Tabriz Journal of Electrical Engineering (TJEE), vol. 55, no. 4, 2025 Serial no. 114
DOI: 10.22034/tjee.2025.65116.4941

A Switch Migration-based Congestion Control Mechanism for Multi-domain SDNs

MohammadReza Jenabzadeh[1], Vahid Ayatollahitafti[2*], MohammadReza Mollakhalili Meybodi[3], Mohammadreza Mollahoseini Ardakani[3]

[1]Department of Computer Engineering, Ya. C., Islamic Azad University, Yazd, Iran.
[2]Department of Computer Engineering, Taft. C., Islamic Azad University, Taft, Iran.
[3]Department of Computer Engineering, May. C., Islamic Azad University, Maybod, Iran.
mr.jenabzadeh@iau.ac.ir; vahid.ayat@iau.ac.ir; meybodi@iau.ac.ir; mr.mollahoseini@iau.ac.ir

*Corresponding author

Received: 21/12/2024 , Revised: 11/05/2025, Accepted: 08/06/2025

Abstract
With the continuous advancement of Software-Defined Networks (SDNs), the adoption of a distributed control plane architecture has become increasingly necessary. One of the primary challenges in these networks is the variable load on the controllers where high loads can lead to congestion. Such congestion can significantly degrade network efficiency. Although previous studies have attempted to address this issue, they have largely failed to effectively manage load exchange between the control plane and the data plane. This paper proposes a migration-based congestion control mechanism for multi-domain SDNs. In this approach, when a controller experiences high load and congestion, selected switches are migrated from the overloaded controller to one with a lower load. If the migration risks congesting the new controller, the mechanism swaps switches between controllers with minimal migrations, drawing inspiration from the Kadane algorithm to prevent congestion elsewhere. The proposed mechanism was evaluated using the D-ITG and IPerf tools with the RYU controller, demonstrating improved system performance. Simulation results show that the mechanism outperforms the baseline approach, increasing average network throughput by approximately 10%, while reducing average delay and jitter by about 30% and 25%, respectively. Furthermore, a comparison between the proposed method and the OptiGSM method reveals that the proposed method offers superior throughput and lower delay, although the OptiGSM method exhibits less jitter than the proposed method.

Keywords: Software-defined networks, Switch migration, Congestion control, RYU controller, D-ITG, Redis

1. Introduction

Software-Defined Networks (SDNs) represent an emerging paradigm in telecommunications and computer networking. Their primary objective is to address challenges such as the complex and cumbersome management inherent in traditional IP-based networks, which dominate modern communication systems [1]. As a broad and versatile concept, Software-Defined Networking, by decoupling the data plane from the control plane, eliminates many limitations and issues of existing networks [2]. This separation is facilitated by APIs such as OpenFlow, which enable communication between controllers and switches. Fig. 1 illustrates the architecture of Software-Defined Networks.

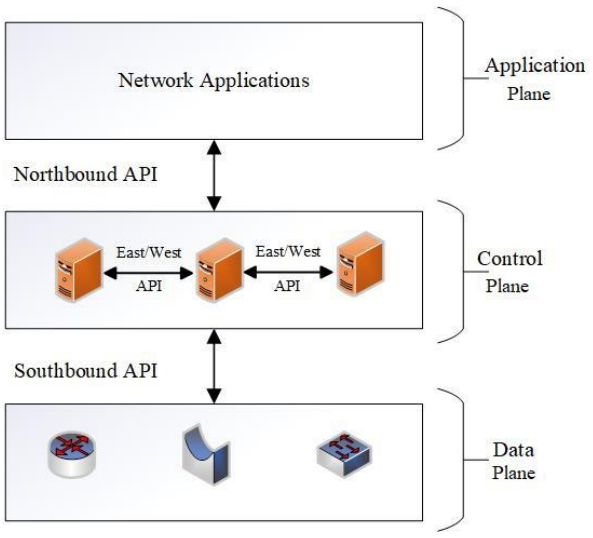


Fig.1 The architecture of software-defined networks.

As network scale expands, a single centralized controller becomes insufficient to handle growing traffic demands. Consequently, researchers have proposed a multi-controller architecture that is logically centralized but physically distributed, dividing the network into distinct

domains managed by multiple controllers. Like traditional networks, SDNs face challenges such as performance, scalability, security, and reliability. Programmability and flexibility in flow processing enhance system performance. The number of switches, controllers, and network topology are critical to scalability. Security remains a sensitive concern between the data plane and control plane, with unique issues arising in SDNs. Reliability, another key challenge, varies with factors like load balancing, controller placement, and congestion management. Congestion occurs when resource demand exceeds available capacity, leading to packet loss and subsequent retransmissions [3]. This state increases queue delays, packet loss rates, and retransmission frequency.

Various congestion control methods have been proposed for SDNs, including topology adjustments, flow table management, rerouting to less-congested paths, and switch migration. Many researchers have explored switch migration, leveraging network statistics to mitigate congestion under different scenarios. While some have simulated these approaches using tools like MATLAB, few have tested them in real-world environments. This paper introduces a switch migration-based mechanism to control congestion in SDNs using the RYU controller. The approach involves three steps: (1) collecting network statistics (e.g., packet-in counts, delays, queue statistics, and controller CPU/memory usage) that contribute to congestion; (2) applying an objective function to identify at-risk controllers, candidate switches for migration, and target controllers; and (3) executing migration decisions to alleviate congestion.

The main contributions of this work are:

- A framework for controlling and preventing congestion.
- A switch migration model balancing migration cost and congestion thresholds, inspired by the Kadane algorithm.
- An Integer Linear Programming (ILP) model for congestion control and prevention.

The paper is structured as follows: Section 2 reviews related work, Section 3 presents the system framework, Section 4 details the system model and problem formulation, Section 5 explains the proposed method and algorithm, Section 6 evaluates performance through simulation, and Section 7 provides a conclusion.

