A Self-Adaptive Load Balancing Approach for Software-Defined Networks in IoT

Ziran Min*, Hongyang Sun*†, Shunxing Bao*, Aniruddha S. Gokhale*, Swapna S. Gokhale‡
*Dept. of EECS, Vanderbilt University, Nashville, TN, USA
{ziran.min, shunxing.bao, a.gokhale}@vanderbilt.edu
†Dept. of EECS, University of Kansas, Lawrence, KS, USA
{hongyang.sun}@ku.edu
‡Dept. of CSE, University of Connecticut, Storrs, CT, USA
{swapna.gokhale}@uconn.edu

***Abstract*—The Internet of Things (IoT) is gaining popularity as it offers to connect billions of devices and exchange data over the internet. However, the large-scale and heterogeneous IoT network environment brings serious challenges to assuring the quality of service of IoT-based services. In this context, Software-Defined Networking (SDN) shows promise in improving the performance of IoT services by decoupling the control plane from the data plane. However, existing SDN-based distributed architectures are able to address the scalability and management issues in static IoT scenarios only. In this paper, we utilize multiple M/M/1 queues to model and optimize the service-level and system-level objectives in dynamic IoT scenarios, where the network switches and/or their request rates could change dynamically over time. We propose several heuristic-based solutions including a genetic algorithm, a simulated annealing algorithm and a modified greedy algorithm with the goal of minimizing the queuing and processing times of the requests from switches at the controllers and balancing the controller loads while also incorporating the switch migration costs. Empirical studies using Mininet-based simulations show that our algorithms offer effective self-adaptation and self-healing in dynamic network conditions.**

***Index Terms*—Switch Migration Problem (SMP), Software-Defined Networking (SDN), Internet of Things (IoT), Queuing Model, Load Balancing, Resource management.**

I. Introduction

The Internet of Things (IoT) is gaining popularity and acceptance in many applications, such as smart cities, smart homes, and smart healthcare [1]. This is evident from a 2021 Gartner survey indicating about 25 billion endpoints already connected as part of different IoT deployments [2]. Cisco also forecasts that there will be 500 billion IoT devices connected by 2030 [3]. Despite these promising trends, IoT still faces many challenges stemming from the large-scale and heterogeneous network environments in which they operate.

In traditional IoT deployments, the control plane and data plane of IoT networks are tightly coupled as shown on the left side of Figure 1. Many technical limitations stem from this approach, such as lack of scalability, poor resource management, and expensive network devices, that have pushed the operations of traditional IoT networks into stasis. To fix these deficiencies, the Software-Defined Networking (SDN) paradigm offers a promising alternative as illustrated on the right side of Figure 1. SDN decouples the data plane from the control plane, which helps mitigate the lack of network scalability and resource management issues. A logically centralized control plane allows network administrators to install, configure and update network devices seamlessly by running network applications, thereby significantly reducing maintenance errors that can otherwise be caused by manual configurations.

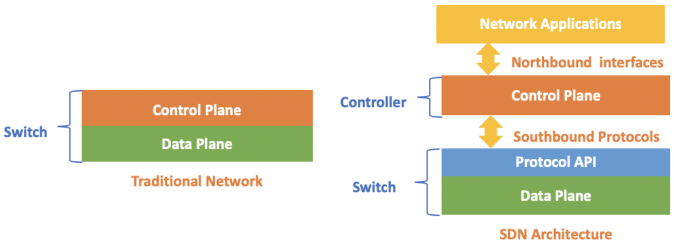

Fig. 1: Traditional network vs. Software Defined Network (SDN).

Early efforts [4], [5], [6] on SDN technology deployments used one physically centralized controller. However, issues of scalability and reliability emerged with a single centralized controller. With increasing network traffic, this approach could hardly meet the growing demands from the users. Furthermore, an individual controller could easily become a single point of failure. To solve these issues, logically centralized but physically distributed controllers were introduced as a more scalable and reliable approach to SDN [7], [8], [9].

Despite these promising directions, in larger-sized network topologies or networks with fluctuating dynamics, such as in IoT, it becomes difficult for SDN developers to determine how many controllers to deploy and where to place them in the network to address the dynamically changing conditions. Heller et al. [10] first introduced the Controller Placement Problem (CPP), which incorporated minimizing the average and worst-case latency between the controllers and the switches, as well as maximizing the number of nodes within a latency bound. Besides defining and quantifying CPP, they also showed that the problem is worth further investigation. Subsequent work, such as in [11] and [12], delivered a number of solutions to minimize latency from different sources and to balance

the network packet load, resulting in significantly improved network performance.

