model (*Cal_Switch*)

This module calculates the energy consumption of all the switches present in the network. When any switch is in sleep mode, then it can awake itself or it can be awaken by its neighbor switches. Firstly, the switch checks its queue size periodically, and if finds sufficient packets for processing, then it awakens itself. On the other hand, it checks the next hop of the packet and awaken its neighboring switches if requires. It is modelled as follows:

$$E_{sw} = \alpha * E_{sw}^{A} + (1 - \alpha') * E_{sw}^{S} \tag{4}$$

E_{sw}^{A} is the energy consumption when a switch awakes itself from $Sleep$ to $Active$ mode and E_{sw}^{S} is the energy consumption when the switch is activated by its neighbor node. Here, α is set to one for active switch and is set to zero when the switch is in sleep mode. E_{sw}^{A} includes conversion energy from sleep to active mode (E_{SA}), sensing energy of the network (E_S) when switch senses incoming packets in its queue and energy consumed by active ports of the switch (E_{pi}). It is modelled as follows:

$$E_{sw}^{A} = E_{SA} + \sum_{i=1}^{n} E_{pi} + f(T) * E_s \tag{5}$$

The function $f(T)$ is initialized to one ($f(T)$=1), when the switch receives packet for processing and is initialized to zero ($f(T)$=0), when the switch does not have enough packets. Subsequently, we computed the function $f(T_i)$ as: $f(T_i)$= $f(T_{i-1}+\delta)$, where δ is a random increment in time interval. The expected value of δ is calculated as $E(\delta)$= $\delta * p_r(t = \delta)$, where $p_r(t = \delta)$ is the probability of increment in time interval equals to δ. This completes the computation of function $f(t)$. The parameter E^S_{sw} includes E_{SA}, E_{pi} and E_{beacon}. Here, E_{beacon} is the energy consumed by beacon message that is at most 1–3% of total battery energy. So, calculation of E^S_{sw} is performed as follows:

$$E^S_{sw} = E_{SA} + \sum_{i=1}^{n} E_{pi} + E_{beacon} \tag{6}$$

In our model, When a switch awakes itself it checks destination id of all incoming packets in its flow table. The switch broadcasts a beacon message to all the neighbor switches if any incoming packet requires activation of those neighbors. Then the neighbor switches are activated. Our proposed switch energy consumption model has been tested experimentally and found to be optimized.

Algorithm 2 GetControllerSwitchPair()

```
  Input: $CL, SL, CT, \delta, \beta$, LLC, HLC
  Output: $(S_k, C_{i'})$
if LLC.size == 0 then
    return -1;
end if
for $k \in$ U do
    for $i' \in LLC$ and $i \in HLC$ do
        $\beta_{i'} = (CL_{i'} + \delta + SL_k)/CT_{i'}$
        if $\beta_{i'} < \beta_i$ && $\beta_{i'} < 1$ then
            hcd = hop_count($S_k, C_{i'}$) - hop_count($S_k, C_i$)
            mapping.push($hcd, S_k, C_{i'}$)
        end if
    end for
end for
if $mapping.size == 0$ then
    return -1;
end if
sort(mapping)
pop(mapping)
return $(S_k, C_{i'})$
```

4.2.2 Controller energy consumption model (*Cal_Controller*)

This module calculates the energy consumption of controllers present in the network and its operation is similar to the Cal_Switch module. It is modeled as follows:

$$E_c = \beta * E^A_c + (1 - \beta^{'}) * E^S_C \tag{7}$$

where E_c^A is the energy consumption when a controller awakes itself, and E_c^S is the energy consumption when the controller is activated by its neighbor. Here we set β to 1 when the controller is active and is set to 0 when the controller is in sleep mode. Here, E_c^A depends upon CPU utilization of different packets ($CPUutil_i$), sensing energy (E_S), and conversion energy (E_{SA}). So, the parameter E_c^S is modeled as follows:

$$E_C^A = E_{SA} + \sum_{i=1}^{n} W_i * CPUutil_i + f(T) * E_s \tag{8}$$

Here, W is the weight parameter that varies for different packets having different CPU utilization. The function $f(T)$ is calculated in the same way as discussed before. The parameter E_c^N considers E_{SA}, $CPUutil_i$ and beacon message energy (E_{beacon}) and is modeled as:

$$E_C^N = E_{SA} + \sum_{i=1}^{n} W_i * CPUutil_i + E_{beacon} \tag{9}$$

The energy consumption of controller plays an important role in optimizing the total energy consumption of the network.

