Date of publication DEC-31, 2023, date of current version MAR-05, 2023.
www.computingonline.net / computing@computingonline.net

Print ISSN 1727-6209
Online ISSN 2312-5381
DOI 10.47839/ijc.22.4.3361

Coalitional Game Strategy and TSSM for Efficient Load Balancing in Software Defined Networking

G. NISHANTHI, R. DEEPA, S. GAYATHRI, B. JAISON
RMK Engineering College, Chennai 601206, India

Corresponding author: B. Jaison (e-mail: bjn.cse@rmkec.ac.in).

ABSTRACT Using distributed SDN control, software defined networking (SDN) delivers additional flexibility to network management, and it has been a significant breakthrough in network innovation. Switch migration is often used for distributed controller workload balancing. The time-sharing switch migration (TSSM) scheme proposed a strategy in which multiple controllers are allowed to share the workload of a switch via time sharing during an overloaded condition, resulting in reduced ping-pong controller difficulty, fewer overload occurrences, and improved controller efficiency. However, it requires more than one controller to accomplish, it has greater migration costs and higher controller resource usage during the TSSM operating time. As a result, we presented a coalitional game strategy that optimizes controller selection throughout the TSSM phase depending on flow characteristics. The new TSSM method reduces migration costs and controller resource usage while still providing TSSM benefits. For the sake of practicality, the proposed strategy is implemented using an open network operating system. The experimental findings reveal that, as compared to the typical TSSM system, the proposed technique reduces migration costs and controller resource usage by approximately 18%.

KEYWORDS Switch migration; load balancing; coalitional game strategy; time sharing switch migration; software defined networking.

I. INTRODUCTION

THE fast proliferation of cloud computing, big data applications, the internet of multimedia things, and increased data traffic have significantly raised network management difficulties. The traditional network architecture system consists of a data plane and a control plane in each switch, with the former handling packet processing and the latter handling decision making and administration. As a result, upgrading the current algorithms and policies to the switches is quite hard and time consuming because all the related switches in the given network must be updated one by one by system administrators or workers [1].

Today, the software defined networking method creates a distinct perspective of network administration in networking applications by shifting the control plane in switches to a central device known as the controller. As a result, the controller may handle many switches in the network. Monitoring and control of network switches are simplified in this current method as compared to traditional network management techniques, because the controller unit can offer such information about the switches.

Furthermore, by creating a set of rules in the controller, the newest algorithms and control policies may be quickly updated to the switches [2]. Aside from that, SDN may support a broad range of applications, such as (i) defending against cyber-attacks, (ii) recognizing malicious access points, and (iii) offering anonymous authentication, among others [3-7].

A single controller in a large network is a difficult option because it creates a bottleneck in network management performance. As a result, distributed SDN control (DSC) is demanded in network applications, and it acts as a promising solution in large network management with many switches [8]. The DSC enables many controllers to communicate with one another to administer the whole network. Where each controller manages a subset of switches (i.e., a subnet), and processes may be transferred among controllers to facilitate cooperation. Each controller is responsible for dividing the workload for the subnets and reassigning the burden of its switches through the periodical check-up of each subnet, which is known as controller placement [9]. The controller placement is mostly focused on load balancing and is carried out using a variety of techniques such as work group control technique [10], deep reinforcement learning technique [11], and so on. The upshot of such control strategies may significantly alter the

switches in the subnet, causing the subnet to become unstable via ping-pong operation. Furthermore, controller placement strategies are not thought to be successful for short-term flows such as distributed denial of service and impulses [12].

Switch migration allows for a smoother change of subnets in a shorter amount of time and addresses the concerns. A switch migration approach examines the workload state of each controller in the network in each time frame (or time interval or period) to determine if they are overloaded (busy) or lightly occupied (available to share other works). If a network is overloaded, the migration technique relocates a switch from the heavily loaded controller subnet to the lightly loaded controller subnet. Most existing switch migration methods follow the smallest slice of the migration, which is one single switch transferred at the start of the period. Once migrated, the switch remains in the most recent sub-net until the switch is picked for the following period. Most crucially, these migration methods always need a controller to supervise a single switch for the duration of the migration. As a result, the controller in these systems encounters ping-pong problems under elephant flow situations (i.e., flow conveys numerous packets) and faces the major problem of subnet instability [13].

II. LITERATURE REVIEW

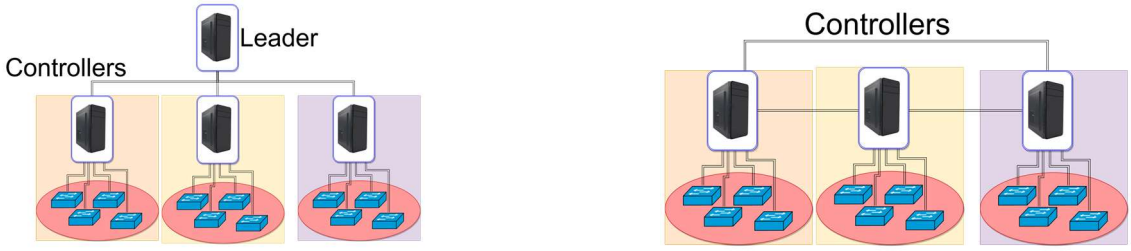
Several studies have been conducted throughout the years to highlight the numerous difficulties in the DSC network. Traditionally, dynamic controller placement methods are used to achieve controller load balancing. Chan et al. [14] presented a strategy for minimizing service interruption time by easily moving the process from one controller to another. In [15], it was described how a lightly loaded controller could operate as a leader in the event of a breakdown of the standard leader controller unit. Controller placement approaches and issues were discussed in [9], which emphasizes the need of controllers maintaining fairness while sharing their tasks. When compared to previous controller placement methods, [16] provided a dependable deployment strategy with the goal of minimizing packet loss and improving network stability. Kim et al [17] developed a strategy for improving the output of a distributed datastore in an Open Daylight controller cluster by regularly distributing shared leaders to cluster members. In [18], a system was described in which controllers collaborate to redirect traffic to prevent congestions during busy or overloaded periods on switches. In [19], it was proposed a software defined cyber seeking system with a hybrid controller for cloudlets and local networks. Work [20] presented prediction-based controllers, which forecast network demand and conduct device transfers based on prediction. The controller placement research, such as the work group control approach and the deep reinforcement learning technique were provided in [10, 11], where these strategies were ineffective during impulses and distributed denial of service, etc. Aside from the dynamic controller placement technique, approaches for DSC workload balancing are classified into three types: (i) switch migration, (ii) flow migration, and (iii) flow splitting.