Despite these prior research efforts, the solutions they proposed operated only under the static assumptions, which ignored the dynamic nature of the network. Due to the explosive growth of IoT devices we are witnessing, however, the distribution of traffic load is always uneven. The unbalanced traffic in the data plane may cause unpredictable delays on essential communication services and real-time applications. Thus, SDN-based network services [13], [14], [15] are applied to monitor and mitigate the traffic in the data plane. While the network services do bring benefits to the data plane, they also result in additional traffic on the control plane. To address this issue, Wang et al. [16] introduced the Switch Migration Problem (SMP), which aims to use live switch migration to improve the performance and scalability in distributed controllers. However, many existing studies [16], [17], [18] that consider the SMP approach focused only on minimizing the switch migration cost without considering the latency between the controllers and the switches.

Addressing these challenges, particularly in IoT scenarios where human intervention is not always feasible, requires autonomic self-healing and self-adapting solutions. To that end, this paper provides such a solution based on a multi-objective optimization approach that minimizes the latency, balances the controller loads and reduces the switch migration cost all at once, and thereby comprehensively and effectively solves the switch migration problem in dynamic IoT networks.

The primary contributions of our work that distinguish them from prior related works are as follows:

- We consider a multi-objective approach to solving the SMP in dynamic scenarios, which includes autonomically adding/deleting switches to/from the network topology and increasing/decreasing the switch request rates.
- We incorporate objectives that capture both service-level and system-level performance, which includes switch-controller latency, controller load balancing, and switch migration cost.
- We formulate an optimization problem while using multiple distributed M/M/1 queues [19] to build our network model and Little's law [20] to analyze the queuing and processing times of the switch requests on controllers.
- We propose three heuristic-based solutions, including a genetic algorithm, a simulated annealing algorithm, and a modified greedy algorithm, to solve the formulated SMP problem that is codified in a self-healing and self-adapting systems software solution.
- We evaluate the performance of the proposed algorithms using Mininet-based simulations, and the results show that they are able to offer solutions that can effectively adapt to dynamic network conditions.

The rest of this paper is organized as follows. Section II provides a brief overview of the related work. The formal problem description is presented in Section III. In Section IV, our three heuristic algorithms are described and explained, followed by the system architecture in Section V. The performance evaluations are presented in Section VI. Finally, Section VII offers concluding remarks alluding to future work.

II. Related Work

This section compares our work to relevant prior efforts. The previous study [21] classified the load balancing method into a static group and a dynamic group. In static scenarios, load balancing is one of the important objectives of solving CPP. In dynamic scenarios, a switch migration based approach is usually used to balance the controller loads. Thus, we consider load balancing in both CPP and SMP.

A. Controller Placement Problem (CPP)

After the CPP problem was proposed, extensive literature has been published due to the significant interest in solving CPP. A comprehensive survey [22] shows that recent efforts concentrate on a single objective. To minimize the end-to-end latency and queue latency, Wang et al. [11] proposed a clustering-based network partition algorithm, which shortens the maximum end-to-end latency between the switches and controllers. In addition to the control latency, researchers have considered more critical system metrics in SDN, such as resilience, reliability and load balancing.

To improve resiliency, Guo et al. [23] introduced interdependence network analysis into CPP. They proposed a new resilience metric that calculates the number of nodes that survive in the steady stage of a cascading failure. Since control load is a critical factor, Yao et al. [24] defined the Capacitated Controller Placement Problem, which takes into consideration the load on controllers. They proposed a new *k-center* strategy combined with dynamic controller provisioning or dynamic scheduling. The strategy is able to reduce the number of controllers and the load of the maximum loaded controller.

Some of the existing works focus on optimizing multiple performance metrics simultaneously. To minimize the switch-controller latency and balance the controller load at the same time, Bo, et al. [25] proposed a two-stage solution. First, they employ a multi-objective genetic algorithm to obtain the connection relationship between controllers and switches. After applying the algorithm, the network is partitioned into a specified number of controllers. Second, a minimum delay algorithm is used to find an optimal location which uses minimum sum of the distance to all switches in its partition. To maximize network reliability and controller load balance ability, and to minimize latency between controllers and switches, Zhang et al. [12] formulated this multi-objective problem into a mathematical model. They used the Adaptive Bacterial Foraging Optimization algorithm to solve the optimization objective function derived from the model.

