Research

Optimal controller selection and migration in large scale software defined networks for next generation internet of things

Mohammad Shahzad[1] · Lu Liu[1] · Nacer Belkout[2] · Nick Antonopoulos[3]

Received: 30 July 2023 / Accepted: 17 October 2023
Published online: 04 November 2023

Abstract
The substantial amount of IoT traffic, coupled with control messages, places a heavy burden on SDN controllers, which compromises their capacity. We investigate how SDN can revolutionize the conventional approach, aiming to overcome the limitations of communication overhead. Additionally, we delve into the essential optimizations required to minimize control overhead and migrations. Determining the appropriate controller necessitates the implementation of a mechanism that justifies the selection. Once the optimal controller has been identified, migration can be initiated. This paper introduces a solution that employs the NSGA-II algorithm to achieve the optimal selection of controllers. We assess the performance of the NSGA-II migration approach linking with the length-based same destination aggregation proposed in our previous work, in terms of packet delivery, packet loss, performance metrics, and the cost associated with the selected optimal controller.

Article highlights

- The article selects optimal controller based on performance and cost calculated mathematically reducing control overhead.
- NSGA-II is employed to select controller to reduce migrations and control communication.
- NSGA-II is linked with our previous work burst assembly and compared with 4 state-of-the-art methods.

Keywords Internet of things · Optimal controller selection · Software defined networking (SDN) · SDN migration

1 Introduction

Software Defined Networking (SDN) revolutionizes the conventional networking approach, which involves routing traffic through switches and routers to reach its destination. In SDN, incoming traffic at switches is processed using a Ternary Content Addressing Memory (TCAM) flow table to identify specific packets [1]. However, issues arise when the flow table in switches lacks certain entries. In such cases, switches rely on the controller to handle flow requests, which returns matching rules. As the number of clients increases, the requests sent to the controller escalate, leading to heightened communication overheads caused by the increased traffic. Consequently, the SDN

✉ Mohammad Shahzad, mns14@leicester.ac.uk | [1]School of Computing and Mathematical Sciences, University of Leicester, Leicester, UK. [2]Department of Computer Science, University of Science and Technology, Houari Boumediene, Algiers, Algeria. [3]Edinburgh Napier University, Edinburgh, UK.

controller is burdened as the growing number of clients or devices push its limits by generating higher communication overhead. IoT-centric applications such as smart cities [2], self-driving cars [3], telehealth, and numerous others have now become ubiquitous. Accompanying these applications are demanding requirements for throughput, latency, bandwidth, and data rates. Overcoming the challenge of overloaded controllers is a crucial issue that needs to be addressed. Among various factors, communication overhead plays a role, but migration to a controller capable of handling the increased load becomes necessary. However, this migration process is not without costs, and attempts have been made to minimize it when complete avoidance is not possible. Latency, controller capacity, and migration costs are critical factors to consider in real network scenarios.

This paper emphasizes the importance of SDN controller selection while emphasizing the need for efficient control communication management. It tackles the challenges associated with control communication and switch migration by implementing essential optimizations. To enhance the selection process of controllers by the switches, we employ NSGA-II and compare the outcomes with the existing state-of-the-art methods.

The paper is structured as follows: Sect. 2 provides literature review on controller selection and migration. Section 3 discusses the design and implementation aspects. Performance evaluation and results are presented in Sect. 4, and Sect. 5 concludes the paper.

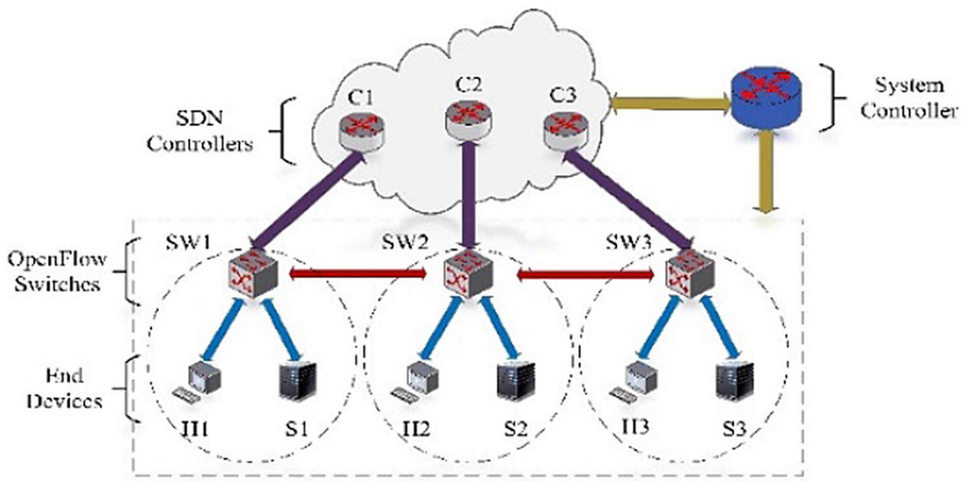
2 Related works

In this study, the widely used communication protocol "OpenFlow" is employed to access the forwarding plane. Authors in [4] discuss the issue of rule duplication during network updates in SDN and propose a strategy to minimize TCAM usage during updates. However, the network still experiences a burden of communication and struggles to alleviate the load on the controller, leading to migration and does not guarantee a low communication overhead. Controllers are only capable of managing the switch and performing configurations in their master state [5]. When controllers become overloaded and exceed their capacity threshold, they are unable to ensure accuracy through dynamic switch migration. Frequent switch migrations not only disrupt ongoing traffic but also come with their own costs. Achieving a balanced load on the controllers and utilizing their full potential necessitates the availability of traffic information and optimized selection. In this context, SDN controllers achieve load balancing through the implementation of the NSGA-II algorithm, which selects the optimal controller based on performance and cost considerations.