Switch Migration: To reduce burden, switch control can be migrated from overloaded controllers to lightly laden controllers. In [21], switch migration in consideration of a controller's CPU and memory allocation exceeding its threshold level was discussed, but it did not define the method of selecting the targeted controllers. Work [22] discussed switch migration utilizing the Q-learning approach, which had lowered the standard deviation of the controller workload. Cui et al. [23] utilized the controller's reaction time to migrate switches. This strategy transfers the switch with the greatest load of the controller in the shortest amount of time. In [24], it was suggested a strategy for selecting targeted controllers for switch migration based on CPU use, memory capacity, and bandwidth, among other factors. Hu et al. [25] suggested a simulated annealing algorithm for selecting the targeted controller to reduce the cost of switch migration.

Flow Migration: Instead of migrating an entire switch, the flow migration approach merely transfers the hardness (i.e., flow beyond the threshold level) of the flow. Hu et al. presented an approach in which a super controller administers each controller in the system and controls the flow controlled by them [26]. Work [27] presented a game theory strategy for managing each controller flow through task exchange. When compared to standard flow migration methods, Maity et al. [28] offered a traffic aware consistent approach for minimizing flow migration duration and obtained a 15% reduction in flow migration time. Furthermore, using a traffic-aware flow migration technique, work [29] offered a method to lower data plane load and obtained a 13% reduction when compared to the two-phase update approach.

Flow Splitting: This approach enables a switch to be managed by many controllers at the same time. Gorkemli et al. [30] presented a solution for flow splitting utilizing virtual overlay on the data plane that switches must negotiate with their controllers. In [31], it was developed a convex quadratic programming-based solution for load balancing and decreasing new switch-controller appointments by modelling the mapping between controllers and switches.

However, due to synchronization and the complexity of the design, a switch cannot be operated by more than one controller at the same time. As a result, flow migration and flow splitting methods violate the OpenFlow protocol and cannot be used in the real-time controller platform.



(a) Hierarchical method (a) Flat method

Figure. 1. Control methods for the DSC architecture

A. PROBLEM DESCRIPTION AND CONTRIBUTION

As described in the literature section, most switch migration solutions struggle with the ping-pong challenge. The following example explains the ping-pong difficulty of the controller. Consider two controllers [C_X and C_Y] and three switches [S_L, S_M, S_N] in a network with a maximum manageable workload of 200 PIMs per second for each controller. Switches S_L, S_M, and S_N generate 120, 160, and 120 PIMS every period, accordingly. C_X manages switches S_L and S_M at time t, and controller C_Y manages switch S_N. Because $\Upsilon_{C_X} = \delta_{L(t)} + \delta_{M(t)}$= 120 + 160 > λ_{C_X} (200 PIMS), C_X is overloaded and requires switch migration. In most switch migration strategies, an overloaded controller will request and take over a switch from other controllers for an extended period. As a result, at time t+1, Switch S_L is moved to controller C_Y's subnet. However, if $\Upsilon_{C_Y} = \delta_{N(t)} + \delta_{L(t)}$ = 120 + 120 > λ_{C_Y} (200 PIMS) at period t+1, controller C_Y will be overloaded. As a result, controller C_Y requests that C_X take over a switch again at time t+2, increasing the complexity of ping-pong.

W.K. Lai et al. [32] recently suggested a time-sharing switch migration technique (TSSM) that mitigates controller ping-pong by spreading the burden of a switch that is monitored by two controllers at the same time during overloaded situations. It proposes a switch migration approach in which the burden of the switch is split across two controllers over a certain time period. Using the preceding example, at time t+1, C_X handles 40 PIMs of S_L, while C_Y manages the remaining 80 PIMs via migration. Both controllers C_X and C_Y are regulating the workload of switch S_L currently. As a result, C_X's workload is $\Upsilon_{C_X} = \delta_{L(t)} + \delta_{M(t)}$ = 40 + 160 = λ_{C_X} (200 PIMS), while C_Y's workload is $\Upsilon_{C_Y} = \delta_{N(t)} + \delta_{L(t)}$ = 120 + 80 = λ_{C_Y} (200 PIMS), indicating that none of the controllers is overloaded (busy) in period t+1. Similarly, at time t+2, C_Y processes 80 PIMs before sending the remaining 20 PIMs to the C_X controller subnet. The TSSM technique can effectively overcome the controller's ping-pong issue using this strategy.

It provides an approach in which two controllers, namely an overload controller (one) and a lightly loaded controller (one), are merged and the switch is relocated from an overloaded to a lightly loaded controller subnet at an appropriate moment in time. When compared to existing switch migration methods such as group-based dynamical controller placement [10], churn-triggered migration [30], and best-fit migration [32], the results of this technique show that it significantly reduces overload occurrences of controllers while effectively balancing the workload of all controllers with improved controller efficiency. Nonetheless, more than one lightly loaded controller operation in the TSSM yields greater controller efficacy than the original (i.e., stated in the research) despite the additional switch migration cost. Furthermore, because the migration switch is managed (i.e., controlled) by more than one controller in the network, this technique consumes additional controller resources during TSSM operation.

As a result, we suggest an approach that optimizes the lightly loaded controller selection during the TSSM period and enables for more than one lightly loaded controller to be used for switch migration during the TSSM period without increasing migration cost. The controller is chosen based on flow characteristics using a coalitional game strategy algorithm, which decreases controller resource consumption by lowering the number of controllers involved in flow processing. The new TSSM method reduces migration costs and controller resource usage while also providing TSSM advantages. For its feasibility, the proposed scheme is implemented using the open network operating system (ONOS), which can respond to approximately one million flow processing requests per second.

B. STRUCTURE OF THE PAPER

The following shows the structure of the paper. The literature review and problem definition of this study are presented in Section II of this paper. Section III describes the fundamentals of the distributed SDN control network, OpenFlow protocol rules, and network model. Section IV discusses the proposed enhanced TSSM scheme and matching algorithms, and Section V presents the performance evaluation of the proposed approach. Finally, in Section VI, the conclusion statement is presented.

III. DISTRIBUTED SDN CONTROLLER

This section discusses the architecture of the distributed SDN control network, the switch transfer mechanism in the OpenFlow protocol, and network models.

A. DISTRIBUTED SDN CONTROL NETWORK ARCHITECTURE

In a distributed SDN control network, two popular control methods are commonly used: (i) hierarchical control and (ii) flat control, also known as circular chain control [8]. In the hierarchical technique, the central distributed controller (called the leader) has a global perspective of the network and updates network regulations and newest algorithms to the sub controllers, as illustrated in Fig. 1. (a). The sub controller controls (oversees) the subnet of its switches and transmits its status to the leader. It should be emphasized that if the original leader is broken down in the hierarchical technique, a new leader will be chosen [15]. In the case of circular chain control, controllers have information about the network's local perspective and authority over its own subnet. The associated controllers exchange information in a distributed fashion, as shown in Fig. 1. (b).