B. Switch Migration Problem (SMP)

The study in [16] introduced SMP and solved it by using a two phase greedy algorithm, which includes load detection and migration action generation. The approach is able to improve migration efficiency. However, the approach ignored the control latency, which is one of the critical objectives in SMP.

The work in [17] proposed a switch migration protocol, which improves system service-level performance. But this protocol increases the controller response time when the packet arrival rate exceeds a threshold.

In summary, the previous approaches comprising only a single objective focus ignored the effects of other objectives while the approaches involving multiple objectives focused only on the static scenario. In contrast, our work falls in the multiobjective category that can handle dynamic changes in an autonomic manner. Therein we propose an approach that is able to balance the controller loads by addressing multi-objective SMP in dynamic scenarios.

III. Problem Description

Our work focuses on developing self-healing and self-adapting solutions for the Switch Migration Problem (SMP) in SDN used in IoT scenarios. In this section, we will formally describe our system assumptions, introduce our network model and formulate the optimization problem.

A. System Assumptions on IoT Deployments

We assume an asynchronous distributed system to design our SMP solution in SDN. We make the following assumptions in this work.

- Our system is representative of large-scale Local Area Networks (LANs) for IoT systems.
- Our deployed solution is capable of monitoring controller loads, analyzing switch request rates, and generating and executing switch migration plans.
- In the network, each switch connects to at most one controller and the switch can be reassigned from one controller to another.
- We focus on the workload that handles the *packet-in* event at the controllers, which according to previous studies (e.g., [26], [27]) is the primary task for controllers.
- The dynamic scenarios we consider in this work include adding/deleting switches to/from the network and increasing/decreasing the switch request rates. These scenarios also include switch failures.

B. Network Model

Heterogeneous IoT devices generate various kinds and volumes of data, which lead to network congestion, thereby affecting the service-level performance. A critical factor that impacts IoT performance is the amount of queue build-up in switches. Some previous studies have used the M/M/1 and M/M/N queuing models to schedule Emergency packets (E-packets) and Regular packets (R-packets) on the switch side in IoT scenarios [28], [29]. We choose the M/M/1 queueing model for its simplicity to schedule the packet-in event on the controller side. Specifically, our work uses M/M/1 queues to model the queuing latency of the requests based on the controller load, and the processing latency based on controller service rate. M/M/1 queuing was also used to model the SDN's OpenFlow architecture as it was applied successfully to analyze the queuing latency [11]. Moreover, a detailed diagnosis of IoT traffic flow by flow indicates that the workload of IoT follows Poisson distribution [30]. In IoT, the flows representing requests can arrive from independent switches, with the interval of arrival time and the server processing time constituting a negative exponential distribution. Thus, the arriving requests follow a Poisson distribution and hence a M/M/1 approach is justifiable.

We assume there are n switches and m controllers in our system. The request flow rate of the i^{th} switch is denoted by λ_i. The total flow request rate is $\lambda = \sum_{i=1}^{n} \lambda_i$. Thus, there are m M/M/1 queues in our system. Every controller has a queue. The centralized scheduler will assign the incoming request flows from the n switches to the m controllers' queues. This scheduler runs in the application layer, where a switch migration plan will be generated. Then, the individual controllers will process the request flows from their respective queues until the controller-to-switch assignment changes. Here, we assume that all the controllers have the same service rate that is denoted by μ. The controller placement decision is indicated by a binary variable x_{ij}, where $x_{ij} = 1$ means that the i^{th} switch is connected to the j^{th} controller. This service model between switches and controllers is shown in Figure 2.

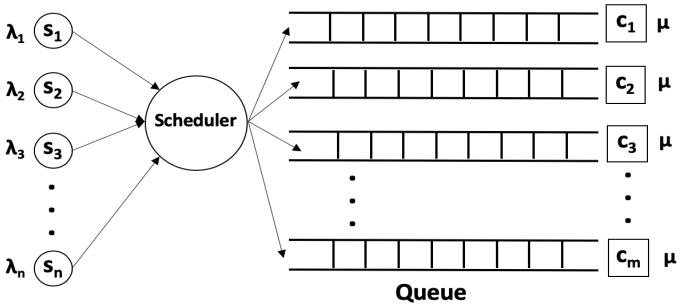

Fig. 2: The service model between switches and controllers.

Table I shows the list of notations used in the problem formulation.