In [6], the authors focus on load balancing utilizing OpenFlow in conjunction with genetic algorithms. Their approach aims to reduce costs and alleviate bottlenecks caused by a single controller. By implementing efficient policies for load balancing and configuring appropriate entries, client data is effectively distributed to servers while ensuring each flow is directed accordingly. Work presented in [7] proposed an algorithm that addresses the load balancing challenge in a multi-controller network by considering capacity and latency constraints. The algorithm utilizes a clustering technique to evenly distribute the load across each controller, ensuring a balanced allocation of resources. In [8] and [9], researchers propose an algorithm for optimizing the deployment position of controllers. This algorithm evaluates the quality of paths and the efficiency of nodes by adjusting the weights of individual nodes using weight parameters. Based on the selected controller deployment location, a sub-domain algorithm is introduced in the SDN environment to enhance the k-center clustering method. In SDN IoT, effective communication between sensors and controllers is crucial. Increased flow requests can lead to performance degradation, highlighting the significance of optimal controller selection. [10] employs an analytical network decision-making process technique to choose the optimal controller based on its features. The controller features are compared with a high-weight controller, prioritizing features over performance. The proposed technique claims to reduce delay in both normal and heavy traffic scenarios. The chosen controller efficiently utilizes the CPU, resulting in increased throughput and reduced recovery latency.

A two-step approach is considered in selection of controller in [11] with analytical network process ranking the controllers. The qualitative features that affect the controller are used to rank it and performance is compared against QoS. The controller that exhibits a significant weight value in the feature-based comparison undergoes quantitative analysis through experimental evaluation. The primary focus of this paper lies in assessing the suitability of the Analytic Network Process (ANP) for the selection of controllers in SDN, considering both their features and real-world performance in Internet and Brite topologies. The simulation outcomes demonstrate that the controller chosen using our proposed method surpasses those selected through existing techniques. Opting for the ANP-driven optimal controller leads to decreased topology discovery time and reduced delays in scenarios involving both normal and heavy traffic loads. Additionally, there is an observed increase in throughput while maintaining a reasonable CPU utilization for the controller suggested in the approach.

Fig. 1 System architecture



The controller plays a vital role in the context of SDN. These controllers possess a range of features that allow them to monitor the network and respond quickly to its ever-changing dynamics. The performance of these controllers has a direct impact on the QoS delivered within an SDN environment. Each controller comes with its unique set of features. However, the prominence of specific features can vary among different controllers. Additionally, relying on a single controller introduces performance bottlenecks, single points of failure (SPOF), and scalability challenges. To address these issues, it becomes essential to have access to an SDN controller with an optimal feature set. Furthermore, creating a cluster of controllers with optimal feature sets can eliminate SPOF concerns and enhance QoS in SDN. In [12], it leverages the analytical network process (ANP) to assess and rank SDN controllers based on their feature support. Subsequently, it creates a hierarchical control plane-based cluster (HCPC) comprising the highest-ranked controllers as determined by the ANP. The performance of this cluster is evaluated in the context of the OS3E network topology. The results obtained through experiments conducted in the Mininet environment illustrate that the HCPC configuration, featuring an optimal controller, leads to improved QoS. Furthermore, in experimental findings, verified within Mininet, demonstrates that the proposed approach outperforms existing distributed controller clustering (DCC) schemes across various performance metrics, including delay, jitter, throughput, load balancing, scalability, and CPU utilization.

Employing SDN in IoT networks holds the promise of simplifying IoT complexity and delivering improved QoS. As the demand for IoT continues to grow, the network will likely accommodate an increasing number of sensors, resulting in substantial network traffic. With the potential surge in traffic generated by IoT sensors, SDN controllers may struggle to cope with the processing demands. To address this challenge and achieve optimal network performance, a dynamic allocation of slave controllers with a strategic mechanism becomes essential for effective task management and migration planning. In response, [13] have introduced an efficient approach based on slave controller allocation for load balancing within a multi-domain SDN-enabled IoT network. This approach is designed to intelligently distribute switches to controllers with available resources. Among the various slave controllers considered for selecting a target controller, the approach employs a multi-criteria decision-making (MCDM) strategy known as the Analytical Network Process (ANP). This enriches communication metrics and maintains high-quality QoS statistics. Furthermore, the model utilizes switch migration using a knapsack 0/1 problem to maximize the utilization of slave controllers. The proposed scheme offers a flexible decision-making process that accommodates controllers with varying resource capacities. The results, as demonstrated in an emulation environment, underscore the effectiveness of the approach, referred to as ESCALB, in addressing the challenges and optimizing SDN-enabled IoT networks.