2. Related works

In recent years, computer science researchers have focused on challenges arising from the growth of software-defined networks. Congestion prevention and control remain critical issues in both traditional and SDN environments. Literature highlights load balancing, quality of service (QoS), and traffic engineering as key aspects of congestion management. Song et al.[4] provided an algorithm by measuring the load of switches and setting a threshold for it that if the load of the switch exceeds this threshold, the controller sets a new path without using the same switch. Zhu et al.[5] have introduced a reliable, efficient, congestion-aware, and robust Multicast Transmission Control Protocol (MCTCP). MCTCP can dynamically bypass dense and broken links and achieves high performance and strength. Hu et al.[6] obtained the port information per switch to monitor congested connections by providing a software-defined congestion control algorithm in the controller. The congestion control mechanism starts in case of network congestion, some suitable data streams are chosen, and the shortest path to congestion control is extracted. This process enables efficient use of network bandwidth resources and increases link usage. One drawback of network congestion is quality of service (QoS) reduction, Rahman et al.[7] analyzed the efficiency of congestion control mechanisms. Implementing the LLDP algorithm showed that the LLDP protocol decreases the number of collisions in the network. Therefore, this protocol mitigates congestion and increases the packet delivery ratio (PDR) and the network throughput. Wang et al.[8] developed a rate-based congestion control scheme to overcome Data-Center-Bridging (DCB) problem by addressing the problem of DCB-enabled switches that do not support any current queues. In the rate-based scheme, each final host must use the transfer rate allocated by the controller to send packets. In the credit-based scheme, the final hosts must receive credits from the controller before sending the data. This design has shown that no packet will be lost due to buffer overflow. In the congestion control scheme proposed by Shen et al.[9], traffic allocation first becomes a multi-part problem. Then, the artificial fish swarm algorithm is used to solve this problem.

Tajiki et al.[10], using a meta-innovative method called CECT, pre-calculate some practical paths and assign a different route to each flow. Hence, a set of different methods is created, and each method is ranked based on some limitations. CECT utilizes the roulette wheel selection algorithm to select some methods for the next-generation ancestors. The new population is created using uniform crossovers and multi-point mutations on these ancestors. This process runs generation until a violation method is detected or the predefined threshold is reached for repetition. Compared to the ECMP method, which is considered a comparison, CECT improves network throughput nearly 3 times and packet loss is reduced about 2 times compared to similar methods. Zhao et al.[11] introduced an intelligent congestion control algorithm supporting SDN architecture. This method uses fuzzy logic to assess the instantaneous congestion of alternative paths using the link load parameters of the network. In addition, the reinforcement learning-based redirection algorithm evaluates the possible efficiency of the diversion options and the best transportation route selection meeting the traffic load requirement for flow transmission. Lei et al.[12] proposed a multitasking DRL-based congestion control model in their paper. This learning model controls

congestion and balances the load of the network. Balancing the load leads to better congestion control.

Chen et al.[13] uses the Traffic Aware Load Balancing (TALB) mechanism for machine-to-machine networks using instant traffic detection and dynamic diversion features in SDN, which reduces response time by up to 50%. Chiang et al.[14] proposed a Dynamic Weighted Random Selection (DWRS) method. DWRS periodically calculates server loads and recalculates server weights dynamically. The server with the highest weight is selected as the destination server, whereas other servers can send the requests. This method intercepts load imbalance due to improper weight adjustments. Zhang et al.[15] also described a Reinforcement Learning (RL)-based scheme that automatically has a mechanism to learn for selecting critical flows in the input traffic. The CFR-RL then redirects the chosen flow by balancing and designing a simple linear programming (LP) problem to balance network connection usage. Performance evaluation of this method shows that CFR-RL has an optimal performance by redirecting about 10% -21.3% of total traffic. Yankam et al.[16] proposed a congestion management scheme to manage the load of the servers and to take into account Quality of Service (QoS). This scheme utilizes three algorithms to balance the load: a task selection algorithm, a server selection algorithm, and an automated task transfer algorithm. The algorithms find the resources in other domains where congestion is located. The scheme reduces the congestion load of the servers by up to 22% compared to similar works.

Gustavo et al. [17] proposed a Reinforcement Learning and SDN-aided Congestion Avoidance Tool (RSCAT). The authors classify data to avoid congestion in the network and use an actor–critic reinforcement learning method to get better parameters for Transmission Control Protocol (TCP). If the network is congested, the learning method extracts suitable configurations for TCP packets. The proposed tool reduces flow completion time and runtime. Prajapati et al.[18] propose a switch migration approach based on a greedy optimization algorithm for load balancing among controllers. If the mean square deviation of the load of each controller from the average load of all controllers is greater than a predefined threshold, OptiGSM initiates the switch migration process from an overloaded controller to an underloaded controller. This method has improved the number of switch migrations and response time compared to the CAMD and SMCLBRT methods, but it has reduced the network throughput.

The primary objective of Darmani et al in [19] is to introduce a novel framework, termed QDFSN, aimed at enhancing congestion control in Software-Defined Networks. This framework leverages Active Queue Management (AQM) and an advanced mathematical model to adaptively compute network parameters, such as service rate, thereby reducing congestion and improving TCP performance. However, due to the computational complexity of the proposed mathematical model, its application to large-scale networks with a high number of nodes becomes intricate and time-consuming.

Ghorbannia et al. proposed an enhanced method called ESV-DBRA for proportional traffic distribution in Multi-tenant Software-Defined Networks in [20]. This approach monitors traffic and latency across network paths, calculates the cost of each path, and directs each tenant's traffic through paths with the lowest cost. However, this method relies on a centralized controller, which may introduce a single point of failure. Additionally, dynamically computing path costs and allocating bandwidth can become complex in large-scale networks.

Although above studies have attempted to give various solutions for congestion issue, they have not been able to manage the load exchange between the control plane and the data plane effectively. Table I presents a comparison of several conducted studies.

3. System model and problem formulation

This section introduces the system model and problem formulation, with key notations summarized in Table II.

Table II. Notations.