In this article, the hierarchical technique is used to implement the suggested switch migration methodology. The leader oversees monitoring the condition of each sub controller and implementing the TSSM scheme to pick the lightly loaded controller over the overloaded controller during flow variations, flow traffic, impulses, distributed denial of service, and so on. Following that, two sub controllers (overloaded and lightly laden) are committed to sharing workloads and migrating the switch as needed.

To avoid undesired switch migrations, the threshold level of the sub controller is likewise established in the leader. When the workload of the controller exceeds the controller's threshold level, it is deemed overloaded, and it is selected based on the controller's maximum capacity and reserve capacity. Generally, network administrators recommend that the threshold level be set between 90 and 95% of the full capacity. The controller's threshold level is also stated as its maximum workload, and it is specified in Eq. (1).

$$\Phi_C = \lambda_C - \upsilon_C, \tag{1}$$

where,

$\Phi_C \longrightarrow$ Threshold workload level of the controller

$\lambda_C \longrightarrow$ Maximum workload capacity of the controller
$\upsilon_C \longrightarrow$ Reserve workload capacity of the controller.

B. SWITCH TRANSFER PROCESS IN OPENFLOW PROTOCOL

OpenFlow allows switch transfers between subnets and establishes connections with many controllers. Each related controller C_X determines the following duties from the perspective of switch S_L.

- OFPCR_ROLE_EQUAL (Equal): This default role grants controller C_X complete authority to switch S_L and allows C_X to send commands to S_L and receive status information. Similarly, when S_L is operating in this capacity, all controllers have complete access to it.
- OFPCR_ROLE_SLAVE (Slave): When the controller C_X role is set to slave, C_X can only read the state of switch S_L.
- OFPCR ROLE MASTER (Master): It is similar to an equal role, and controller C_X has full power over S_L. It is insisted, however, that only one controller (e.g., C_X) is considered a master controller for a switch S_L, and all other controllers are considered slaves to switch S_L.

The OpenFlow protocol defines the switching process, which is seen in Fig. 2. The master controller initiates the switch transferring operation since it has complete control over the switch. For example, controllers C_X and C_Y are the master and targeted (slave) controllers for the switch S_L, respectively. It is insisted that overloaded controllers use leader to move a switch to other controllers for workload balance (controller). After receiving a directive from the leader, the master controller (C_X) sends a transferring request for switch S_L to the targeted controller C_Y. Following that, controller Cq requests that the switch S_L alter the role of S_L control to master rather than slave using the Role Request (Master) message, and the switch S_L responds to C_Y with the Role Reply message (Master). After all, C_Y sends a notification message to C_X indicating the successful migration of switch S_L, and controller C_X subsequently operates as a slave controller for switch S_L.

OpenFlow protocol versions 1.2, 1.3, 1.4, and 1.5 enable switch migration (most recent version). It has been discovered that OpenFlow regulation simply instructs how to modify (migrate) the switches between controllers for their tasks and exchange messages between controllers. However, OpenFlow does not specify how to choose target controllers and switches for migration. The proposed enhanced TSSM method optimizes controller selection and determines when switch migration should occur during the TSSM period.

C. NETWORK MODELLING

Let us imagine an SDN-based network with a collection S_N of switches and a collection C_N of controllers. A switch (e.g., S_L) in S_N is controllable by a controller in C_N (e.g., C_X) with the model of one switch is controlled by a controller concurrently advocated by OpenFlow, i.e., C_X acts as a master controller for S_L and may be altered after the switch migration.

Packet In messages (PIMs) sent from switches determine each controller's workload. Switch workload ($\delta_{(t)}$) is calculated specifically by the number of PIMs created by a switch during each period 't'. Following that, controller workload capacity is defined as the maximum number of PIMS that may be processed in each period. For example, if controller C_X manages switches S_a to S_z, the workload of controller C_X is determined as follows:

$$\Upsilon_{C_X} = \sum_{S_a}^{S_z} \delta(t). \quad (2)$$

In general, the controller's maximum workload (Υ_C) should be smaller than its maximum capacity (λ_C), considering the need for reserve load under unwanted scenarios such as flow fluctuation, sudden demand, and so on. Hierarchical control of DSC architecture is studied in this work; hence, the leader receives workload from all controllers at each period and directs switch migration across controllers, as necessary.

IV. PROPOSED SWITCH MIGRATION SCHEME

The controller placement technique or network operators are used to set the network switches at the first stage, with each switch managed by a master controller. As described in the preceding section, conventional switch migration methods include migrating a switch at the start of the period as well as migrating the entire switch even if it is not necessary. As a result, the link between controllers and switches remains constant for the duration. In the case of TSSM, switch migration is enabled via time-sharing, and switches in the network can dynamically change their connections with the controller at any moment. Furthermore, as mentioned in section 2.1, the TSSM approach efficiently overcomes the controller ping-pong challenge. Nonetheless, controller resource consumption is greater during the TSSM time, which may raise the method's migration cost when compared to other migration techniques since it allows more than one controller to share their (switch) loads during the TSSM period. It is discovered that migration costs are approximated based on the number of controllers and switches used. As a result, this research suggests an approach that greatly decreases the number of controllers associated with switches during time sharing migration depending on flow characteristics. We developed a coalitional game strategy to establish the best possible connection between switches and controllers during the time-sharing migration phase, reducing the number of controllers connected with the switch and, as a result, controller resource consumption and migration cost are reduced. The algorithms listed below are intended to ensure the effective completion of the proposed switch migration method.

ALGORITHM 1: IDENTIFYING OVERLOADED AND LIGHTLY LOADED CONTROLLERS

This algorithm ensures that all overloaded (referred to as busy) and lightly loaded controllers (referred to as assistant or target controllers) in the given network are found, as represented by C_{busy} and C_{light}, respectively. The burden of each controller (e.g., Υ_{C_X}) is evaluated using Eq. (2) by adding the loads of each switch in the subnet (e.g., $\delta_{L,t}^{(X)} + \delta_{M,t}^{(X)} + \cdots$) and is specified in

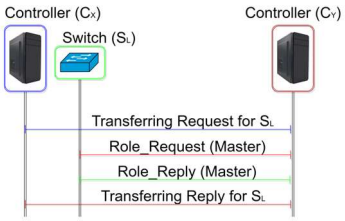


Figure 2. Switch transferring process in OpenFlow Protocol

Algorithm 1: Identifying Overloaded and Lightly Loaded Controllers

1 $C_{busy} \leftarrow$ ø and $C_{light} \leftarrow$ ø ;
2 **foreach** $C_X \in C$ **do**
3 $\Upsilon_{C_X} \leftarrow 0$;
4 **foreach** $S_L \in S_X$ **do**
5 $\Upsilon_{C_X} \leftarrow \Upsilon_{C_X} + \delta_{L,t}^{(X)}$;
6 if $\Upsilon_{C_X} > \Phi_{C_X}$ then
7 $C_{busy} \leftarrow C_{busy} \cup \{C_X\}$;
8 else if $\Upsilon_{C_X} < \mu \times \Phi_{C_X}$ **then**
9 $C_{light} \leftarrow C_{light} \cup \{C_X\}$;
10 If $C_{busy} \neq$ ø and $C_{light} \neq$ ø **then**
11 Use Algorithm 2 for load balancing between C_{busy} and C_{light} ;