3 Design and implementation

The system architecture in Fig. 1 includes multiple OpenFlow switches connected to SDN controllers. Each switch is linked to multiple clients and servers, and every task received by the switch is transmitted to the controller's application as PACKET_IN messages. The controller application processes these tasks and provides instructions to the switch on how to handle them. Additionally, an additional system controller is introduced into the architecture as depicted.

At regular intervals, each controller transmits information about its current resource consumption and load to the system controller. The system controller plays the role of monitoring the state of all controllers within the system by gathering essential data related to their load and performance. The overloaded controller sends a message to inform the system controller that it has reached the migration threshold. Upon receiving this notification, the system controller instructs the controller to initiate the migration process by executing the migration application. The migration application selects the most heavily loaded switch to be migrated. Subsequently, the migration application running on the controller requests the ID of the target controller to which the selected switch should be migrated. The system controller takes charge of selecting the new controller by utilizing the NSGA-II algorithm. Since it possesses comprehensive information about the state of each controller, the NSGA-II algorithm can determine the optimal controllers based on two key parameters: the controller's performance (P_{er}) and the migration cost (C). These parameters are influenced by various metrics, including RAM and CPU consumption, delay, and the cost associated with exchanging migration message packets. The calculation of each parameter is performed using mathematical computations. Upon receiving the ID of the new controller, the overloaded controller initiates communication with the new controller to facilitate the switch migration process. This involves synchronizing their parameters and exchanging essential information about the target switch, such as the OpenFlow protocol version, IP address, system version, and other relevant details.

Algorithm 1 presents the switch migration to the destination controller chosen based on the metrics calculated.

3.1 Migration performance calculation

The first parameter is performance (P_{er}), which indicates the current performance level of each controller. Performance is calculated using (1):

$$P_{er}(t) = RAM_{left}(t) + CPU_{left}(t) \tag{1}$$

P_{er}: Controller resource performance left at a specific time t.

RAM_{left}: Memory capacity percentile left at a specific time t.

CPU_{left}: CPU capacity percentile left at a specific time t.

RAM and CPU resource availability is calculated by (2) and (3).

$$RAM_{left}(t) = \frac{RAM_c - \left(RAM_{app} * Ns(t)\right)}{RAM_c} \tag{2}$$

RAM_c: Total RAM in the controller.

RAM_{app}: Total RAM required to manage a switch application.

```
    Input: load threshold $LT$; $controllerList$;
    Output: $destController$; $switchToMigrate$;
1  Initialization: $controllerQueue \leftarrow \emptyset$; connectedSwitches $\leftarrow \emptyset$; switchToMigrate $\leftarrow \emptyset$; destController $\leftarrow \emptyset$; netDevController $\leftarrow \emptyset$; switchLoad $\leftarrow \emptyset$; controllerLoad $\leftarrow \emptyset$; CqueueLoad $\leftarrow \emptyset$; maxSwitchLoad $\leftarrow \emptyset$;
2  for each $TimeUnit$ in $SimulationTime$ do
3      for each $C$ in $controllerList$ do
4          $netDevController \leftarrow getNetDevController\ (C)$
5          $controllerQueue \leftarrow getcontrollerQueue\ (netDevController)$
6          $CqueueLoad \leftarrow getCQueueLoad\ (controllerQueue)$
7          if $CqueueLoad = LT$ then
8              $connectedSwitches \leftarrow getConnectedSwitches\ (C)$
9              for each $switch$ in $connectedSwitches$ do
10                 $switchLoad \leftarrow getSwitchLoad\ (switch)$
11                 if $switchLoad > maxSwitchLoad$ then
12                     $switchToMigrate \leftarrow switch$
13                     $maxSwitchLoad \leftarrow switchLoad$
14                 end if
15             end for
16             for each $controller$ in $controllerList$ do
17                 $controllerLoad \leftarrow getSwitchLoad\ (controller)$
18                 if $CqueueLoad > controllerLoad$ then
19                     $destController \leftarrow controller$
20                     $controllerLoad \leftarrow CqueueLoad$
21                 end if
22             end for
23             $C.\ remove$ FromSwitchList ($switchToMigrate$) $destController$ .addToSwtichList($switchToMigrate$)
24             perform hello handshake between $destController$ and $switchToMigrate$
25         end if
26     end for
27 end for
```

Algorithm 1 Pseudo code for switch migration

N_s: number of switches managed by the controller c at a specific time t.

$$CPU_{left}(t) = \frac{CPU_c - \left(M_{cpu} * n_p(t)\right)}{CPU_c} \tag{3}$$

CPU_c: CPU capacity of the controller c.

M_{CPU}: CPU power required to process a packet.

n_p: number of packets processed by the controller c at a specific time t.