TABLE I: LIST OF NOTATIONS

Symbol	Meaning
s_i	i^{th} switch, total n switches
c_j	j^{th} controller, total m controllers
λ_i	request rate of i^{th} switch
μ	service rate of all controllers
X	the controller to switch assignment matrix; $x_{ij} = 1$ means that i^{th} switch is connected to j^{th} controller, $x_{ij} \in \{0, 1\}$, $\forall i, j$
θ_j	load of j^{th} controller
$\overline{\theta}$	average load of all controllers

C. Problem Formulation

Assuming that the request flow arrivals follow a Poisson process with $\sum_{i=1}^{n} \lambda_i < m\mu$, and λ_i's are mutually independent, the load of the j^{th} controller can be represented by:

$$\theta_j = \sum_{i=1}^{n} \lambda_i x_{ij} \tag{1}$$

and the average load among all controllers is given by:

$$\overline{\theta} = \frac{1}{m} \sum_{j=1}^{m} \theta_j \tag{2}$$

Applying Little's law, the expected waiting time before a request from a switch is served can be represented by:

$$W_q = \frac{\theta_j}{\mu(\mu - \theta_j)} \tag{3}$$

and the expected end-to-end response time, which is the sum of the request queuing time and the controller service time, is represented by:

$$W_s = W_q + \frac{1}{\mu} = \frac{1}{\mu - \theta_j} \tag{4}$$

In this work, we aim to minimize both a service-level objective and a system-level objective. Specifically, the service-level objective is the average latency of all the requests represented by the end-to-end response time while the system-level objective is the balance of all the loads on the controllers represented by their variance. The following illustrates the problem formulation:

$$\min \quad F(X) = \frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{m} \frac{x_{ij}}{\mu - \theta_j} + \frac{1}{m} \sum_{j=1}^{m} (\theta_j - \overline{\theta})^2 \tag{5}$$

$$\text{s.t.} \quad \sum_{i=1}^{n} \lambda_i x_{ij} < \mu, \forall j \tag{6}$$

$$\sum_{i=1}^{m} x_{ij} = 1, \forall i \tag{7}$$

$$x_{ij} \in \{0, 1\}, \forall i, j \tag{8}$$

Eq. (5) aims to minimize both the average end-to-end response time and the variance in the controller loads. Eqs. (6)-(8) represent the constraints. As mentioned before, a) for each controller, the sum of the request rates from all placed switches should be less than the controller's processing rate μ; b) each switch can only be connected to one controller at a time; and c) the controller placement decision is encoded in a matrix x_{ij} with binary elements.

While the above formulation only captures the static scenarios, dynamic scenarios due to changes in network conditions could arise that include the following:

Adding new switches: When the number of the switches increases and causes additional load from the new switches to the system.

Deleting existing switches: When the number of switches decreases thereby causing unbalanced loads among the controllers. This also includes switch failures.

Increasing request rates: When some switches' request rates increase thereby causing performance degradation or unbalanced loads among the controllers.

Decreasing request rates: When some switches' request rates decrease thereby causing unbalanced loads among the controllers.

Due to these dynamic changes in the network, an existing placement may no longer be able to provide good system-level and service-level performance thereby requiring switch migration decisions. To adapt to these dynamic fluctuations, we also consider reducing the switch migration cost formulated in Eq. (9) as another metric, where $X^0 = \{x_{ij}^0\}$ denotes the new decision after switch migration while $X = \{x_{ij}\}$ denotes the old decision before migration.

$$G(X, X^0) = \frac{\sum_{i=1}^{n} \sum_{j=1}^{m} (x_{ij}^0 - x_{ij})^2 \lambda_i}{\sum_{i=1}^{n} \sum_{j=1}^{m} x_{ij} \lambda_i} \tag{9}$$

It is defined as the fraction of the sum of affected switch request rates and the sum of all the switch request rates. The affected switches can be the new switches that are added to the system or the reconnected switches. In the latter case, the switches are those that are still in the system but need to be reconnected to other controllers for load balancing. This metric measures the impact on the switch requests during switch migration.

IV. Heuristic Algorithms for Switch Migration Problem

A prior study has proven that SMP is an NP-hard problem [31]. Hence, to solve our formulated switch migration optimization problem effectively at run-time as part of an autonomic self-healing and self-adapting systems software, we developed three heuristic algorithms for balancing the controller loads under dynamic traffic conditions in IoT. This section describes these heuristics in detail.

A. Genetic Algorithm (GA)

GA is an adaptive heuristic search strategy inspired by natural evolution [32]. GA is designed to simulate a natural selection by introducing some random genetic operators. Selection, crossover, and mutation are the main operators of GA. Algorithm 1 shows the pseudocode of GA while the pipeline of GA is illustrated in Figure 3 with a simple example.