Algorithm 2: Switch Migration Segment for Load Balancing

1 SORT (C_{busy} , $\Upsilon_{C_X} - \Phi_{C\_X}$);
2 SORT (C_{light}, $\Phi_{C\_Y} - \Upsilon_{C_Y}$);
3 **foreach** $C_X \in C_{busy}$ **do**
4 SORT (S_X , $\delta_{L,t}^{(X)}$);
5 **while** $\Upsilon_{C_X} > \Phi_{C\_X}$ **do**
6 **if** C_{light}= ø **then**
7 Cease this module ;
8 Pick the optimized controllers $[C_{Y1}, C_{Y2},...]$from C_{light} ;
9 (Controller-Switch Association Matrix) ← Algorithm 3 (Request PIM´s of Switch, Switches from C_{busy})
10 $(S_L, [\tau_1 , \tau_2 , \ldots] , [\rho_1 , \rho_2 , \ldots])$ ← Algorithm 4 (C_X , $[C_{Y1}$, $C_{Y2},...]$) ;
11 Transfer S_L to $[C_{Y1}, C_{Y2},...]$'s subnet after $[\tau_1 , \tau_2 , \ldots]$ units of time;
12 $\Upsilon_{C_X} \leftarrow \Upsilon_{C_X} - [\rho_1 , \rho_2 , \ldots]$;
13 $\Upsilon_{C_{Y\_1}} \leftarrow \Phi_{C\_Y} + [\rho_1 , \rho_2 , \ldots]$;
$\Upsilon_{C_{Y\_2}} \leftarrow \Phi_{C\_Y} + [n_1 , n_2 , \ldots]$;
14 **if** $\Upsilon_{C_{Y[1,2,\ldots]}} \geq \mu \times \Phi_{C\_Y[1,2,\ldots]}$ **then**
15 $C_{light} \leftarrow C_{light} \setminus , [C_{Y1}, C_{Y2},...]$;
16 **else**
17 SORT (C_{light}, $\Phi_{C\_Y} - \Upsilon_{C_Y}$) ;

the method code between 3 and 5 lines. Following that, the controller workload (e.g., Υ_{C_X}) is compared to the threshold level (Φ_{C_X}), and if it is more than the threshold level, the controller is deemed overloaded and included in the overload controllers (described in lines 6 -7) unit in the leader. Then, in line 8, lightly laden controllers are chosen based on a lightly loaded coefficient ´μ´, with a value between 0.8 and 0.85 (specified by network managers). Following that, the lightly loaded coefficient is multiplied by the threshold value, and if the workload of the controllers is less than the multiply value, it is regarded a lightly loaded controller and is added to the leader's lightly loaded controller unit. It is required that switch migration take place when both the C_{busy} and C_{light} controllers are not empty, as shown in line 10.

ALGORITHM 2: ORDERING THE OVERLOADED AND ASSISTING CONTROLLERS, AS WELL AS SWITCH MIGRATION

This algorithm's goal is to distribute workload across controllers by identifying a pair of overloaded and lightly burdened controllers. The SORT function aids in the organization of overloaded and lightly laden controllers in decreasing workload order. Line 1 of the code sorts the overload controllers, whereas line 2 sorts the information about the lightly loaded controller. As a result, a controller with very excess capacity will be prioritized in contributing to the task of an overloaded (busy) controller. The code in lines 3–17 tackles each controller in the network using a for-loop, from the most overloaded to the least overloaded. Line 4 arranges the switches under C_X management in decreasing order based on their workload. The while loop on lines 5-16 continues to reduce the burden of the C_X by moving a switch until it reaches the threshold workload. However, if there is no assistant controller to assist (i.e., C_{lig} is empty) and there are still overloaded controllers in the domain, algorithm 2 ends as shown in lines 6-7. Otherwise, if we wish to pick a lightly loaded controller C_Y for workload sharing, the time-sharing switch migration technique must be enabled. Initially, Algorithm 3 is used to determine the best controllers $[C_{Y1}, C_{Y2},...]$ for TSSM in terms of controller resource usage and migration cost. Following the discovery of the optimal controllers, the TSSM scheme based on Algorithm 4 is run. As seen in line 10, the result of Algorithm 4 gives three output parameters. In which ´τ´ specifies the time switch S_L should migrate to other controllers,

Algorithm 3: Selection of Optimized Controller for TSSM Scheme

1 **Input:** Organized light and busy controllers =$\{C_{busy}\}$, $\{C_{light}\}$ obtained from Algorithm 2 ;
2 SORT ($\delta_{L,t}^{(X)}$, $\delta_{M,t}^{(X)}$, $\delta_{N,t}^{(X)}$, ..) ;
3 Capacity and redundant load for each controller under a leader
4 Traffic Matrix: $T^{'} = [T_{XY}]$
5 **Initialization:** $T = [T_{XY}]$, $C_N = [C_{XY}^{'}]$, λ_C, υ_C
6 **repeat**
7 Every switch performs its most desired migration.
8 Initial migration pair S_L: $C_X \rightarrow C_Y$;
9 **for all** controllers **do**:
10 **if** S_L: $C_X \rightarrow C_Y$; and $\Upsilon_{C_Y} \leq \lambda_{C\_Y}.\upsilon_{C\_Y}$: satisfy migration does not violate capacity constraint.
11 **if** migration value $(S_n, C_n) < 0$: consider a weight factor between control resource consumption and control traffic overhead.
12 Implement switch migration selection $S_L \rightarrow C_Y$
13 Update $C_N = [C_{XY}^{'}]$;
14 **end if**
15 **end if**
16 **end for**
17 **Until** no proposals have been made by the switches