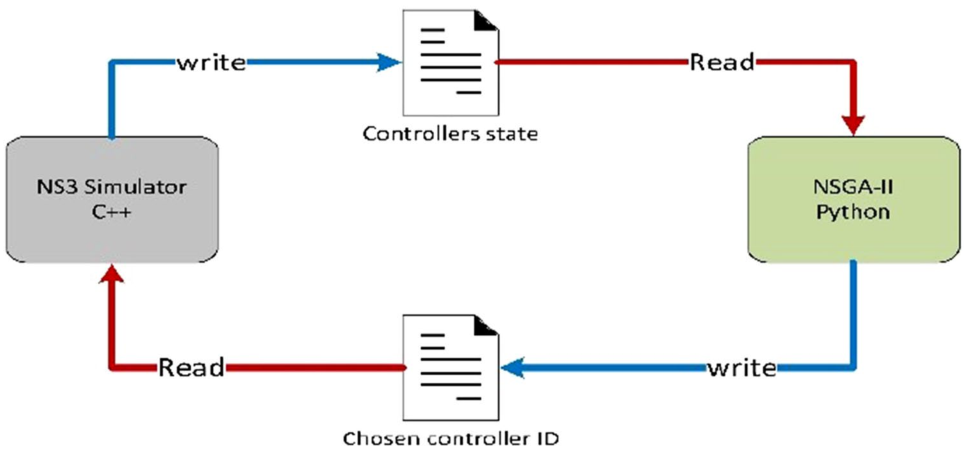


Fig. 2 Interaction between NSGA-II and NS3

3.2 Migration cost calculation

The cost of migration is calculated as in (4). While the dropped packets during migration and the cost of migration are presented in (5) and (6). Equation 7 presents the delay of each control message sent.

$$C = C_x + C_{Dp} \tag{4}$$

C_x: The cost of the exchange control messages between the controllers during the migration.

C_{Dp}: The dropped packets while performing the migration.

$$C_{Dp} = k * \alpha \tag{5}$$

k: the number of dropped packets.

α: the delay in sending control messages.

$$C_x = d * n_x * \gamma \tag{6}$$

d: the distance between different controllers, represented by the number of hops from source to destination.

n_x: number of exchanged control messages between controllers.

γ: delay of each control message sent.

$$\gamma = \frac{M_{size}}{\beta} \tag{7}$$

M_{size}: control message size.

β: link bandwidth.

3.3 NSGA-II implementation

In our case, we employ Non-dominated Sorting Genetic Algorithm (NSGA-II) as a multi-objective optimization algorithm to select the most suitable controller based on performance and minimal migration cost. This algorithm involves non-dominated sorting and crowding distance sorting to identify the optimal migration controller. Given that the NS3 simulation is implemented in C++ while NSGA-II is developed in Python, the interaction between the two languages is facilitated through external files, as illustrated in Fig. 2. The state of each controller, comprising its performance and migration cost, is periodically recorded in an external file. When executing the NSGA-II algorithm, the program reads this external file to determine the optimal new controller. Subsequently, the output of the NSGA-II algorithm, i.e., the selected controller, is written to another file. This file is then read by the NS3 simulation to facilitate the migration process.

NSGA -II addresses the optimization of two conflicting objectives: controller performance and migration cost. This approach uses rigorous mathematical principles to ensure efficiency and effectiveness. For efficient initialization and objective functions, this proposed approach for SDN switch migration using the NSGA-II, prioritizes efficiency right from the initialization of optimization process. It commences by initializing a population of potential solutions, denoted as a set of real numbers X, where each solution x_i falls within a predefined range [min_x, max_x]. This initialization procedure ensures diversity in exploration of controller configurations.

Population Initialization,

$$X = \{x_1, x_2, \ldots, x_N\} \tag{8}$$

Furthermore, two pivotal objective functions are:

i. Controller performance objective is represented as FP(x), quantifying controller performance.

$$FP(x) = -x^2 \tag{9}$$

ii. Migration cost objective assesses migration cost employing FM(x), which is expressed as,

$$FM(x) = -(x - 2)^2 \quad (10)$$

A distinctive feature of this approach is the seamless integration of NS3 SDN data. This integration is facilitated through an external dataset, denoted as D, which contains crucial information about controller performance (P_{er}) and migration cost (C) for each controller. Dataset is represented as,

$$D = \left\{ \left(Per_1, C1\right), \left(Per_2, C_2\right), \ldots, \left(Per_N, C_N\right) \right\} \quad (11)$$

The NSGA-II algorithm categorizes solutions into distinct fronts based on their dominance relationships, as determined by our defined objective functions. Specifically, a solution A dominates a solution B if the following dominance condition holds true:

$$FP(A) \le FP(B) \text{ and } FM(A) \le FM(B) \quad (12)$$

This dominance relation is at the core of NSGA-II's non-dominated sorting, allowing us to categorize solutions into fronts, where each front contains solutions that are not dominated by others. Moreover, to maintain diversity within each front and prevent premature convergence, the crowding distance (CD) calculation is employed. CD reflects the density of solutions around a given solution and is determined by Eq. 13.

$$CD(x) = \left(FP\left(x_{next}\right) - FP\left(x_{prev}\right)\right)/(\max(FP) - \min(FP)) \quad (13)$$

where x_{next} and x_{prev} represent solutions adjacent to 'x' based on their rank within the same front. A similar calculation is applied to FM, ensuring that diversity is preserved in both objective spaces. In the selection phase of NSGA-II, selection probabilities are assigned to solutions based on their ranks and crowding distances. These probabilities are denoted as $P_{select}(x)$ to guide the choice of parent solutions for crossover. The weighted combination of two parent solutions, x_i and x_j, is represented by Eq. 14.

$$x_{offspring} = \alpha * x_i + (1 - \alpha) * x_j \quad (14)$$

where α is a random weight within the range (0, 1). This mechanism ensures that parent solutions with superior ranks and crowding distances have a higher likelihood of contributing to the next generation while preserving diversity within the population. To introduce controlled randomness and enhance the exploration of the solution space, this work incorporates a mutation mechanism into the proposed approach. Each solution 'x' has an associated mutation probability, denoted as $P_{mutation}$. If $P_{mutation}$ is less than 1, a mutation operation is applied as follows.