4.2.3 Active link energy consumption (*Cal_linkpath*)

This module finds the energy consumption by each active link present in the network with an objective of minimizing the link path energy. It considers the total traffic (x_i) along the ith link and the capacity (c_i) of that link. In addition, it incorporates the energy consumption of the respective source and destination switches associated to the link. Therefore, the total energy consumption (E_{link}) of all the active links in the network is modeled as follows:

$$E_{link} = min\left(\sum_{i=1}^{n} E_i\right) + E_{sw}^{src} + E_{sw}^{dst} \qquad \left[E_i = E\left(\frac{x_i}{c_i}\right)\right] \tag{10}$$

Here, E_{sw}^{src} and E_{sw}^{dst} respectively denotes the source and destination devices energy with respect to ith link. We ignore the energy consumption of a link with very less traffic load and assume it as idle. An idle link is to be activated if high traffic load arrives to it. In such case, the energy consumption of the respective link is calculated as $E_i = E_{SA} + E\left(\frac{x_i}{c_i}\right)$. This module functions for all the routes between different controllers as well as the routes between SDN switches.
Next, we present our proposed (*EERAS*) algorithm, which uses the above mentioned energy computation modules.

4.2.4 Energy-efficient routing algorithm selection

Energy-efficient Routing Algorithm Selection, $EERAS$ is a heuristic energy-optimization algorithm that selects efficient route with an objective of minimizing energy consumption by controllers, switches and link paths.

The $EERAS$ algorithm takes two stacks of state-of-art routing algorithms as input and provides the energy-efficient efficient routing algorithm and the energy consumed by the respective controllers, the link paths as the output. The $stack1$ consists of 3 Hierarchical routing protocols namely Leach, Teen, Apteen where as, the $stack2$ consists of hybrid protocols such as EIGRP, CEDAR, ZRP. The hierarchical routing protocols reduce the size of routing tables providing better scalability. Similarly, hybrid protocols return the most efficient route consuming minimum energy. Our $EERAS$ algorithm first computes the energy of controller E_c and active links E_{link} in the network, under each routing protocol in $stack1$. These data is stored in a matrix called Eng_mat. Then for each protocol, our algorithm checks if the controller energy consumption E_c is below a predefined threshold Th. After wards, it returns the protocol having the minimum link energy E_{link}. In addition, it outputs the values of E_c and E_{link} in the network under that protocol. This is done by traversing the data stored in the matrix Eng_mat. On the other hand, if the controller energy E_c is more than the predefined threshold Th value under all the protocol in *stack1* our algorithm objectively selects an efficient hybrid protocol from $stack2$. The selection process of hybrid protocol from *stack2* is similar to the selection process used on *stack1*. Finally, our algorithm returns the selected protocol that provides optimized energy consumption and the associated values of E_c, E_{link}. The details of EERAS algorithm is presented in Algorithm 3.