Algorithm 4: Time to Switch Migration Estimating Segment

1 $\Delta_{over} \leftarrow \min(\Upsilon_{C_X} - \Phi_{C\_X})$ & $\Delta_{light} \leftarrow \max(\Phi_{C\_Y} - \Upsilon_{C_Y})$;
2 $S_X^{\chi} \leftarrow \emptyset$ and $S_X^{\psi} \leftarrow \emptyset$;
3 **foreach** $S_L \in S_X$ **do**
4 **if** $\delta_{L,t}^{(X)} \geq \Delta_{over}$ **then**
5 $S_X^{\chi} \leftarrow S_X^{\chi} \cup \{S_L\}$;
6 **else**
7 $S_X^{\psi} \leftarrow S_X^{\psi} \cup \{S_L\}$;
8 **if** $S_X^{\chi} \neq \emptyset$ **then**
9 $S_L \leftarrow$ the last switch of S_X^{χ} ;
10 $\tau = \left[\%\ of\ \Delta_{light}\ with\ respect\ to\ \Delta_{over}\right] \times (L_t)$;
11 **else**
12 $S_L \leftarrow$the first switch of S_X^{ψ};
13 $\tau \leftarrow 0$ then $\rho \leftarrow \delta_{L,t}^{(X)}$;
14 $\delta_{L,t}^{(X)} \leftarrow \delta_{L,t}^{(X)} - \rho$ and $\delta_{L,t}^{(Y)} \leftarrow \rho$;
15 **return** (S_L, τ, ρ);

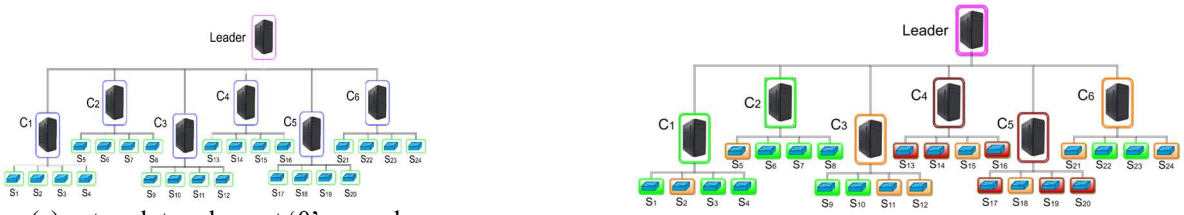
whilst 'ρ' specifies the number of PIMs to be migrated to each controller. Following that, workload updates of C_X and [C_{Y1}, C_{Y2},...] are performed in lines 11 to 13, and if [C_{Y1}, C_{Y2},...] exceeds the threshold level, these controllers are removed from the lightly loaded controllers as shown in line 14, otherwise these controllers are returned to the lightly loaded controller unit as shown in lines 17 and 2.

ALGORITHM 3: OPTIMIZATION OF THE CONTROLLER FOR THE TSSM SCHEME TO SAVE MIGRATION COSTS

This algorithm's goal is to produce efficient controllers for TSSM operation. The optimized controller is chosen based on flow characteristics to decrease controller resource usage and, as a result, switch migration cost. The coalitional game strategy [33] is used for optimal controller selection and is shown in Algorithm 3. This algorithm requires the PIMs of each switch in the overloaded controller C_X, as well as the controller's threshold level, network topology map, and so on. Between lines 3 and 12, the flow sort function evaluates the total quantity of flow in each path and sorts it in ascending order. Lines 4-6 execute and choose a controller that covers most of the switches in the route.

ALGORITHM 4: TIME TO SWITCH MIGRATION ESTIMATING SEGMENT

After defining the best lightly loaded controllers (C_{Y1}, C_{Y2},...) using Algorithm 3, they are paired with an overloaded controller to accomplish three tasks using Algorithm 4. The tasks are as follows: (i) choose a switch (from an overloaded controller) to share their burden with lightly loaded controllers, (ii) compute the switch migration time (τ), and (iii) calculate the number of PIMs (ρ) that lighter loaded controllers will process. Line 1 of Algorithm 4 is executed, with 'Δ_{light}' representing the remaining capacity of the lightly loaded controllers and 'Δ_{over}' representing the lowest amount of overload in the overloaded controllers. Following that, switches in the overloaded controllers are divided into two subnets, S_X^{χ} and S_X^{ψ}, respectively; if the switch load is greater than ´Δ´, it is sorted in S_X^{χ} with decreasing load order, and S_X^{ψ}



(a) network topology at '0' second

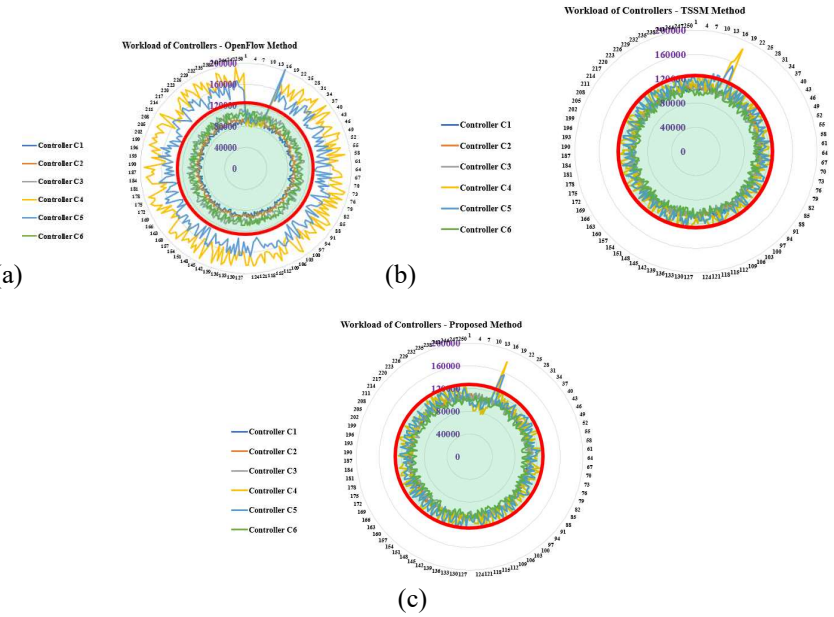
(b) network topology at 16th second

Figure 3. Network topology used in the simulation test platform

includes remaining switches in the overloaded controllers; respecting codes are given in lines 2-7. In order to reduce the number of migrations (executed in lines 8 – 9), switches near ′Δ′ (might be the very last switch in S_X^{χ} based on load soring order) are selected in the S_X^{χ} subnet for migration. This is because a minimal amount of overload in the overloaded controllers can easily be placed in the lightly loaded controllers. The estimated switch migration time is determined by the number of PIMs generated in the switch, the Δ_{light} in the optimum lightly loaded controllers, and the Δ_{over} in the switch. For example, if Δ_{lig} is half the Δ_{over} value and the rate of PIMs created is constant, the switch migration time is expected to be half the period duration provided in Eq. (3). If $\tau = 0$, switch migration happens at the start of the period, as shown in line 13. Furthermore, once the switches in the S_X^{χ} subnet are empty, the S_X^{ψ} subnet is evaluated for better load balancing even though it is not overloaded, as seen in lines 11 and 12. This procedure will be continued until all the controllers are load balanced for each switch in the time-sharing scheme using optimum controller finding (Algorithm 3) and then returned to Algorithm 2.