$P_{mutation}$ is a stochastic event that determines whether a solution undergoes mutation. In each generation of the NSGA-II algorithm, $P_{mutation}$ is a random variable taking values in the range of (0, 1) to represent the probability of mutation. In mathematical terms, $P_{mutation}$ follows a uniform distribution, $P_{mutation} \sim U(0,1)$. U(a,b) represents a uniform distribution with a lower bound 'a' and an upper bound 'b'. This distribution models the random nature of mutation probability, allowing for controlled exploration within the optimization process.

If $P_{mutation} < 1$, a mutation is applied:

$$x_{mutation} = min\_x + (max\_x - min\_x) * P_{mutation} \quad (15)$$

This mechanism allows for the emergence of unexpected and potentially more efficient solutions while preventing convergence to local optima. NSGA-II runs for a specified number of generations (max_gen). After termination, the algorithm produces a set of non-dominated solutions, forming the Pareto front. These solutions provide a trade-off between controller performance and migration cost.

4 Performance evaluation

4.1 Experimental environment

In this section, we focus on obtaining performance results for the switch migration and assembly strategy proposed in our previous linked work [14] by configuring simulation parameters. These parameters are carefully selected based on extensive runs to effectively evaluate the performance of migration and aggregation. For instance, the bandwidth parameter is set to a sufficiently high value to facilitate the transmission of numerous packets from the switch to the controllers, thereby triggering controller overloading. Without this condition, there would be no need to initiate switch migration. The remaining parameters such as the number of clients, packet size, and controller CPU are also selected to analyze the system's behavior comprehensively. We conduct the simulation in two distinct scenarios:

(1) In the first scenario, we vary the number of clients to generate a larger volume of data, thereby increasing the workload on the controllers.
(2) In the second scenario, we modify the switch migration threshold in the controllers to trigger switch migration at different levels of controller load.

Table 1 provides an overview of the simulation configuration, showcasing the values assigned to each parameter for the respective scenarios.

Table 1 Simulation parameters

Number of clients	[200–600]
Number of controllers	3
Number of switches	6
Packet size (KB)	500
Link data rate (Gbps)	1
Controller CPU rate (Gbps)	4
Controller RAM (GB)	[2–4]
Switch application RAM (MB)	500
Migration threshold (%)	50
Burst assembly threshold (n)	50

4.2 Results

The results are compared with state-of-the-art methods presented in [15–17]. In scenario 1, we observe through the total packets transmitted graphs depicted in Figs. 3 and 4 that as the number of hosts increases, a corresponding rise in the number of packets generated is observed across all strategies. However, it is worth noting that there is a discernible difference between the strategies. Specifically, when 200 hosts are present, all migration strategies exhibit nearly identical numbers of transmitted packets.

Nevertheless, the classic strategy (without migration and burst assembly) shows a lower count of transmitted packets compared to the other strategies. This disparity