Algorithm 3 EERAS Algorithm

```
1: while !empty($stack1$) do
2:     $protocol_i \leftarrow pop(stack1)$;
3:     for i=1 to n do
4:         $Eng\_mat \leftarrow compute\_energy(protocol_i)$;
5:     end for
6:     if $E_c^i \leq Th$ then
7:         $min\_cost \leftarrow E_{link}^i$;
8:         for all i=2 to n do
9:             if $E_{link}^i < min\_cost$ then
10:                $min\_cost \leftarrow E_{link}^i$;
11:            end if
12:        end for
13:    end if
14: end while
15: if $min\_cost \neq 0$ then
16:     $P= protocol_i$;
17: else
18:     Process $Stack2$
19:     Repeat step-6 to step-10;
20: end if
21: Return $P$, $E_c^i$ and $E_{link}^i$;
```

Our $EERAS$ algorithm invokes $compute\_energy$ function for computing energy consumption of controller and links for efficient selection of routing algorithms. This procedure is presented in Algorithm 4. It determines the energy computation of each protocol from both the stacks. In first step, a controller sends a broadcast message to all its neighbor $neigh(c)$ and subsequently the neighbors update their link states. The link state of a device is represented as matrix, $LS(node, l_i)$, where node denotes the neighbour node and l_i is the respective link cost. The update link state ($Update\_LS$) procedure takes the link weight and connectivity matrix as input and finds the minimum cost route. Finally the neighboring controllers send the updated link state LS to the initiating controller. Then the controller computes its energy consumption E_c as aggregation of the energy consumption of $Broadcast$ message and link state LS update procedure. After wards, the initiating controller updates its link state and finds the optimal route $R < c, n_1, n_2, \ldots n_k, destination >$ to the destination. Then, the controller computes the link energy E_{link} as aggregation of energy consumption of all the links in the route R. This completes the energy computation for a chosen routing protocol. Our EERAS algorithm uses this result to select an effective routing protocol (P) that involves minimum energy consumption for controller (E_c) and link (E_{link}).

Algorithm 4 compute_energy function

```
1: Start a timer t;
2: Controller c broadcasts beacon message to neighbors: $Broadcast(beacon, neigh(c))$;
3: for all neigh(c) do
4:     Update link state matrix: $Update\_LS$;
5: end for
6: Controller C receives updated link states,$LS$;
7: Find the optimal route R <c,$n_1$,$n_2$,...$n_k$,destination> using $LS$
8: End timer t;
9: Calculate $E_c$: $E_c = E_{Broadcast} + E_{Update\_LS}$;
10: for all $l_i \in R$ do
11:     Calculate $E_i$;
12: end for
13: compute $E_{link}$;
14: Return $E_c$ and $E_{link}$;
```

5 Experimental results and evaluation

We evaluate our controller system model in three stages. Firstly, we analyze the performance of our model considering different load conditions. Secondly, we evaluate the characteristics of our controller system model towards its usability in real life network applications. Finally, we verify the accuracy of our model.

5.1 Methodology

For simulation experiment setup, we take Floodlight [32] as the SDN controller. We simulate the real Internet service topology, BT Asia Pacific (20 nodes) from the Internet Topology Zoo [33] to make this implementation more similar to the real life scenario. Out of those 20 nodes, we choose five as controllers (Floodlight controller) and rest nodes are OpenFlow switches. The switches are assigned to the five controllers for their flow management and inject incoming packets to their corresponding controllers.

Table 3 Performance metrics values using proposed controller system model

	Throughput	Packet drops	Load balancing time	Control overhead
Controller system model	5768 # packets/s	5 # packets/s	3 s	3% of # M

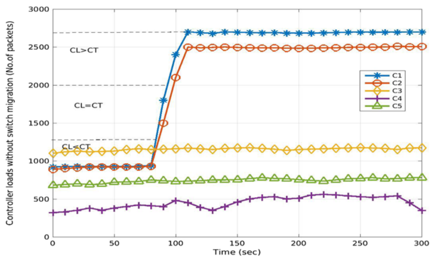
More packets are generated by the end hosts using the mininet platform to create a high load on the controllers and to measure the efficiency of our proposed load balancing scheme SALB. For the purpose of our analysis, we create different test cases and calculate the energy consumption in the network using our $EERAS$ algorithm. Our test cases cover following three traffic load conditions in the network: 1) high load ($CL > CT$), 2) low load ($CL < CT$) and 3) load= threshold ($CL = CT$). We also measure the network performance in terms of throughput, load balancing time, packet drops and control overhead.