The algorithm starts from a population of a randomly generated controller-to-switch assignments and the number of generations. In each generation, k assignments will be randomly chosen to have a tournament. Tournament Selection (TS) is a selection strategy used for selecting k switch-to-controller assignments from a possible assignments population set as shown in Figure 3. The winner of the tournament, which has the best fitness by applying Eq. (5), will be chosen as a parent. Then, the parents will participate in the crossover and

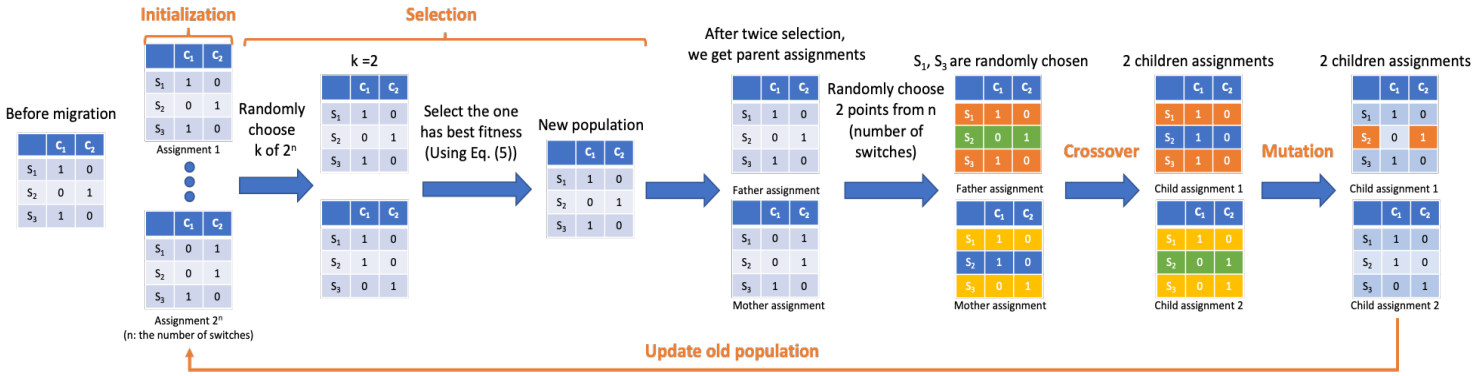

Fig. 3: Illustration of the GA pipeline with a simple example.

the mutation to generate child assignments. The crossover and the mutation are presented in Figure 3.

In the crossover, we will randomly choose two points in both parent assignments and swap the switch assignments between the two points. In the mutation, we randomly generate a number for each child assignment. If this random number is less than the mutation rate r, we will choose one random switch in the child assignment and reassign it to a random controller. The population will be updated by the child assignments by the function UPDATE_POPULATION_SET(X_0, X_{best}, X_{c1}, X_{c2}, Pn, p) in Algorithm 1, where X_0 represents the initial assignment; X_{best} represents the assignment which has the best fitness so far; X_{c1}, X_{c2} are two children assignments; Pn is the population set, and p is the number of possible assignments. Both Eq. (5) and Eq. (9) will be applied within the function. The algorithm terminates when the number of generations is met. In our evaluation, we set the population size to be $3mn$ and the number of generations to be $6mn$. Also, the number of randomly selected assignments in the tournament is set as $k = m$ and the mutation rate is set as $r = 0.02$. The simulation results show that these parameters lead to fast convergence and GA is able to provide a good solution to the problem.

B. Simulated Annealing Algorithm (SA)

SA is another efficient global search optimization algorithm [33]. It is inspired by the annealing process in metallurgy, where a material is heated to a high temperature quickly and cooled slowly. Algorithm 2 depicts the pseudocode of SA used to solve our problem.

The algorithm starts with a high temperature t_i, a final temperature t_f, a temperature decrease rate r, an iteration number, and an initial switch-to-controller assignment X_0. In each basic iteration, the algorithm considers its neighboring switch-to-controller assignment, which is generated by the function CREATE_NEW_ASSIGNMENT(X_0) in Algorithm 2. This function will generate a neighboring assignment by randomly swapping the switch-to-controller connections in the old assignment. Then, the algorithm will evaluate the neighboring assignment by using Eq. (5). If the neighboring assignment has better performance, the algorithm will accept it immediately and update the best assignment so far by applying Eq. (9); otherwise, the algorithm will accept the neighboring assignment according to the Metropolis rule of probability [34], as indicated in line 22 of Algorithm 2. This will enable SA to jump out of a local optimum where most algorithms get stuck. When the iteration number is met, the algorithm will decrease the temperature according to r as shown in line 26 of Algorithm 2. In our evaluation, we set the high temperature to be $10m$ and the final temperature to be 10^{-8}. The iteration number is set as $iteration = n$ and the decrease rate is set as $r = 0.98$. The results of evaluation show that SA is able to efficiently find an approximate global optimum.