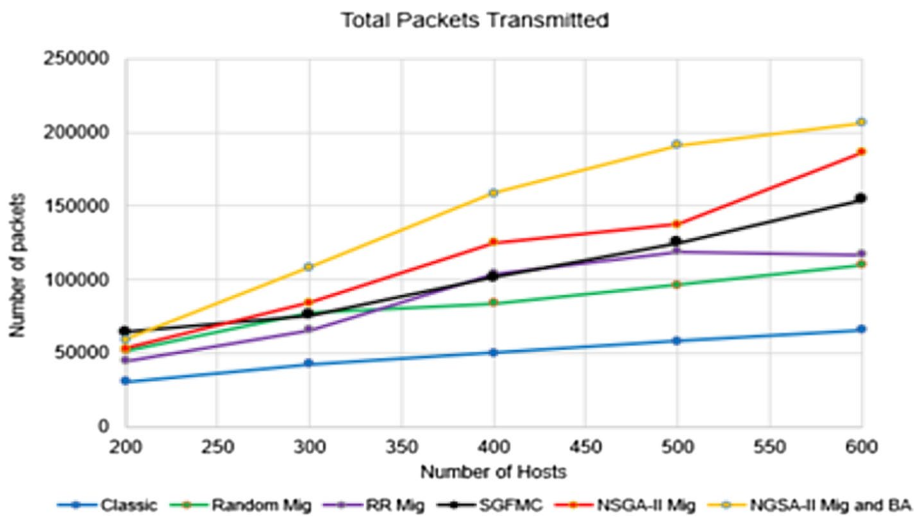


Fig. 3 Packets transmitted with variable hosts

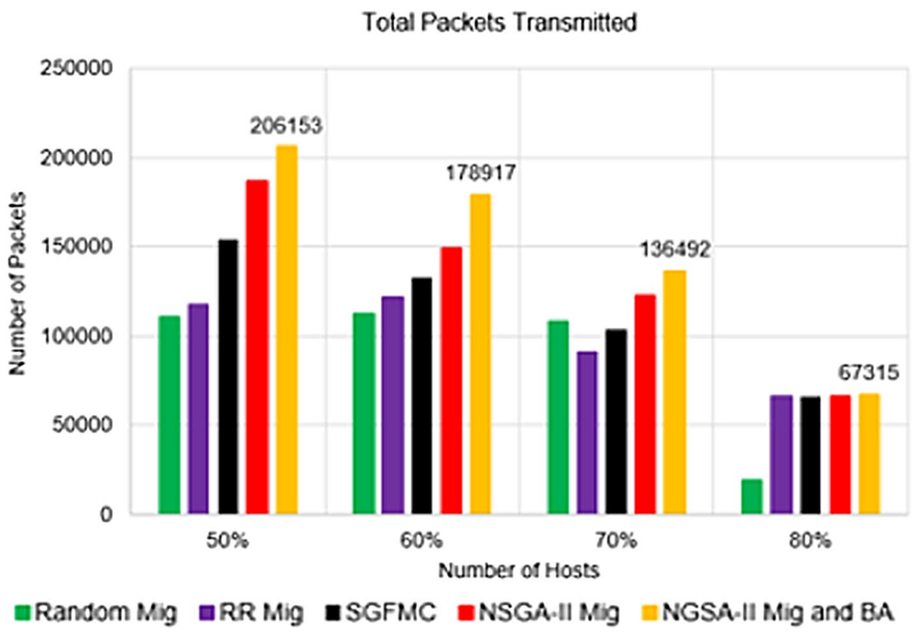


Fig. 4 Packets transmitted with migration threshold

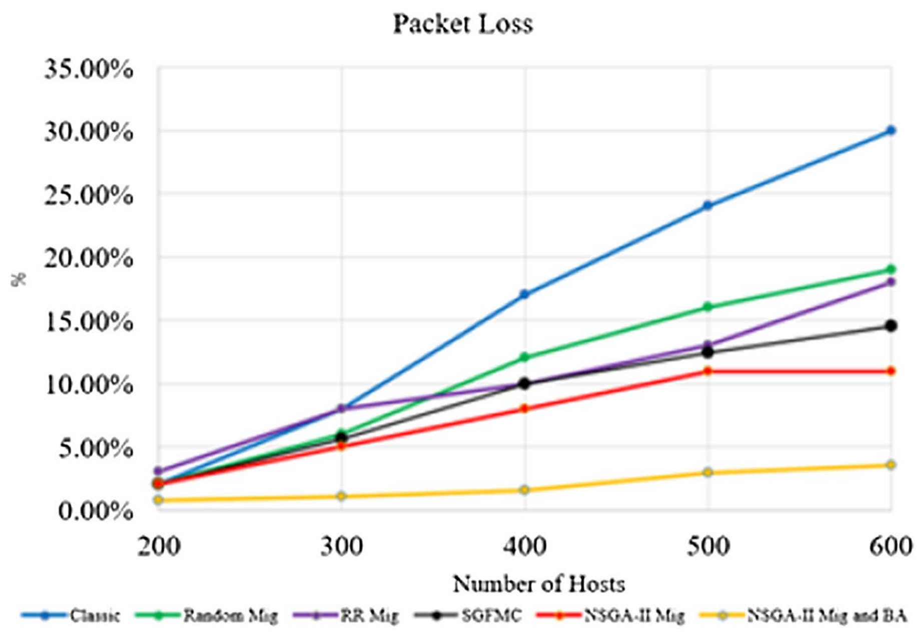


Fig. 5 Packet loss with variable hosts

can be attributed to the communication overhead caused by the migration processes. Additionally, as the number of hosts increases, the discrepancy in the number of transmitted packets among the strategies becomes more pronounced. This is primarily because the system experiences substantial packet loss, which can be attributed to two main factors: controller overloading and link saturations. The packet loss is illustrated in the packet loss graph shown in Fig. 5.

The packet loss graph clearly indicates that the classic strategy (without migration and burst assembly) exhibits the highest packet loss rate, reaching up to 30%. This is expected as this strategy does not employ any specific measures to mitigate controller overloading or links saturation. On the other hand, the migration strategies demonstrate improved packet loss performance, with a lower percentage of dropped packets attributed to controller limitations. Notably, the NSGA-II migration strategy shows a significant reduction in packet loss (approximately 10% with 600 hosts) compared to random migration. This can be attributed to the selection of the optimal controller for switch migration, which effectively alleviates controller overloading. Furthermore, the combination of NSGA-II migration and burst assembly achieves the best performance with packet loss below 5%. This strategy successfully addresses both controller overloading and links saturation, which are not effectively mitigated by switch migration alone.

Now, let's delve into the performance and cost outcomes of the NSGA-II migration combined with burst assembly, as depicted in Figs. 6 and 7. The performance result showcases the performance of the selected controller, while the cost results illustrate the associated migration cost. These outcomes specifically pertain to the switch migration strategies, as explained earlier in the equations. Hence, these results exclusively focus on the implications of the switch migration strategies.