5.2 Performance evaluation:

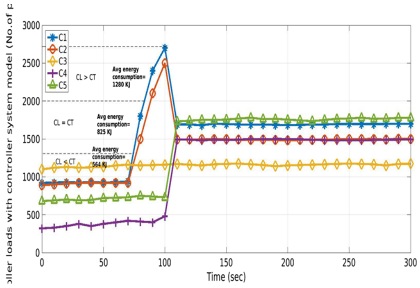
We test our integrate controller system model with real-world traffic applications like Telnet, VoIP- G.711, G.723, G.729, voice activity detection (tested on voice samples available on SndLib library), compressed RTP-DNS, and network games. We vary the queue size (Q_{size}) between 500 and 2000 over all these applications. Then, we measure the network performance in terms of throughput, packet drops, load balancing time, control overhead. Here, the control overhead is measured as the number of the messages exchanged during the load balancing process. Table 3 shows the performance results of our controller system model.

We observe the performance results as an average of over fifty simulation runs with heterogeneous traffic flows. The result shows that the control overhead is 3% of the total number of the message exchanged (M), which is reasonable for running real-life network applications. Figure 3a and b show the load distribution in two scenarios considering the three load conditions respectively. In Fig.3a; without any load balancing schemes, loads of highly loaded controllers keep on increasing. However, in Fig. 3b; with CSM the switches of highly loaded controllers C_1 and C_2 are migrated to lightly loaded controllers C_4 and C_5 respectively before 110 s and the load of C_1 and C_2 are stabilized before 115 s. It also shows the energy consumption by our proposed CSM with time in an average. The result presents that the load balancing process consumes more energy under heavy load condition before the network stabilizes.

In the rest of this section, we mainly compare our controller system model with state-of-art load balancing schemes; DALB [16], LI [12], SMDM [18], SMCLBRT [19], and energy-efficient frameworks; HEATE [26], heuristic algorithm [13]. We simulate the same test environment (BT Asia Pacific) for all the schemes and frameworks and run each for 5 min. We observe the load distribution and energy consumption changes for the five controllers continuously by increasing the packet arrival rates. Figure 4a shows the load distribution of 5 controllers for the five load balancing schemes. For our controller system model, the load balancing time starts at 102 s and ends at 105 s. However, the start time for DALB, LI, SMDM, SMCLBRT are 111 s, 107 s, 107

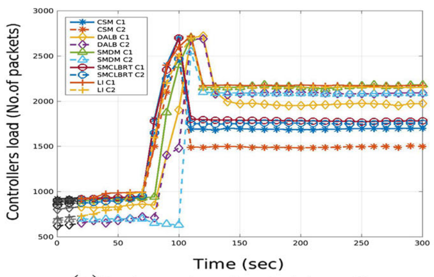


(a) Load distribution without any load balancing scheme

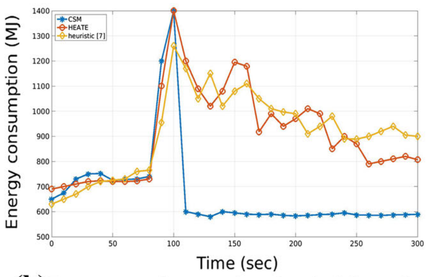


(b) Load distribution and energy consumption with CSM

Fig. 3 Load distribution and energy consumption of five controllers



(a) Load comparison of overloaded controllers



(b) Energy consumption comparison of overloaded controllers

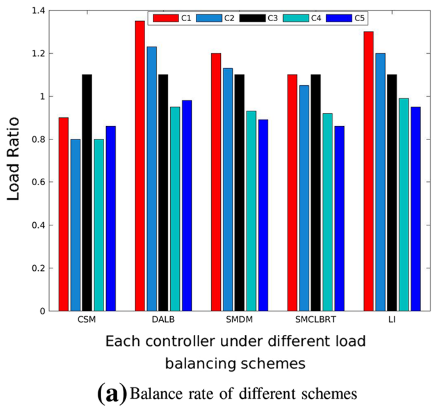
Fig. 4 Load distribution and energy consumption under different schemes and frameworks