C. Modified Greedy Algorithm (MG)

Finally, we present a modified greedy algorithm to adapt to the dynamic nature of the network. The algorithm is a modification of well-known greedy algorithm for static load balancing [35] and aims to minimize the switch migration cost while reducing the end-to-end response time and the variance of controller loads. Algorithm 3 shows the pseudocode for this modified greedy algorithm. MG is automatically triggered whenever any controller's utilization exceeds 80%.

The algorithm works again in an iterative manner given a load-balancing threshold and a "maxIteration" number. In each iteration, we first find a controller c_j that has the maximum load as shown in Algorithm 3 line 6. Then, among all the switches that connect to c_j, we find a switch s_{jr} with the minimum load so as to reduce the switch migration cost. Finally, we migrate s_{jr} from c_j to a controller c_j^0 that has the minimum load among all controllers to achieve load balancing. We define δ to be the average absolute difference between the controller loads and the average load, as shown in line 11 of Algorithm 3, and it characterizes the uneven distribution of the switch request rates. The maximum number of iterations is used to avoid an infinite loop, which can happen when multiple placements achieve the same objectives. The above steps are then repeated until either δ falls below the threshold or the maximum number of iterations is reached. In the evaluation, we set the value of the threshold to be $3\%\mu$ and the maximum number of iterations to be n.

Algorithm 1: Genetic Algorithm

Input: A initial controller to switch assignment in X_0 matrix, the number of population p, the number of generation g, a population set $Pn = \{X_1, X_2, \ldots, X_p\}$ of p possible controller to switch assignments, a mutation rate mr

Output: A controller to switch assignment in X_{best} matrix

Function `Update_Population_Set(`X_0, X_{best}, X_{c1}, X_{c2}, Pn, p`):`
  $Pn.append(X_{c1})$;
  $Pn.append(X_{c2})$;
  while $len(Pn) \neq p$ **do**
    Remove a controller to switch assignment, which has the worst fitness from Pn by applying Eq. (5);
  end
  Find the controller to switch assignment X, which has the best fitness from Pn by applying Eq. (5);
  if $X_{best} == X_0$ **then**
    $X_{best} \leftarrow X$;
  end
  else if $G(X_0, X) \leq G(X_0, X_{best})$ **then**
    $X_{best} \leftarrow X$;
  end
  return X_{best};
End Function
Function `Main(`X_0, Pn, p, g, mr`):`
  $X_{best} = X_0$;
  for $j \leftarrow 1$ **to** g **do**
    $children\_size = 0$;
    while $children\_size < p$ **do**
      Select X_{father}, X_{mother} by a tournament process;
      Generate X_{c1}, X_{c2} by applying a crossover operator and a mutation operator;
      $Pn, X_{best} \leftarrow$ UPDATE_POPULATION_SET(X_0, X_{best}, X_{c1}, X_{c2}, Pn, p);
      $children\_size = children\_size + 2$;
    end
  end
  return X_{best};
End Function

Algorithm 2: Simulated Annealing Algorithm

Input: A initial temperature t_i, a final temperature t_f, the current temperature t_c, a iteration number $iteration$, a temperature decrease rate r, a initial controller to switch assignment in X_0 matrix

Output: A controller to switch assignment in X_{best} matrix

Function `Create_New_Assignment(`X_0`):`
  $temp = random.randint()$;
  if $temp\%2 == 1$ **then**
    Randomly swap 2 rows in X_0;
  else
    Randomly swap 3 rows in X_0 in cyclic form;
  end
  return X_0;
End Function
Function `Main(`t_i, t_f, t_c, $iteration$, r, X_0`):`
  $t_c = t_i$;
  $X_{best} = X_0$;
  while $t_c > t_f$ **do**
    for $j \leftarrow 1$ **to** $iteration$ **do**
      $X_{new} \leftarrow$ CREATE_NEW_ASSIGNMENT(X_0);
      if $F(X_{new}) < F(X_0)$ **then**
        $X_0 \leftarrow X_{new}$;
        if $X_{best} == X_0$ **then**
          $X_{best} \leftarrow X_{new}$;
        end
        else if $G(X_0, X_{new}) \leq G(X_0, X_{best})$ **then**
          $X_{best} \leftarrow X_{new}$;
        end
      end
      else if $exp(F(X_0) - F(X_{new})/t_c) > random()$ **then**
        $X_0 \leftarrow X_{new}$;
      end
    end
    $t = t * r$;
  end
  return X_{best};
End Function