It is evident that both migration performance and cost increase as the number of hosts grows, since more switch migrations are triggered. Notably, the NSGA-II migration strategy exhibits superior performance compared to random migration. This is due to the NSGA-II's ability to select a high-performance controller for migration, resulting in improved overall performance. Additionally, NSGA-II migration achieves this with minimal cost, as depicted in the graphs. Based on this analysis, it can be concluded that a random migration strategy may occasionally select a less performant controller with higher associated costs. In contrast, utilizing the NSGA-II algorithm allows for an optimal choice balancing performance and cost considerations. Furthermore, the burst assembly algorithm effectively addresses network issues. Network performance metrics are significantly enhanced when compared to strategies without burst assembly (Classic, random, and NSGA-II).

Simultaneously, the migration takes place with the objective of achieving the lowest cost, as depicted in the figures. From this analysis, it can be inferred that a random migration strategy may occasionally select a less performant controller with a higher cost. However, by utilizing the NSGA-II multi-objective optimization algorithm, an optimal choice can be made considering both performance and cost factors. Furthermore, the burst assembly algorithm effectively tackles network issues. It leads to substantial enhancements in network performance

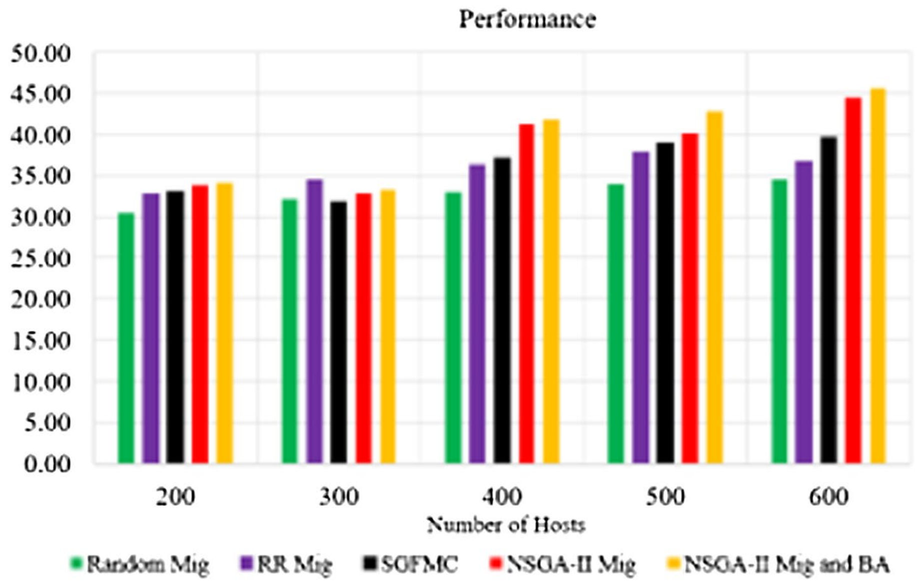


Fig. 6 Performance with variable hosts

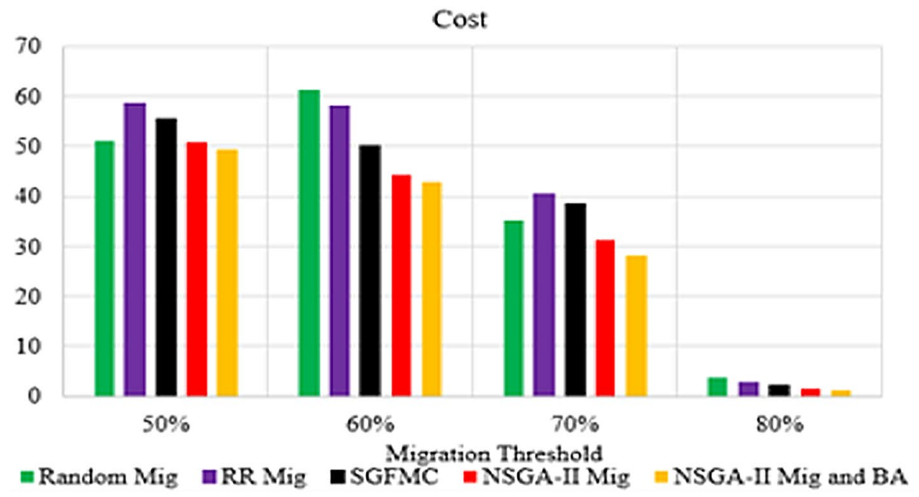


Fig. 7 Cost with variable thresholds

metrics compared to strategies without burst assembly, such as classic, random, and NSGA-II approaches.

5 Conclusion

This paper focuses on exploring the load balancing challenge among multiple SDN controllers while addressing communication overhead. To enhance communication overhead, we employ our previously proposed burst assembly method that operates based on the same destination principle. Additionally, for migration purposes, we adopt an optimal controller selection approach using the NSGA-II algorithm, considering both cost and performance factors. Enhancing the overall system performance involves tackling both processing and networking loads concurrently. The migration strategy plays a crucial role in improving CPU response time by migrating switches from high load controllers to ones with lower loads. Additionally, the NSGA-II algorithm enables the selection of more suitable controllers based on their performance and cost, resulting in significant improvements in controller computing speed and reduced packet loss. On the networking side, the burst assembly technique proves beneficial in addressing networking metrics by assembling packets destined for the same destination. As a result, the switch migration strategy utilizing NSGA-II and burst assembly complement each other and contribute to achieving optimal system performance.