Notation	Description
G	Multidomain SDN network
U	Set of Controllers
V	Set of Switches
P_i	Number of packet-in messages from switch i
d_i	Propagation delay time
q_i	Packet-in process time
r_i	Rule installs time
LC_j	The load of controller j
LC_jS_i	The load of controller j from switch i
LS_{MS}	The switch nominated for migration
$LCpu_j$	CPU usage in controller j
$LMem_j$	Memory usage in controller j
γ_{max}	Maximum controller load threshold
γ_{min}	Minimum controller load threshold
σ	Threshold ratio
$\delta_{C_iC_j}$	Load rate between controller i and controller j
LS_{ij}	The load of switch i in the domain of controller j
F_i	The number of entry flows to switch i
t_{ci}	The compare time (compare between entry flow and flow table rules)
t_{ti}	Time for sending the packet to the destination node
t_{mmi}	Time for sending packet-in message from switch i to its master controller
t_{Ri}	Time to install a rule to flow table
t_{qi}	Time of wait in entry flow queue
S_M	Set of migration candidate switches
C_s	Set of migration source controllers
C_d	Set of migration destination controllers
$x_i^{jj'}$	Is 1 when switch i has migrated from controller j to controller j' and otherwise is 0
μ_i	The processing rate of controllers

λ_i	The expected rate of arrival of the flow to the switch
ϑ	The variance load of all controllers
$\widehat{LC_j}$	The controller's load after migration

3.1. System model

The network is modeled as G(U,V), where U is the set of controllers and V is the set of switches The maximum numbers of controllers and switches are N and M:

$$U = \{C_1, C_2, \dots, C_N\}, V = \{S_1, S_2, \dots, S_M\} \quad (1)$$

and each switch connected to only one master controller:

$$V_j = \forall S_i \in V | S_i \subset C_j, \bigcup_{j=1}^{N} V_j = V \ , \ \bigcap_{j=1}^{N} V_j = \ \emptyset, i = 1,2, \dots, M, j = 1,2, \dots, N \quad (2)$$

Fig. 2 illustrates this structure. Each controller acts as the master for switches in its domain and as a slave for others. S_{ij} refers to the ith switch of the jth controller domain. Each switch maintains a flow table managed by its master controller. Incoming flows are matched against flow table rules; if no match is found, the switch sends a packet-in to the controller, which responds with a packet-out to install a new rule.

The load on the master controller is influenced not only by the switches in its domain (denoted as LC_jS_i) but also by the controller's CPU and memory status.

Table I. Comparison of several conducted

Author(s)	Year	Main Challenge	Method / Algorithm	Advantages	Disadvantages
Dawei Shen et al.	2018	Congestion control and traffic scheduling in mobile SDN networks	AFSA	Load balancing, reduced congestion, improved throughput	High computational complexity, full topology info required
Kshira Sagar Sahoo & Bibhudatta Sahoo	2019	Controller load balancing via switch migration	CAMD	Reduced response time, optimal controller selection	Requires inter-controller coordination, complex selection logic
Junjie Zhang et al.	2020	Traffic engineering with minimal disruption	CFR-RL (Reinforcement Learning)	Adaptive learning, optimized link utilization	Requires training, uses LP for each cycle
Mei-Ling Chiang et al.	2020	Server-side load balancing in SDN clusters	DWRS + Multi-threaded	Real-time load adaptation, better than RR	Needs multithreading, internal Floodlight mods
Upendra Prajapati et al.	2023	Minimal switch migration for controller load balancing	OptiGSM (Greedy-based)	Fewer migrations, higher throughput, lower delay	Complexity in large-scale, needs deviation calc
Yannick Florian Yankam et al.	2024	Congestion control using work stealing in SDN	WoS-CoMS	Up to 22% load reduction, faster task transfer	Complex destination server management, strategy sensitivity

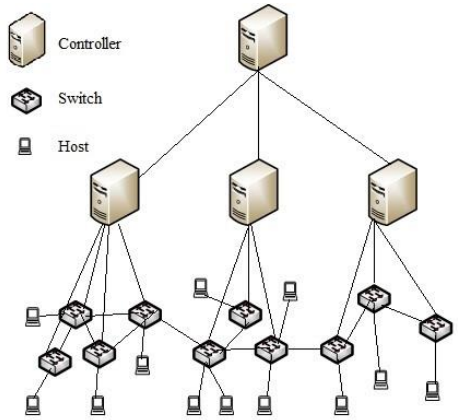


Fig.2 The structure of the proposed model.

3.2. Formulation

The load imposed by switch i on controller j is:

$$LC_jS_i = P_i\ (\ d_i + \ q_i + \ r_i\ ) \quad (3)$$

Where P_i is the number of packet-in sent from switch i to controller j, and d_i, q_i and r_i are propagation delay, packet processing time, and rule installation time, respectively. The total controller load is:

$$LC_j = \sum_{i=1}^{k} LC_jS_i + LCpu_j + LMem_j \quad (4)$$

Post-migration load ($\widehat{LC_j}$) is:

$$\widehat{LC_j} = LC_j + (\sum_{j'\in U|j\neq j',} \sum_{i\in V} \left(LC_{j'}S_i \times \ {x_i}^{j'j} - LC_jS_i \times \ {x_i}^{jj'}\right)) \quad (5)$$

Where ${x_i}^{jj'}$ is 1 if switch iii migrates from controller j to j' and 0 otherwise.

Migration cost is the variance of controller loads:

$$\vartheta = \frac{1}{n}\ \sum_{j=1}^{n} (\widehat{LC_j} - \ \overline{LC_j}) \quad (6)$$

The objective is to minimize migration cost (an ILP problem):

$$Min(\frac{1}{n}\ \sum_{j=1}^{n} (\widehat{LC_j} - \ \overline{LC_j})) \quad (7)$$

Subject to:

$$\sum_{j\in U} \sum_{j'\in U} x_i^{jj'} \ \leq 1 \qquad \forall i \in V \quad (8)$$

$$0 \leq \ \widehat{LC_j} \ \leq \gamma_{max} \quad (9)$$

$$x_i^{jj'} \ \in \{0,1\} \qquad \forall\ j,j' \in U\ ,\ i \in V \quad (10)$$

$$LCpu_j < 50\% \quad (11)$$

$$LMem_j < 50\% \quad (12)$$

Constraint (8) ensures that each switch is controlled by only one controller. Inequality (9) establishes an upper bound on the maximum controller load following switch migration. Equation (10) indicates that $x_i^{jj'}$ is a binary variable, while Inequalities (11) and (12) ensure that CPU and memory usage do not exceed 50% of their respective capacities. For small networks, this is solvable; for larger ones, a heuristic is proposed.