V. EVALUATION AND ANALYSIS

The proposed switch migration strategy's performance is tested using time domain simulation analysis. As illustrated in Fig. 3, the ONOS platform is used as the test platform, and a hierarchical DSC design is used for the experimental network, which contains 7 controllers and 24 switches. As a result, one controller acts as a leader, and its major purpose is to oversee the other six controllers in the network; however, it is not involved in switch management; the secondary six controllers operate their switches in their subnet. This test platform considers simulation duration to be 250 seconds divided into 50 phases. Each secondary controller has a PIMs processing capacity of 800,000 PIMs every 5 second interval. Furthermore, the barrier for each controller is set at 640,000 PIMs every period. As a result, the overall controller affordable load is estimated to be 3.84 ×106 PIMs each period. The switches loads are divided into three levels: (i) light load, (ii) medium load, and (iii) big load. Each switch generates roughly 17,000 PIMs per second under mild load, whereas a switch producing 33,500 PIMs per second is considered medium load. However, if a switch generates more than 51,000 PIMs per second, it is considered a big load. If all switches are lightly loaded, the overall controller affordable load is 2 ×106 PIMs per period, which is approximately 48% of the total controller affordable load. However, if all switches are deemed heavy loads, the overall load is 6.01 ×106 PIMs per period, which is significantly greater than the total controller affordable load. As a result, in this simulation research, the simulation begins with a minimal load in all switches, and the load is randomly raised in the switches using the cbench tool as simulation duration advances, to evaluate the performance of the switch



(a) (b)

(c)

Figure 4. Comparison of workload of controllers: (a) OpenFlow method, (b) Conventional TSSM, (c) Proposed method

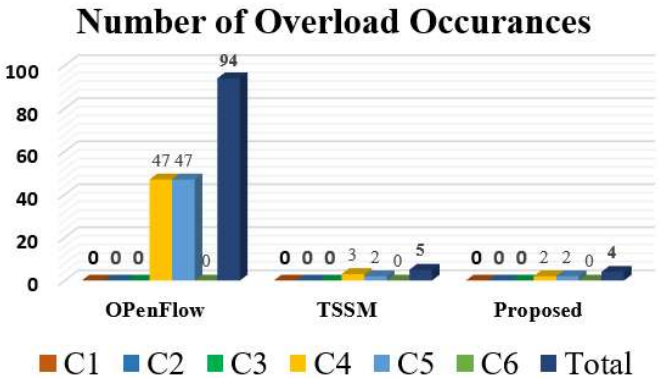


Figure 5. Comparison of number of overload occurrences in conventional and proposed method

migration approach. For example, at the 16th second time, ten switches (S1, S3, S4, S6, S7, S8, S9, S10, S22, S23,) are creating about 17,000 PIMS/s, eight switches (S2, S5, S11, S12, S15, S18, S21, S24,) are generating 33,500 PIMs/s, and the remaining switches (S13, S14, S16, S17, S19, S20) are carrying 51,000 PIMs/s. As a result, the total controller workload is 3.772 ×106 PIMs each period, and switch migration must occur using both the traditional (full switch) and TSSM schemes. Three examples are studied for assessing the performance of the suggested method: (i) work loads of controllers, (ii) overload events, and (iii) controller resource consumption.

TEST 1: WORKLOAD OF CONTROLLERS

As previously stated, each controller may process up to 640,000 PIMs every period, and if the controller has processed more than 128,000 PIMs/s, it is deemed overloaded. Two standards approach, (i) OpenFlow and (ii) TSSM schemes, are studied in this test, and their test results are compared with the proposed method for assessing performance.

Because switch migration is not done in the OpenFlow technique, controllers C4, and C5 are significantly overloaded, as seen in Fig. 4a, based on PIMs generated in the switches. During this time, controllers C4 and C5 must handle about 932,000 PIMs every period, which exceeds their maximum capacity (800,000 PIMs per period) and causes unforeseen challenges in the networking domain. In the case of the TSSM scheme, it distributes workload across controllers via time sharing migration and ensures that all controllers are under their threshold limits, as illustrated in Fig. 4b. Furthermore, the Ping-Pong problem (no high leaps, and often transmitted switches are treated as nil) is not detected in the test results. The suggested switch migration scheme's test results are shown in Fig. 4c. When compared to the TSSM scheme, load sharing between controllers is substantially flatter (i.e., almost all controllers are sharing around similar load, which improves efficiency and reduces downtime or maintenance activities).

TEST 2: NUMBER OF OVERLOAD OCCURRENCES

This test is important for determining the performance of the switch migration technique by evaluating the number of overload occurrences for the controllers for the whole duration (250S). Fig. 5 shows a comparison of overload occurrence for all three approaches. It demonstrates that the OpenFlow method provides a high number of over-load occurrences because there is no switch migration action, and thus controllers C1, C2, C3, and C6 are in the lightly loaded range, whereas C4, and C5 are highly loaded, and these controllers are completely overloaded during the given period.

In the case of TSSM, it has considerably decreased the number of overload events for the controller since it avoids the ping-pong problem and so switches that are repeatedly moved are ignored. When compared to the TSSM system, the proposed method reduces the amount of overload incidents

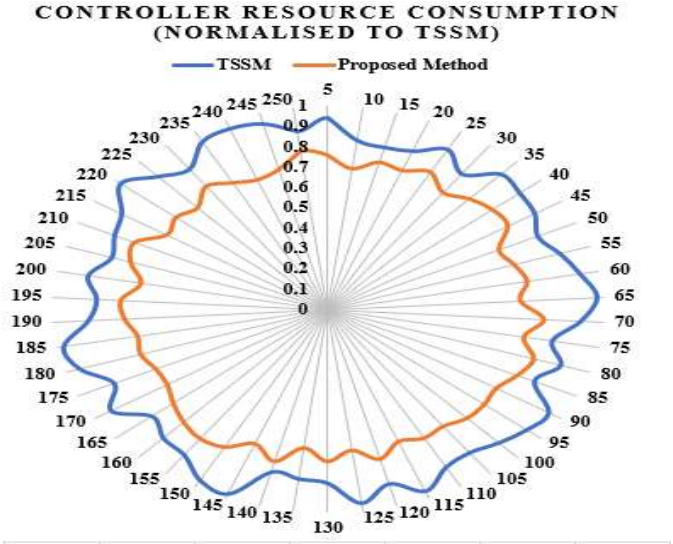


Figure 6. Comparison of controller resource consumption between TSSM and proposed switch migration method

even more. During time sharing migration, the suggested technique employs more than one optimal controller as a lightly loaded controller, which may minimize the frequency of overload events. Because, in the conventional TSSM method, if one controller is not sufficient to share the load of the switch (this controller may be considered initially as excess in this situation), then it is necessary to find another controller for switch sharing. This may occur when the requirement of load sharing is high in the over-loaded controller and lightly loaded single converters are insufficient to handle this load. The proposed strategy, on the other hand, selects more optimal controllers based on load sharing and minimizes unnecessary processing and overload situations.