The research proposed here has major contributions in terms of selecting right controller and using burst assembly. The aggregation here is based on same destination while in future work, aggregation without destination can be achieved. Other than CPU response time, CPU utilization can be considered. NSGA-II can also be used to select controller other than performance and cost. Another future extension is selecting right switches to connect to the right controller using fair selection.

6 Discussion

Selecting an optimal controller in SDN is crucial for ensuring efficient operation. It addresses the problems of over migration and the cost associated with the migrations. This study has tackled the same problem by mitigating unnecessary migrations and their associated expenses. It accomplishes this by carefully selecting the most suitable controller based on factors such as cost and performance. Control communication is influenced by numerous variables, including the need for updates and the addition of missing entries, which can impose an extra burden. To alleviate this, the proposed approach minimizes control communication by introducing a burst assembly technique for communication to the same destination. Section II of the paper provides an overview of the existing literature that addresses controller selection challenges. In response to this challenge, our work leverages NSGA-II and implements it within ns-3 to determine controller states and make informed controller choices. The choice of controller holds a pivotal role in the context of NG-IoT within SDN and sets the stage for further advancements in this domain. Future endeavors will involve expanding the criteria for controller selection beyond cost and performance considerations.

Author contributions MS—Writing, Concept, Experiment, Simulation. LL—Supervisor, Review. NB—Experiment, Concept discussion. NA—Supervision.

Data availability Data sharing is not applicable to this article as no datasets were generated or analyzed during the current study.

Declarations

Competing interests The authors declare no competing interests.

References

1. Aqdus A et al (2023) "Detection collision flows in SDN based 5G using machine learning algorithms. Comput Mater Continua 75(1):1413
2. Duraisamy A, Subramaniam M, Robin CRR (2021) An optimized deep learning-based security enhancement and attack detection on IoT using IDS and KH-AES for smart cities. Stud Inf Control 30(2):121–131
3. Patel MJA, Punam C, Phade GM (2021) Design of SMV model in machine to machine (M2M) communication for 5G network. IJETT 8(2):138
4. Maity I, Mondal A, Misra S, Mandal C (2018) CURE: consistent update with redundancy reduction in SDN. IEEE Trans Commun 66(9):3974–3981
5. Dixit A, Hao F, Mukherjee S, Lakshman TV, Kompella RR (2014) ElastiCon; an elastic distributed SDN controller. In: ACM/IEEE symposium on architectures for networking and communications systems (ANCS), IEEE, pp. 17–27
6. Chou LD, Yang YT, Hong YM, Hu JK, Jean B (2014) A genetic-based load balancing algorithm in openflow network. In: Huang YM, Chao HC, Deng DJ, Park JJ (eds) Advanced technologies, embedded and multimedia for human-centric computing. Springer, Dordrecht, pp 411–417
7. Qin K, Huang C, Wang C, Chen X (2016) Balanced multiple controllers' placement with latency and capacity bound in software-defined network. J Commun 37(11):90–103
8. Tao H, Jian-Hui Z, Teng M, Wei Z (2017) "Multi-controller balancing deployment strategy based on reliability evaluation in SDN. J Commun 38(11):188–198
9. Hu T, Yi P, Guo Z, Lan J, Zhang J (2018) Bidirectional matching strategy for multi-controller deployment in distributed software defined networking. IEEE Access 6:14946–14953
10. Ali J, Roh BH (2022) A novel scheme for controller selection in software-defined internet-of-things (SD-IoT). Sensors 22(9):3591
11. Ali J, Roh BH, Lee S (2019) QoS improvement with an optimum controller selection for software-defined networks. PLoS ONE 14(5):e0217631
12. Ali J, Roh BH (2021) Quality of service improvement with optimal software-defined networking controller and control plane clustering. Comput Mater Contin 67:849–875
13. Ali J, Jhaveri RH, Alswailim M, Roh BH (2023) ESCALB: an effective slave controller allocation-based load balancing scheme for multi-domain SDN-enabled-IoT networks. J King Saud Univ Comput Inform Sci 35(6):101566
14. Shahzad M, Liu L, Eddine N (2023) Control overhead reduction using length-based same destination aggregation (LSDA) for large scale software defined networks in next generation internet of things, Accepted,. The 26th IEEE International Conference on Computational Science and Engineering (CSE-2023), Exeter, UK, 1–3 Nov 2023
15. Cziva R, Jouët S, Stapleton D, Tso FP, Pezaros DP (2016) SDN-based virtual machine management for cloud data centers. IEEE Trans Netw Serv Manage 13(2):212–225
16. Raza A, Lee S (2018) Gate switch selection for in-band controlling in software defined networking. IEEE Access 7:5671–5681
17. Soleimanzadeh K, Ahmadi M, Nassiri M (2019) SD-WLB: An SDN-aided mechanism for web load balancing based on server statistics. ETRI J 41(2):197–206