4. Proposed Method

This paper proposes a migration-based congestion control and prevention framework, as depicted in Fig. 3. The framework dynamically monitors controller congestion across domains and optimizes switch migration through six modules: topology discovery, traffic monitoring, switch and controller load calculation, congestion detection, CCSM[1], and migration execution.

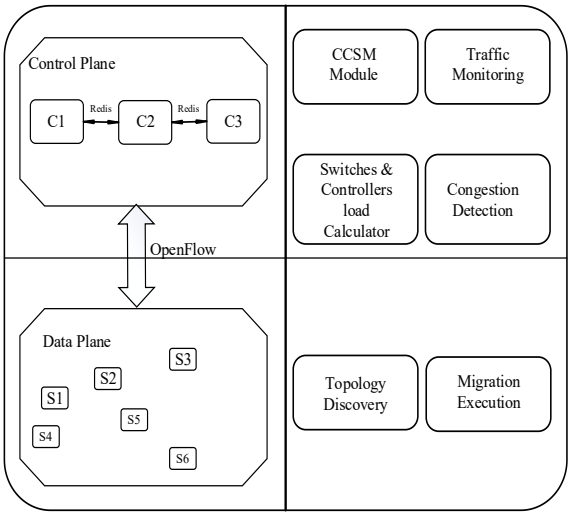


Fig.3 The framework of the proposed method.

- Topology discovery module: Provides an overview of the multi-domain network, supplying data for other modules' calculations.
- Traffic monitoring module: Periodically collects statistics on controllers and switches, feeding data to the load calculation module for congestion assessment.
- Switch and controller load calculator module: Computes switch and controller loads at set intervals based on monitoring data, forwarding results to the congestion detection module.
- Congestion detection module: Assesses controller congestion by comparing loads to predefined maximum and minimum thresholds, triggering the congestion control method if needed .

[1] Congestion Control with Switch Migration

- CCSM module: Manages controller loads to prevent congestion, selecting switches for migration and coordinating with the migration execution module.
- Migration execution module: Executes migrations by reassigning switches from congested to underloaded controllers, updating their master controller roles.

The Congestion Control via Switch Migration (CCSM) mechanism is presented in three steps: load calculation, congestion detection, and migration execution.

4.1. Calculating Switch Load

For each incoming packet, a switch compares it to its flow table rules. If matched, the packet is forwarded; otherwise, it is sent as a packet-in to the controller, which installs a rule via packet-out. Each of the above steps contributes to the load imposed on the switch, enabling the definition of the following equation for switch load:

$$LS_{ij} = \sum_{i=1}^{M_j} (PI_i \times (t_{ci} + t_{ti} + t_{mmi} + t_{Ri} + t_{qi})) \quad (13)$$

Where LS_{ij} is the load imposed on switch i within the domain of controller j, M_j is the number of switches in the domain of controller j, PI_i is the number of packets entering the switch, t_{ci} is the time spent comparing the packet with the flow tablet_{ti} is the time required to send the packet to the destination, t_{mmi} is the time required to send the packet to the master controller, t_{Ri} is the time required to install a rule in the switch, and t_{qi} is the waiting time in the entry flow queue. In equation (13), depending on whether the input flow matches the rules in the switch's flow table, either t_{ti} or t_{mmi} is zero.

The input flow to switch i follows a Poisson distribution with parameter λ_i, which represents the expected arrival rate of flows to the switch. Upon the flow's arrival at the switch, comparison with the flow table begins immediately at a processing rate of μ_i.

Using the M/M/1 queue model to describe flow processing in the switch, , $\frac{\lambda_i}{\mu_i}$ indicates the percentage of time the switch is busy, and $\frac{1}{\mu_i}$ represents the processing time of a flow in switch i. Therefore, the average waiting time of a flow in the switch can be calculated using the following equation [21]:

$$t_{qi} = \frac{\lambda_i}{\mu_i(\mu_i - \lambda_i)} \quad (14)$$

Thus, the load imposed on each switch within the domain of each controller can be calculated (Algorithm 1), allowing the selection of the switch or switches nominated for migration. First, the load of each switch i under controller j is computed using equation (13). Then, this set is sorted in descending order.

Algorithm 1. Switch Load calculation

Input: G(U,V)
Output: Sorted $\{S_{Mi}\}$
1: Begin
2: For C_j in U
3: For $S_i \in C_j$
4: Calculate LS_{ij} using Eq No. (13)
5: End for
6: $\{S_{Mi}\}$= Sort Descending LS_{ij}
7: End for
8: End

4.2. Congestion detection mechanism

Various papers have identified different parameters as influential factors affecting controller load, with most recognizing the number of packet-in messages as the primary parameter [22], [23].

When a new flow enters a switch and no corresponding rule exists, the switch sends it to the controller for determination (packet-in). The load imposed by the switch on the network can be divided into two components: first, the load directly imposed on the controller by sending packet-in messages (equation (3)); and second, the switch load resulting from input flows and the traffic generated within the domain supervised by the controller (equation (13)).

The congestion detection and migration set selection algorithms are presented in Algorithm 2 . In this algorithm, the load of each controller j is calculated using equation (4). If a controller's load exceeds the maximum threshold (γ_{max}), it is added to the set of controllers designated for migration (Cs). From this set (under the management of this controller), one or more switches must be selected for migration. Conversely, if a controller's load falls below the minimum threshold (γ_{min}), it is added to the set of destination controllers for migration (Cd), the candidate switches for migration are migrated to this controller. If no such controller exists in the network, the controller with the lowest load is designated as the migration destination.