TEST 3: CONTROLLER RESOURCE CONSUMPTION

This test could be utilized to determine the migration cost of switch migration techniques based on controller resource usage. Controller resource consumption describes how many controllers and switches are used. It should be noted that minimizing the number of controllers associated with the switches minimizes the network's switch migration cost. Because OpenFlow is not conducted during the switch migration event, it is excluded from this assessment research. When compared to alternative switch migration methods, the standard TSSM has a lower migration cost. However, it is greater when compared to the proposed switch migration technique since the proposed approach selects the appropriate controllers for workload sharing based on flow characteristics, which minimizes controller resource consumption and switch migration cost. Fig. 6 depicts the control resource usage of the switch migration strategy. When compared to the conventional TSSM system, the suggested switch migration approach saves approximately 18% on switch migration costs.

VI. CONCLUSION

This research offers an enhanced TSSM methodology that addresses the issue of higher switch migration cost in the standard TSSM method by locating several optimum target controllers throughout the time-sharing period. Flow characteristics are used to determine the best controllers applying a coalitional game strategy method. Furthermore, the suggested switch migration strategy provides TSSM benefits that have overcome the ping-pong controller challenge. The ONOS platform was used to evaluate the performance of this study, and it was discovered that the modified TSSM scheme outperformed the standard TSSM approach in terms of controller workload sharing, number of overload events, and controller resource consumption. When compared to the typical TSSM, it decreases controller resource use by 18%.

References

[1] N. Anerousis, P. Chemouil, A. A. Lazar, N. Mihai, and S. B. Weinstein, "The origin and evolution of open programmable networks and SDN," *IEEE Commun. Surveys Tuts*., vol. 23, no. 3, pp. 1956–1971, 2021. https://doi.org/10.1109/COMST.2021.3060582.

[2] Y.-C. Wang and H. Hu, "An adaptive broadcast and multicast traffic cutting framework to improve Ethernet efficiency by SDN," *J. Inf. Sci Eng*., vol. 35, no. 2, pp. 375–392, 2019.

[3] M. Alsaeedi, M. M. Mohamad, and A. A. Al-Roubaiey, "Toward adaptive and scalable OpenFlow-SDN flow control: A survey," *IEEE Access*, vol. 7, pp. 107346–107379, 2019. https://doi.org/10.1109/ACCESS.2019.2932422.

[4] J. H. Cox, R. Clark, and H. Owen, "Leveraging SDN and WebRTC for rogue access point security," *IEEE Trans. Netw. Service Manag*., vol. 14, no. 3, pp. 756–770, 2017. https://doi.org/10.1109/TNSM.2017.2710623.

[5] Y.-C. Wang and S.-Y. You, "An efficient route management framework for load balance and overhead reduction in SDN-based data center networks," *IEEE Trans. Netw. Service Manag*., vol. 15, no. 4, pp. 1422–1434, 2018. https://doi.org/10.1109/TNSM.2018.2872054.

[6] W. Iqbal et al., "ALAM: Anonymous lightweight authentication mechanism for SDN-enabled smart homes," *IEEE Internet of Things Journal*, vol. 8, no. 12, pp. 9622-9633, 2021, https://doi.org/10.1109/JIOT.2020.3024058.

[7] Y.-C. Wang and R.-X. Ye, "Credibility-based countermeasure against slow HTTP DoS attacks by using SDN," *Proceedings of the IEEE Annu. Comput. Commun. Workshop Conf*., 2021, pp. 890–895. https://doi.org/10.1109/CCWC51732.2021.9375911.

[8] F. Bannour, S. Souihi, and A. Mellouk, "Distributed SDN control: Survey, taxonomy, and challenges," *IEEE Commun. Surveys Tuts*., vol. 20, no. 1, pp. 333–354, 2018. https://doi.org/10.1109/COMST.2017.2782482.

[9] J. Lu, Z. Zhang, T. Hu, P. Yi, and J. Lan, "A survey of controller placement problem in software-defined networking," *IEEE Access*, vol. 7, pp. 24290–24307, 2019. https://doi.org/10.1109/ACCESS.2019.2893283.

[10] H. Sufiev, Y. Haddad, L. Barenboim, and J. Soler, "Dynamic SDN controller load balancing," *Future Internet*, vol. 11, no. 3, pp. 1–21, 2019. https://doi.org/10.3390/fi11030075.

[11] Y. Wu, S. Zhou, Y. Wei, and S. Leng, "Deep reinforcement learning for controller placement in software defined network," *Proceedings of the IEEE INFOCOM Workshop*, Toronto, Canada, 2020, pp. 1254–1259. https://doi.org/10.1109/INFOCOMWKSHPS50562.2020.9162977.

[12] Y.-C. Wang and Y.-C. Wang, "Efficient and low-cost defense against distributed denial-of-service attacks in SDN-based networks," *Int. J. Commun. Syst*., vol. 33, no. 14, pp. 1–24, 2020. https://doi.org/10.1002/dac.4461.

[13] F. Tang, H. Zhang, L. T. Yang, and L. Chen, "Elephant flow detection and load-balanced routing with efficient sampling and classification," *IEEE Trans. Cloud Comput*., vol. 9, no. 3, pp. 1022–1036, 2021. https://doi.org/10.1109/TCC.2019.2901669.

[14] Y.-C. Chan, K. Wang, and Y.-H. Hsu, "Fast controller failover for multidomain software-defined networks," *Proceedings of the Eur. Conf. Netw. Commun*., Paris, France, 2015, pp. 370–374.

[15] W. H. F. Aly, "Controller adaptive load balancing for SDN networks," *Proceedings of the Int. Conf. Ubiquitous Future Netw*., Zagreb, Croatia, 2019, pp. 514–519.

[16] T. Hu, J. Zhang, L. Cao, and J. Gao, "A reliable controller deployment strategy based on network condition evaluation in SDN," *Proceedings of the IEEE Int. Conf. Softw. Eng. Serv. Sci*., Beijing, China, 2017, pp. 367–370. https://doi.org/10.1109/ICSESS.2017.8342934.

[17] T. Kim, J. Myung, and S.-E. Yoo, "Load balancing of distributed datastore in OpenDaylight controller cluster," *IEEE Trans. Netw. Service Manag*., vol. 16, no. 1, pp. 72–83, 2019. https://doi.org/10.1109/TNSM.2019.2891592.