V. Systems Software Architecture

A self-healing/self-adaptive solution will require effective monitoring and dissemination of instrumented properties to the decision making logic. To that end, our autonomic adaptive approach for controller load balancing is codified in a systems software whose architecture is depicted in Figure 4. Effectively it illustrates the Monitor-Analyze-Plan-Execute (MAPE) loop manifested in our system design that are key to developing an autonomous self-healing/self-adapting system. The system's software architecture comprises three modules: Pub/Sub Messaging, a Network Monitor, and a Decision Maker. The roles and responsibilities of each are explained next.

Pub/Sub Messaging: We use Pub/Sub Messaging, which is a lightweight asynchronous messaging service to communicate messages between controllers and Network Monitor. The controllers that manage flows from the data plane serve as the publishers. They create a topic

Algorithm 3: Modified Greedy Algorithm

Input: A set $C = \{c_1, c_2, \ldots, c_m\}$ of m controllers, a set $S = \{s_1, s_2, \ldots, s_n\}$ of n switches, request rates of all switches $\Lambda = \{\lambda_1, \lambda_2, \ldots, \lambda_n\}$, existing loads of all controllers $\Theta = \{\theta_1, \theta_2, \ldots, \theta_m\}$ and their average load $\overline{\theta}$, a load-balancing threshold, a maxIteration number, an initial controller to switch assignment in X_0 matrix, $X_0 = \{x_{00}, x_{01}, \ldots, x_{nm}\}$

Output: An adapted controller to switch assignment in X_{best} matrix

Function `Main`(C, m, S, n, Λ, Θ, $\overline{\theta}$ $counter$, $maxIteration$, X_0)**:**
- $\delta \leftarrow INF$;
- $counter \leftarrow 0$;
- **while** $\delta > threshold$ *and* $counter < maxIteration$ **do**
  - Find controller c_j with the maximum load from C;
  - Find controller $c_{j'}$ with the minimum load from C;
  - Find the set S_j of switches that connect to c_j and the switch $s_{jr} \in S_j$ with the minimum load;
  - In X_0, set $x_{jr,j} \leftarrow 0$ by disconnecting s_{jr} from c_j;
  - In X_0, set $x_{jr,j'} \leftarrow 1$ by connecting s_{jr} to $c_{j'}$;
  - Update the controller loads $\theta_j \leftarrow \theta_j - \lambda_{jr}$ and $\theta_{j'} \leftarrow \theta_{j'} + \lambda_{jr}$;
  - Compute $\delta \leftarrow \frac{1}{m} \sum_{j=1}^{m} |\theta_j - \overline{\theta}|$;
  - Increment counter++;
- **end**
- $X_{best} \leftarrow X_0$;
- **return** X_{best};

End Function

with an IP address and a port number and keep sending the instrumented network data to a broker. The network data includes topology information, switch request rates, controller packet-in rates, and controller processing time. After receiving the network data, the broker will forward the data to the Network Monitor (acting as a subscriber), which subscribes to the topic created by the publishers.

Network Monitor: After receiving the data from the broker, the Network Monitor running in the application plane is able to generate a controller to switch assignment in X_i matrix and store the switch request rates in λ. Then, the Network Monitor will pass all the network-related parameters to the Decision Maker.

Decision Maker: The Decision Maker can be configured to execute one of three alternative heuristic algorithms that we developed corresponding to our optimization problem: Genetic Algorithm, Simulated Annealing Algorithm, and Modified Greedy Algorithm. Each algorithm runs independent of each other. The output of the algorithms is a new controller-to-switch assignment that will be compared with the initial controller-to-switch assignment to create a switch migration plan.