s, 105 s and the end time are 122 s, 116 s, 115 s, 110 s respectively. Figure 4b shows the energy consumption comparison of CSM, HEATE, heuristic algorithm. We observe that our controller system model reduces the total energy consumption, which is approximately 587 MJ for large scale network lies below 700 MJ (Threshold energy consumption value for real-life SDN applications) [34].

Our model runs the energy-optimized framework and EERAS algorithm for the highly loaded controllers. EERAS algorithm selects the shortest distance with minimum energy consumption for the switch migration. After that, the load balancing module again finds the minimum hop count distance of the selected switch from source and target controller for optimal route selection. Our module consumes less energy than other frameworks for sleep and active mode of devices and optimal routing algorithm selection. As multiple switches from multiple controllers are migrated simultaneously and dynamic threshold adjustment for different load conditions, CSM takes less time as compared to other schemes.

We also compare the load distribution ratio of different load balancing schemes as shown in Fig. 5a and energy consumption ratio of various frameworks shown in Fig. 5b for efficient performance analysis. the load distribution ratio is calculated as follows:

$$Load\_ratio = \frac{Load_{C_i}}{Load} \tag{11}$$



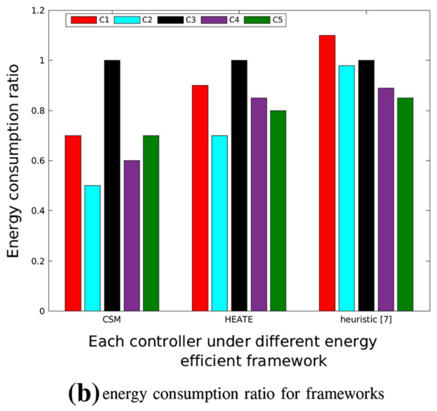


Fig. 5 Load distribution and energy consumption ratio of five controllers

where $Load_{C_i}$ is the load of controller C_i and $Load$ presents the average load of 5 controllers. Similarly energy consumption ratio is calculated as:

$$energy\_ratio = \frac{energy_{C_i}}{energy} \tag{12}$$

where $energy_{C_i}$ is the energy consumption of controller C_i and $energy$ presents the average energy consumption of 5 controllers.

The time complexity of our proposed EERAS heuristic algorithm is $O(n)$, where n is the number of nodes present in the network. The energy consumption associated with EERAS algorithm is negligible as it is invoked occasionally at the time of load balancing requirements. On the other hand, the controller-switch pair selection process involves time complexity $O(n)$. This justifies the total execution time of our controller system model is linear on network size.

5.3 Characteristics analysis

We analyze the impact of our load balancing scheme on overall energy consumption in the network. Here, we consider the energy requirement of OpenFlow switches, controllers, and link paths. It is trivial that the energy consumption in a network is proportional to the overall load on the network, especially for running load balancing process. Our approach requires migration of control of switches from source to target controller avoiding the complexity of data plane. These characteristics play major roles in reducing energy consumption with our controller system model. In addition, we have calculated the amount of CO_2 emission as a function of energy consumption in the network. It is calculated as follows:

$$CE = \sum_{i=1}^{n} EC_i * CEE_i \tag{13}$$

Table 4 Output (partial) of WireShark

Output without controller system model
3C-95-09-14-A1-95 (hp) consumes 722 MJ
68-b5-99-fc-d1-df, 00-01-2a-43-ab-c2 consume 600, 598 MJ respectively
Output with controller system model
3C-95-09-14-A1-95 (hp) consumes 589 MJ
68-b5-99-fc-d1-df, 00-01-2a-43-ab-c2 consume 561, 476 MJ respectively

Here, CE is the amount of CO_2 emission, EC is the energy consumption, CEE is the CO_2 emission per unit energy [23]. We have observed the reduction in carbon dioxide emission to the environment using our model, which ensures it can be effectively deployed in real life network implementations.