If the set Cd is empty, the load ratio between controllers ($\delta_{C_iC_j}$) is calculated according to equation (15) and compared with the threshold σ. If any ratio exceeds this threshold, the controllers corresponding to the numerator and denominator of the ratio are added to the source Cs,and destination Cd sets for migration, respectively. If no controllers are identified for the set Cs , the network is considered balanced, and switch migration is unnecessary, as the load of all controllers remains below the maximum threshold (γ_{max}). The sets Cs and Cd are returned as the algorithm's outputs.

$$\delta_{C_iC_j} = \frac{LC_i}{LC_j} \qquad i,j = 1,2,\dots,N \quad (15)$$

Algorithm 2. Congestion detection and migration set selection

Input: $G(U,V)$
Output: $\{Cs\},\{Cd\}$
1: Begin
2: For $j = 1$ to N
3: $Calculate\ LCj\ using\ Eq.(10)$
4: If $LC_j > \gamma_{max}$ then
5: $Add\ Cj\ to\ \{Cs\}$

6: Elseif $LC_j < \gamma_{min}$ then
7: $Add\ Cj\ to\ \{Cd\}$
8: End For
9: If {Cd}==NULL
10: For $i, j = 1\ to\ N$
11: $Calculate\ \delta_{C_iC_j}\ using\ Eq.(15)$
12: If $\delta_{C_iC_j} > \sigma$ then
13: $Add\ C_i$ to $\{C_s\}$
14: $Add\ C_j$ to $\{C_d\}$
15: End if
16: End for
17: End if
18: End

4.3. Switch migration mechanism

After executing Algorithm 2 and identifying the controllers and switches nominated for migration, source triples are created: source controller Cs , migrant switch SM , and destination controller Cd , denoted as <Cs,SM,Cd>. The objective here is to determine whether migration should occur, based on the load that the switch in question reduces from the source controller and adds to the destination controller. This process is repeated until the source controller is no longer congested and the destination controller remains uncongested.

$$LS_{MS} = P_M\ (\ d_M + q_M + r_M\ ) \quad (16)$$

$$LC_s(t) = LC_s(t-1) - LS_{MS} \quad (17)$$

$$LC_d(t) = LC_d(t-1) + LS_{MS} \quad (18)$$

where LS_{MS} represents the load of the switch nominated for migration within the domain of the congested controller, subject to the condition that the load of the controllers after migration remains below the upper load threshold:

$$\forall\ LC_i : LC_s(t)\ ,\ LC_d(t) < \gamma_{max} \quad (19)$$

The switch migration problem is NP-hard [24]. Based on the migration conditions, the proposed CCSM method is executed in two steps: determining the switches nominated for migration and performing the migration, as detailed below (Algorithm 3).

Step 1: Selecting the Switches Nominated for Migration

After executing Algorithm 1 and identifying the controller or controllers experiencing congestion {Cs}, this step involves selecting the switch or switches nominated for migration from those within the domain supervised by the controllers in the set {Cs}.

Step 2: Performing Migration

Considering the outputs of the previous two steps and provided that {Cs} is not empty, migration is performed in this step based on the controller and switch loads, using equations (16) to (18) and the condition in equation (19). If the load of S_{Mi}, after being removed from controller Cs and added to the domain of controller Cd , reduces the load of Cs below the threshold γmax \gamma_{max} γmax while keeping the load of Cd below γ_{max}, migration proceeds. However, if either condition is not satisfied, the switches are adjusted within the sorted set S_M using the modified Kadane algorithm, and the steps are repeated.

Algorithm 3. Heuristic network congestion control **.**

Input: $G(U,V)$, $\{S_{Mi}\}$ from algorithm 1, $\{C_s\}$, $\{C_d\}$ from algorithm 2
Output: Optimized $G(U,V)$
1: Begin
2: For $S\ in\ \{S_{Mi}\}$ do
3: For C_j in $\{C_s\}$ do
4: For $C_{j'}$ in $\{C_d\}$ do
5: Calculate $LC_j(t)$, $LC_{j'}(t)$ by Eqs (17), (18)
6: If $LC_j(t)$ and $LC_{j'}(t) < \gamma_{max}$ then
7: Migrate S from C_j to $C_{j'}$
8: Remove S from $\{S_{Mi}\}$
9: Continue for next C_j
10: Else If $LC_j(t) > \gamma_{max}\ and\ LC_{j'}(t) < \gamma_{max}$ then
11: Migrate S from C_j to $C_{j'}$
12: Remove S from $\{S_{Mi}\}$
13: Continue for next S
14: Else
15: Select switches to migrate using the modified Kadane algorithm
16: End If
17: End For
18: End For
19: End For
20:End

5. Performance evaluation

In this research, the Linux Ubuntu 20.0 operating system installed on a VMware virtual machine with 16 GB of RAM and an Intel(R) Core(TM) i7-4750HQ @ 2.00GHz processor was used to simulate the proposed mechanism.

5.1. Simulation model and environment

As shown in Fig. 3, the network consists of 3 domains that 3 controllers manage, and a supervisor controller is responsible for monitoring the performance of the whole network. There are also 10 switches and 12 hosts in the network that are distributed among the controllers.

In this work, the Mininet simulator was used to simulate the experimental environment. The controllers used in this model are RYU controllers. In Mininet, each is assigned a port, as shown in Fig. 4, and they are launched as depicted in Fig. 5. Communication between the controllers is facilitated by Redis, an in-memory messaging system. Each controller serves as the Master for the switches assigned to it in Mininet and as the Slave for all other switches. Communication between switches and controllers is established via the OpenFlow 1.3 protocol.

D-ITG and iPerf are also used to generate consistent traffic between hosts. [25], [26].

In this section, the throughput, delay, jitter, and RTT parameters are used to evaluate the proposed algorithm. γ_{max}, γ_{min}, and σ are equal to 130, 80, and 1.6 respectively. The evaluations are done in 30 and 60 seconds intervals.

First, a desired topology, as shown in Fig. 4, is defined using Mininet commands. Then, three main network controllers and one supervisor controller are launched in four separate terminals, and the topology is executed in Mininet (Figs. 5 and 6).

```
def myNetwork():

    net = Mininet( topo=None,
                   build=False,
                   ipBase='10.10.10.0/24')

    info( '*** Adding controller\n' )

    c0=net.addController(name='c0',
                         controller=RemoteController,
                         ip='127.0.0.1',
                         protocol='tcp',
                         port=6653)

    c1=net.addController(name='c1',
                         controller=RemoteController,
                         ip='127.0.0.1',
                         protocol='tcp',
                         port=6654)

    c2=net.addController(name='c2',
                         controller=RemoteController,
                         ip='127.0.0.1',
                         protocol='tcp',
                         port=6655)
```

Fig. 4 Sample code for creating topology in Mininet.