[18] Y.-C. Wang and E.-J. Chang, "Cooperative flow management in multidomain SDN-based networks with multiple controllers," *Proceedings of the IEEE Int. Conf. Smart Commun. Improving Qual. Life Using ICT IoT AI*, Charlotte, USA, 2020, pp. 82–86. https://doi.org/10.1109/HONET50430.2020.9322815.

[19] S. Nithya, M. Sangeetha, K. N. A. Prethi, K. S. Sahoo, S. K. Panda, and A. H. Gandomi, "SDCF: A software-defined cyber foraging framework for cloudlet environment," *IEEE Trans. Netw. Service Manag*., vol. 17, no. 4, pp. 2423–2435, 2020. https://doi.org/10.1109/TNSM.2020.3015657.

[20] K. S. Sahoo, P. Mishra, M. Tiwary, S. Ramasubbareddy, B. Balusamy, and A. H. Gandomi, "Improving end-users' utility in software-defined wide area network systems," *IEEE Trans. Netw. Service Manag*., vol. 17, no. 2, pp. 696–707, 2020. https://doi.org/10.1109/TNSM.2019.2953621.

[21] A. Dixit, F. Hao, S. Mukherjee, T. V. Lakshman, and R. Kompella, "Towards an elastic distributed SDN controller," *ACM SIGCOMM Comput. Commun. Rev*., vol. 43, no. 4, pp. 7–12, 2013. https://doi.org/10.1145/2534169.2491193.

[22] Z. Min, Q. Hua, and Z. Jihong, "Dynamic switch migration algorithm with Q-learning towards scalable SDN control plane," *Proceedings of the Int. Conf. Wireless Commun. Signal Process*., Nanjing, China, 2017, pp. 1–4. https://doi.org/10.1109/WCSP.2017.8171121.

[23] J. Cui, Q. Lu, H. Zhong, M. Tian, and L. Liu, "SMCLBRT: A novel load-balancing strategy of multiple SDN controllers based on response time," *Proceedings of the IEEE Int. Conf. High Perform. Comput. Commun*., Exeter, U.K., 2018, pp. 541–546.

[24] K. S. Sahoo et al., "ESMLB: Efficient switch migration-based load balancing for multicontroller SDN in IoT," *IEEE Internet Things J*., vol.

7, no. 7, pp. 5852–5860, 2020. https://doi.org/10.1109/JIOT.2019.2952527.
[25] T. Hu, J. Lan, J. Zhang, and W. Zhao, "EASM: Efficiency-aware switch migration for balancing controller loads in software-defined networking," *Peer-to-Peer Netw. Appl.*, vol. 12, pp. 452–464, 2019. https://doi.org/10.1007/s12083-018-0632-6.
[26] Y. Hu, W. Wang, X. Gong, X. Que, and S. Cheng, "BalanceFlow: Controller load balancing for OpenFlow networks," *Proceedings of the IEEE Int. Conf. Cloud Comput. Intell. Syst.*, Hangzhou, China, 2012, pp. 780–785. https://doi.org/10.1109/CCIS.2012.6664282.
[27] W. Lan, F. Li, X. Liu, and Y. Qiu, "A dynamic load balancing mechanism for distributed controllers in software-defined networking," *Proceedings of the Int. Conf. Meas. Technol. Mechatronics Autom.*, Changsha, China, 2018, pp. 259–262. https://doi.org/10.1109/ICMTMA.2018.00069.
[28] I. Maity, S. Misra and C. Mandal, "Traffic-aware consistent flow migration in SDN," *Proceedings of the 2020 IEEE International Conference on Communications (ICC-2020)*, 2020, pp. 1-6. https://doi.org/10.1109/ICC40277.2020.9148983.
[29] I. Maity, S. Misra and C. Mandal, "DART: Data plane load reduction for traffic flow migration in SDN," *IEEE Transactions on Communications*, vol. 69, no. 3, pp. 1765-1774, 2021, https://doi.org/10.1109/TCOMM.2020.3042271.
[30] B. Gorkemli, S. Tatlcıoglu, A. M. Tekalp, S. Civanlar, and E. Lokman, "Dynamic control plane for SDN at scale," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 12, pp. 2688–2701, 2018. https://doi.org/10.1109/JSAC.2018.2871308.
[31] G. Cheng and H. Chen, ''Game model for switch migrations in software defined network,'' *Electron. Lett.*, vol. 50, no. 23, pp. 1699–1700, 2014. https://doi.org/10.1049/el.2014.2086.
[32] W.-K. Lai, Y.-C. Wang, Y.-C. Chen and Z.-T. Tsai, "TSSM: Time-sharing switch migration to balance loads of distributed SDN controllers," *IEEE Trans. on Network and Service Management*, vol. 19, no. 2, pp. 1585-1597, 2022. https://doi.org/10.1109/TNSM.2022.3146834.
[33] Y. Zhang, Y. Ran, and Z. Zhang, "A simple approximation algorithm for minimum weight partial connected set cover," *J. Combinat. Optim.*, vol. 34, no. 3, pp. 956–963, 2017. https://doi.org/10.1007/s10878-017-0122-4.

***G. NISHANTHI** is currently working as an Assistant Professor in the department of computer science and engineering, RMK engineering college, Chennai, India. she is having more than three years of teaching and research experience in different institutions in computer science and engineering.*

She completed her B.E and M.E degrees in computer science & engineering from Anna university, Chennai in the year 2018, 2020, respectively. she published more than four papers in international journals and national conferences. Her areas of interest include network security, big data & data mining.

***R. DEEPA** is currently working as an Assistant Professor in the department of computer science and engineering, RMK engineering college, Chennai, India. she is having more than seven years of teaching and research experience in computer science and engineering.*

She completed her B.E and M.E degrees in information technology from Anna university, Chennai in the year 2008, 2013, respectively. Her areas of interest include Data Analytics. Machine Learning and Deep Learning. She is a life member of ISTE.

***S. GAYATHRI** is currently working as an Assistant Professor in the Department of Computer Science and Engineering, RMK Engineering College, Kavaraipettai, Chennai, India. She completed her M.E in Computer Science and Engineering Specialization from College of Engineering, Guindy, Anna University, Chennai in the year 2022. Her areas of interest include Data Science, Deep Learning, Image Processing and Computer Networks. She has published conference and journal papers in the deep learning and image processing domain. She is a life member in ISTE.*

***B JAISON** is currently working as a Professor in the Department of Computer Science and Engineering, RMK Engineering College, Chennai, India. He is having more than 24 years of teaching Experience in computer science engineering. He completed his M.E degree in Computer Science & Engineering from Anna University, Chennai in the year 2007 and Ph. D in Information and Communication Engineering from Anna University, Chennai in the Year 2015. He has published more than 50 Research Articles in International Journals and attended many International Conferences. His areas of interest include Data mining, Image Processing and Cloud Computing. He is a life member in IAENG, IACSIT and ISTE.*