5.4 Accuracy verification

We run our proposed controller system model in EstiNet-emulation platform with BT Asia Pacific network to verify the accuracy. Here we introduce 15 number of MT198T switches to forward the heterogeneous traffic generated by D-ITG over a network of 200 hosts and five controllers. The hardware set up was built inside different laboratories in our institute with 500 links to connect the network. The link capacity varies between 500 Mbps-1 Gbps according to the network requirement. We install WireShark in each network device to find out energy consumption by them. Then, we calculate the average energy consumption in the network under our proposed controller system model. The result is presented in Table 4. We noticed that our controller system model successfully balances the load in the network with reasonably less time. We observe that the network consumes approximately 590 MJ under our controller system model. This shows the accuracy of our controller system model in the SDN network testbed and thereby can be effectively deployed in practical network applications. Besides, our controller system model improves different performance metric that, in turn, provides better QoS and scalability.

Our proposed load balancing approach objectively reduces the no. of broadcast messages between the controllers, which essentially enhances the throughput. In addition, our approach converges in all possible load conditions in the controllers, that is one of the unique features of our approach. While evaluating the state-of-art load balancing approach, we found that energy consumption during load balancing process by the SDN devices is significantly high, which is one of the major limitations. Therefore, we introduce a heuristic-based EERAS algorithm in integration with load balancing as our proposed controller system model. This justifies the efficacy and usability of our proposed model.

6 Conclusion

Here, we present a controller system model for SDN controllers consisting of a new load balancing technique SALB and an energy-optimization framework. The interaction between these two components effectively allows the controller to balance the network traffic load with the reduction in energy consumption. We observed that this reduction in energy consumption is approximately 25% in large scale network with heterogeneous applications. Besides, our proposed controller system model enhances the performance of the network in terms of throughput, packet drops, and load balancing time. This work can be extended to design a new efficient routing protocol for SDN controller that inherently use energy consumption as one of the metrics for routing decision.

References

1. Priyadarsini M, Bera P (2018) A new approach for SDN performance enhancement. In: The 25th international conference on computer networks (CN), Gliwice, Poland, pp 115–129, CCIS 860
2. Gravelle E, Martínez S (2017) An anytime distributed load-balancing algorithm satisfying capacity and quantization constraints. IEEE Trans Control Netw Syst 4(2):279–287
3. Li L, Xu Q (2017) Load balancing researches in SDN: a survey. In: The 7th IEEE international conference on electronics information and emergency communication (ICEIEC)
4. Rao S (2016) A guide for running multiple controllers in software defined networks. An Article on "TheNewStack"
5. Salman O, Elhajj IH, Kayssi A, Chehab A (2016) SDN controllers: a comparative study. In: Proceedings of the 18th mediterranean electrotechnical conference MELECON, Limassol, Cyprus, pp 18–20
6. Golinelli ES (2015) A software defined networking evaluation approach to distributing load. Master's Thesis submitted to the Department of Informatics, University of Oslo
7. Zhong H, et al (2015) An efficient SDN load balancing scheme based on variance analysis for massive mobile users. Hindawi Publishing Corporation Mobile Information Systems, Article ID 241732
8. Phan TK (2017) Design and management of networks with low power consumption. Ph.D. Thesis submitted in University of Sophia Antipolis
9. Gao X (2016) Traffic load balancing schemes for devolved controllers in mega data centers. IEEE Trans Parallel Distrib Syst 23(5):572–585
10. Tkachova O et al (2016) A load balancing algorithm for SDN. Scholars J Eng Technol (SJET) 9:25–36
11. Chen-xiao C (2016) Research on load balance method in SDN. Int J Grid Distrib Comput 9(1):25–36
12. Yu et al (2016) A load balancing mechanism for multiple SDN controllers based on load informing strategy. In: The 18th Asia pacific network operations, and management symposium (APNOMS)
13. Cruz AF (2017) Optimization of power consumption in SDN networks. In: The ninth international conference on emerging networks and systems intelligence
14. Priyadarsini M, et al (2018) A new approach for energy efficiency in software-defined network. In: The fifth international conference on software defined systems (SDS), Barcelona, Spain
15. Wu J, et al (2015) Goodput-aware load distribution for real-time traffic over multipath networks. IEEE Trans Parallel Distrib Syst 26(8):2286–2299
16. Zhou Y, et al (2014) A load balancing strategy for SDN controller based on distributed decision. In: IEEE 13th international conference on trust, security and privacy in computing and communications
17. Hu Y, et al (2012) Balanceflow: controller load balancing for openflow networks. In: IEEE CCIS
18. Wang CA et al (2017) A switch migration-based decision-making scheme for balancing the load in SDN. IEEE Access 5:4537–4544
19. Cui J et al (2018) A load-balancing mechanism for distributed SDN control plane using response time. IEEE Trans Netw Serv Manag 15(4):1197–1206
20. Zhou Y, et al (2017) Load balancing for multiple controllers in SDN based on switches group. In: The 19th Asia Pacific network operations and management symposium (APNOMS), pp 227–230
21. Huin N (2018) Energy-efficient software defined networks. Ph.D. Thesis submitted to HAL
22. Rodrigues BB, et al (2016) GreenSDN: bringing energy efficiency to an SDN emulation environment, paper in Research gate
23. Wu GH,et al (2010) Calculation and analysis of carbon emissions from energy consumption: case in Jinan City. In: IEEE 17th international conference on industrial engineering and management