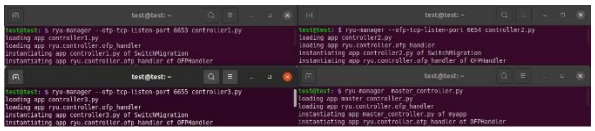


Fig. 5 Launching the controllers.

```
test@test: ~$ sudo python topo1.py
[sudo] password for test:
*** Adding controller
*** Add switches
*** Add hosts
*** Add links
*** Starting network
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12
*** Starting controllers
*** Starting switches
*** Starting CLI:
mininet>
```

Fig. 6 Running the topology.

Evaluations are conducted first without congestion control (Without_CCMS) and then with the execution of the proposed algorithm (With_CCMS)

5.2. Simulation results

As shown in Figs. 7 and 8, the controller loads are measured under two conditions: without congestion control and with congestion control. In Fig. 7, the load remains uncontrolled, whereas in Fig. 8, the controller loads are maintained below the threshold after approximately 10 seconds, when the supervisor controller begins its initial statistics collection.

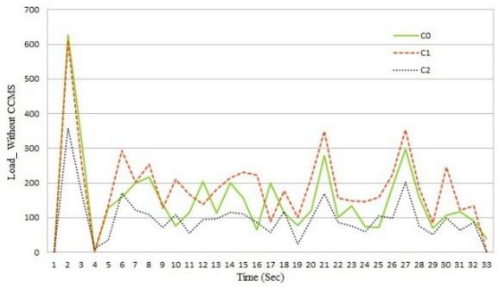


Fig.7 Load of controllers without executing the proposed algorithm of congestion control.

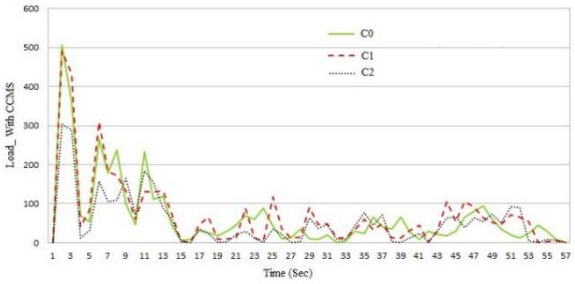


Fig.8 Load of controllers executing the proposed algorithm of congestion control.

Figs. 9, 10, and 11 illustrate the throughput of the controllers. The throughput when using the proposed algorithm is, on average, 10% higher than when the algorithm is not used, demonstrating its effectiveness.

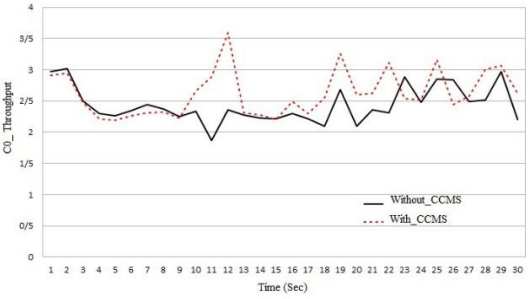


Fig.9 Thoughput of controllers executing the proposed algorithm of congestion control.

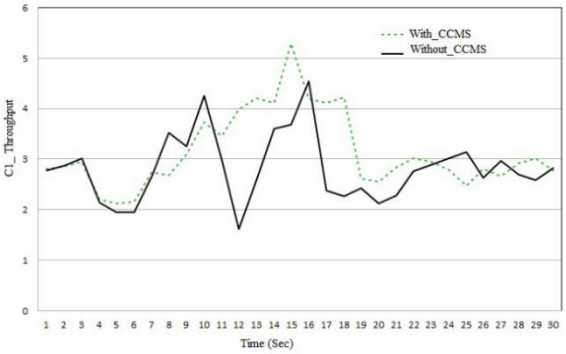


Fig.10 Thoughput of controllers executing the proposed algorithm of congestion control.

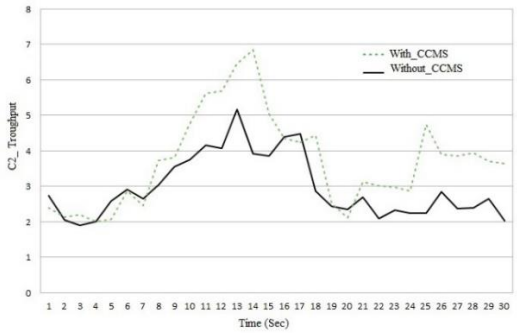
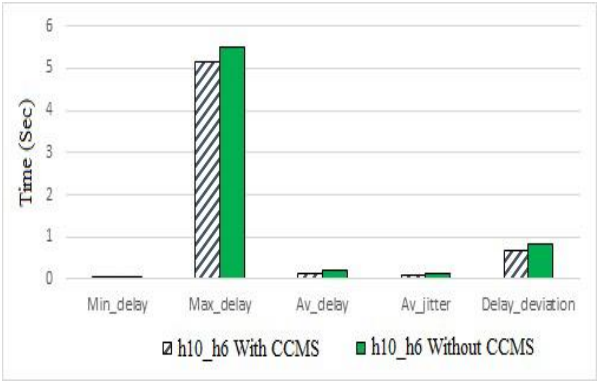
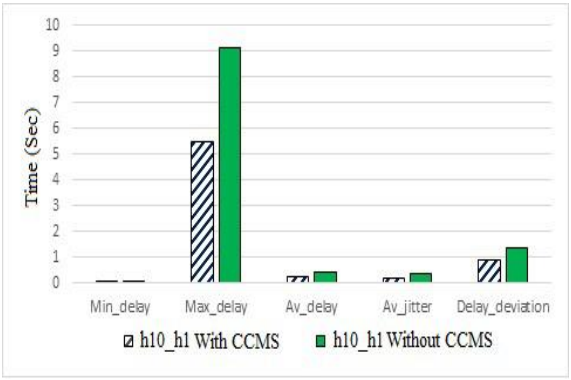


Fig.11 Thoughput of controllers executing the proposed algorithm of congestion control.

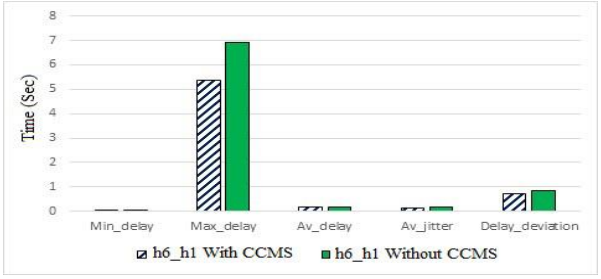
Fig. 12 examines the delay and jitter values between hosts across different domains. When the proposed congestion control algorithm is applied, delay and jitter are reduced by approximately 30% and 25% on average, respectively, compared to the scenario without the algorithm.



(a)



(b)



(c)

Fig. 12: Comparison of delay and jitter between hosts in different domains in two cases: with and without executing the proposed algorithm.

Finally, Table III compares the Round-Trip Time (RTT) between hosts. h1, h2, and h4 reside in one domain, while h6 and h8 are in another. Consequently, the RTT between hosts within the same domain is lower than that between hosts in different domains. Additionally, due to the time-intensive computations and switch migration actions required to prevent congestion, the RTT when using the proposed algorithm is higher than during normal system operation. However, a comparison of CPU usage percentages indicates that the proposed method results in approximately a 10% reduction in CPU consumption.

Table III - Comparison of RTT between hosts in different domains in two cases with and without executing the proposed algorithm.

RTT	min	ave	max	mdev	CPU Usage
h1-h2_CC	0.118	2.272	67.843	13.807	42%
h1-h2	0.153	1.157	48.923	6.695	50%
h1-h4_CC	0.142	2.24	74.807	12.098	38%
h1-h4	0.101	1.216	53.279	6.342	45%
h1-h6_CC	0.177	4.168	118.777	22.994	54%
h1-h6	0.159	2.791	92.211	14.721	65%
h1-h8_CC	0.179	2.202	97.566	15.184	37%
h1-h8	0.141	1.788	71.996	9.321	42%
h4-h8_CC	0.176	3.067	96.487	16.033	53%
h4-h8	0.177	2.737	91.465	14.793	59%

Additionally, to compare the proposed method with existing approaches, paper [18], which implements migration based on the standard deviation of controller load, employs a greedy algorithm, and introduces the OptiGSM method, was selected. The OptiGSM method exhibits a time complexity of O(m·n), whereas the proposed method has a time complexity of O(m+n), where n and m represent the number of controllers and switches, respectively.

As illustrated in Fig.13, the average throughput of the proposed method demonstrates an improvement of approximately 15% compared to the OptiGSM method. Fig.14 presents a comparison of delay and jitter between the two aforementioned methods. Owing to its lower time complexity, the proposed method exhibits reduced delay; however, the OptiGSM method, which employs a metaheuristic algorithm, achieves lower jitter.

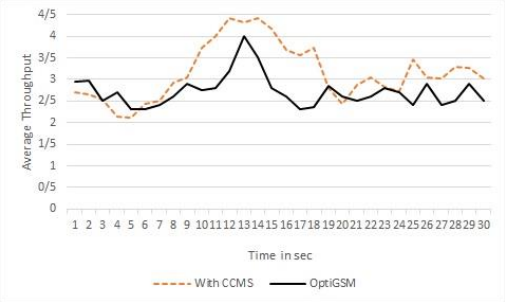


Fig. 13: Comparison of average throughput.

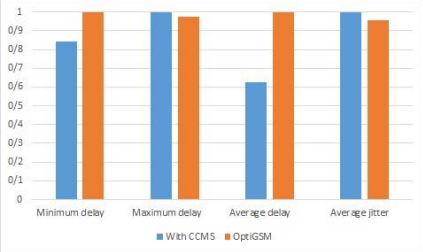


Fig. 14: Comparison of delay and jitter.

To assess the preservation of the time complexity of the proposed method, the network scale was expanded to include 5 controllers, 14 switches, and 22 hosts, and the two aforementioned methods were re-evaluated. The results, as depicted in Figure 15, demonstrate that the proposed method maintains its complexity despite the increased network scale.

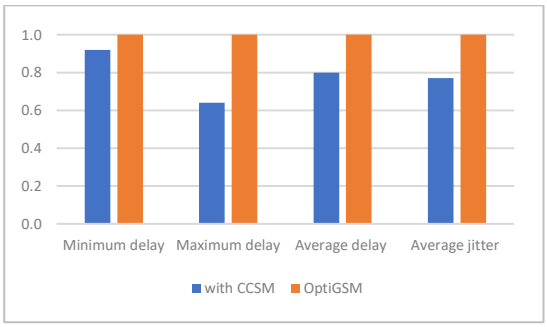


Fig. 15: Comparison of delay and jitter in the network with 5 controllers.

6. Conclusion

This paper presents a switch migration mechanism designed to prevent controller congestion through the optimal selection of one or more switches for migration. In this mechanism, the supervisor controller periodically measures the loads of the controllers and switches in the network, sorting them by load. If a controller's load exceeds the threshold, the switch with the highest load is selected for migration to the controller with the lowest load. Before migration, the load of the destination controller is calculated assuming the switch is migrated. If the resulting load remains below the threshold, migration proceeds; otherwise, switches are selected for relocation and migration between controllers using an optimized Kadane algorithm. Evaluations indicate that throughput, delay, and jitter improve by 10%, 30%, and 25%, respectively, although RTT increases by approximately 20% due to the time-consuming migration processes of the CCSM algorithm. Furthermore, a comparison between the proposed method and the OptiGSM method reveals that the proposed method offers superior throughput and lower delay, although the OptiGSM method exhibits less jitter than the proposed method. For the future scope of the work, this framework can be implemented and evaluated for IoT environment as well as task scheduling and load balancing in SDN-based cloud computing.