24. Lange S et al (2015) Heuristic approaches to the controller placement problem in large-scale SDN networks. IEEE Trans Netw Service Manag 12(1):4–17
25. Wu Y et al (2016) Adaptive flow assignment and packet scheduling for delay-constrained traffic over heterogeneous wireless networks. IEEE Trans Veh Technol 62(10):8781–8787
26. Wei Y et al (2016) Energy-aware traffic engineering in hybrid SDN/IP backbone networks. J Commun Netw 5(2):559–566
27. Bolla R et al (2015) Fine-grained energy-efficient consolidation in SDN networks and devices. IEEE Trans Netw Serv Manag 12(2):132–145
28. Nam TM, et al (2015) Energy-aware routing based on power profile of devices in data center networks using SDN. In: 2015 12th international conference on electrical engineering/electronics, computer, telecommunications and information technology
29. Celdran AH et al (2016) Policy-based management for green mobile networks through software-defined networking. Springer, New York
30. Study Paper by Telecommunication Engineering Department, Government of India (2015) Software Defined Networking (SDN) as a tool for energy efficiency approaches in Information and communication technology (ICT) networks
31. Cisco works on LAN management solutions. https://www.cisco.com/c/en/us/td/docs/mnt-setup.pdf
32. FloodLight, Open SDN Controller. http://www.projectfloodlight.org/blog/2016/03/10/announcing-floodlight-v1-2/
33. Knight S et al (2011) The internet topology zoo. IEEE J Sel Areas Commun 29(9):1765–1775
34. Chai R et al (2017) Energy consumption optimization-based joint route selection and flow allocation algorithm for software-defined networking. J Inf Sci 60:040306. https://doi.org/10.1007/s11432-017-9043-8

Affiliations

Madhukrishna Priyadarsini[1] · **Shailesh Kumar**[1] · **Padmalochan Bera**[1] · **Mohammad Ashiqur Rahman**[2]

Shailesh Kumar
sk38@iitbbs.ac.in

Padmalochan Bera
plb@iitbbs.ac.in

Mohammad Ashiqur Rahman
marahman@fiu.edu

[1] Indian Institute of Technology Bhubaneswar, Bhubaneswar, India

[2] Florida International University, Florida, USA