7. References:

[1] Parsaei, M. Reza, R. Mohammadi, and R. Javidan. "A new adaptive traffic engineering method for telesurgery using ACO algorithm over software defined networks."*European Research in Telemedicine/La Recherche Europeenne en Telemedecine,* vol. 6, no. 3-4, pp. 173-180, 2017.

[2] S. Rowshanrad, V. Abdi, M. Keshtgari, "Performance evaluation of sdn controllers: Floodlight and opendaylight." *IIUM Engineering Journal* , vol. 17, no. 2, pp. 47–57, 2016.

[3] C.Y. Chu, K. Xi, M. Luo, H.J. Chao, "Congestion-aware single link failure recovery in hybrid SDN networks." In 2015 IEEE Conference on Computer Communications (INFOCOM) (IEEE, 2015), pp. 1086–1094.

[4] S. Song, J. Lee, K. Son, H. Jung, J. Lee, "A congestion avoidance algorithm in SDN environment." In 2016 International Conference on Information Networking (ICOIN) (IEEE, 2016), pp. 420–423.

[5] T. Zhu, D. Feng, F. Wang, Y. Hua, Q. Shi, Y. Xie, Y. Wan, "A congestion-aware and robust multicast protocol in sdn-based data center networks." *Journal of Network and Computer Applications* 95, 105–117 (2017).

[6] Hu, Y., Peng, T., & Zhang, L. (2017). "Software-Defined Congestion Control Algorithm for IP Networks." *Scientific Programming,* 2017(1), 3579540.

[7] M. Rahman, N. Yaakob, A. Amir, R. Ahmad, S. Yoon, A. Abd Halim, "Performance analysis of congestion control mechanism in software defined network (SDN)," In MATEC Web of Conferences, vol. 140 (EDP Sciences, 2017), p. 01033.

[8] S.Y. Wang, L.M. Chen, S.K. Lin, L.C. Tseng," Using SDN congestion controls to ensure zero packet loss in storage area networks," In 2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM) (IEEE, 2017), pp. 490–496.

[9] D. Shen, W. Yan, Y. Peng, Y. Fu, Q. Deng, "Congestion control and traffic scheduling for collaborative crowdsourcing in sdn enabled mobile wireless networks." *Wireless Communications and Mobile Computing* 2018, 1–11 (2018).

[10] M.M. Tajiki, B. Akbari, M. Shojafar, S.H. Ghasemi, M.L. Barazandeh, N. Mokari, L. Chiaraviglio, M. Zink, "Cect: Computationally efficient congestion-avoidance and traffic engineering in software-defined cloud data centers." *Cluster Computing* 21, 1881–1897 (2018).

[11] J. Zhao, M. Tong, H. Qu, J. Zhao, "An Intelligent Congestion Control Method in Software Defined Networks," In 2019 IEEE 11th International Conference on Communication Software and Networks (ICCSN) (IEEE, 2019), pp. 51–56.
[12] K. Lei, Y. Liang, W. Li, "Congestion control in sdn-based networks via multi-task deep reinforcement learning." *IEEE Network* 34(4), 28–34 (2020).
[13] Y.J. Chen, L.C. Wang, M.C. Chen, P.M. Huang, P.J. Chung, "Sdn-enabled traffic-aware load balancing for m2m networks." *IEEE Internet of Things Journal* 5(3), 1797–1806 (2018).
[14] M.L. Chiang, H.S. Cheng, H.Y. Liu, C.Y. Chiang, "Sdn-based server clusters with dynamic load balancing and performance improvement." *Cluster Computing* 24, 537–558 (2021).
[15] J. Zhang, M. Ye, Z. Guo, C.Y. Yen, H.J. Chao, "CFR-RL: Traffic engineering with reinforcement learning in sdn." *IEEE Journal on Selected Areas in Communications* 38(10), 2249–2259 (2020).
[16] Y.F. Yankam, V.K. Tchendji, J.F. Myoupo, "Wos-coms: Work stealing-based congestion management scheme for sdn programmable networks." *Journal of Network and Systems Management* 32(1), 23 (2024).
[17] G. Diel, C.C. Miers, M.A. Pillon, G.P. Koslovski, "Rscat: Towards zero touch congestion control based on actor–critic reinforcement learning and software-defined networking." *Journal of Network and Computer Applications* 215, 103639 (2023).
[18] U. Prajapati, B.C. Chatterjee, A. Banerjee, " Optigsm: Greedy-based load balancing with minimum switch migrations in software-defined networks." *IEEE Transactions on Network and Service Management* 21, 2200-2210 (2023).
[19] Y. Darmani, M. Sangelaji. "QDFSN: QoS-enabled Dynamic and Programmable Framework for SDN." *Tabriz Journal of Electrical Engineering* 51, no. 1 , 1-10 (2021)
[20] A.Ghorbannia Delavar, K. Beigi. "ESV-DBRA: An enhanced method for proportional distribution of the multitenant SDN traffic load." *Tabriz Journal of Electrical Engineering* 52, no. 4, 269-280 (2022).
[21] B. Xiong, X. Peng, J. Zhao, "A concise queuing model for controller performance in software-defined networks". *J. Comput.* 11(3), 232–237 (2016).
[22] U. Srisamarn, L. Pradittasnee, N. Kitsuwan, "Resolving load imbalance state for sdn by minimizing maximum load of controllers." *Journal of Network and Systems Management* 29(4), 46 (2021).
[23] O. Adekoya, A. Aneiba, M. Patwary, "An improved switch migration decision algorithm for sdn load balancing." *IEEE Open Journal of the Communications Society* 1, 1602–1613 (2020).
[24] Y. Zhou, K. Zheng, W. Ni, R.P. Liu, "Elastic switch migration for control plane load balancing in sdn." *IEEE Access* 6, 3909–3919 (2018).
[25] M.T. Islam, N. Islam, M.A. Refat, "Node to node performance evaluation through ryu sdn controller." *Wireless Personal Communications* 112, 555–570 (2020).
[26] S. Bhardwaj, S.N. Panda, " Performance evaluation using ryu sdn controller in software-defined networking environment." *Wireless Personal Communications* 122(1), 701–723 (2022).