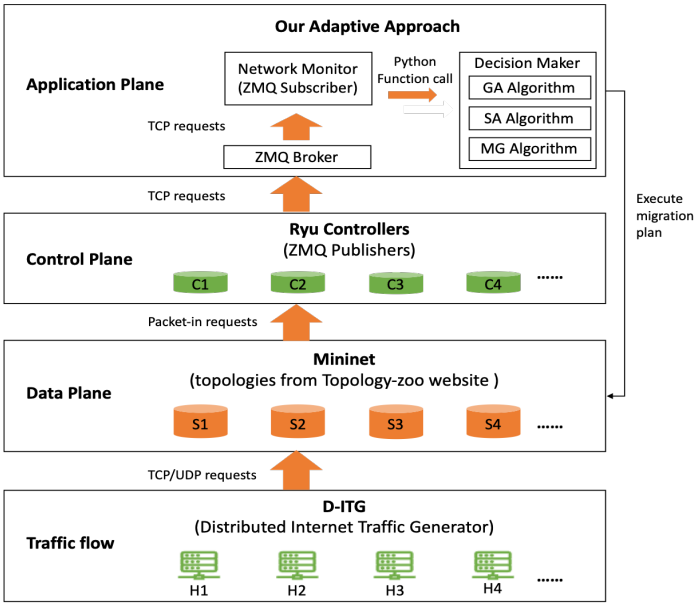

Fig. 4: MAPE loop in our systems architecture.

VI. Empirical Evaluation

Our experiment design used to evaluate the efficacy of our techniques is depicted in Figure 4. We implemented our algorithms in Python[1] and evaluated their performance in Mininet [36], which provides a simple and straightforward network emulation testbed for developing and evaluating SDN applications. We use the Ryu controller [37] as our Mininet remote controller. We implemented our Pub/Sub Messaging module using the ZeroMQ PUB/SUB pattern [38]. The topology we have used is the real internet service topology called ARN (nodes: 30, edges: 29) available from the Internet Topology Zoo [39].

We use the Distributed Internet Traffic Generator (D-ITG) [40] to generate traffic flows on the host side. We focus mainly on the Packet-in event on the controller side, which consumes more controller resources than other flow events. The traffic flows from D-ITG follow a Poisson distribution. We evaluated our proposed algorithms in the application plane by using matrices to solve minimization goals that are defined in Eq. (5) and Eq. (9). We also compared our heuristic algorithms with the Switch Migration-Based Decision-Making Scheme (SMDM), which applied a switch migration algorithm based on the greedy method proposed by Wang et al. [16]. Their approach also aims at balancing the load of controllers in a efficient manner, thereby improving the network performance.

[1]The implementation is publicly available at https://github.com/minziran/Switch-Migration-Problem

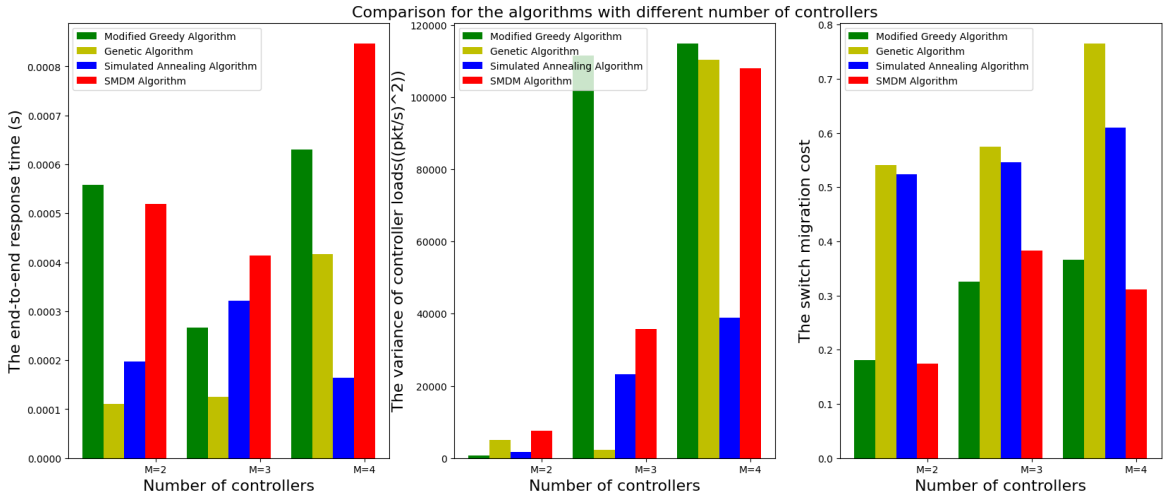

Fig. 5: Comparison for the algorithms with different number of controllers.

They presented a numerical evaluation by using Mininet emulator.

A. End-to-End Response Time

Figure 5 shows the end-to-end response time in the system with different number of controllers. The number of controllers (m) is varied from 2 to 4. We run each simulation for 15 mins and the overall switch request rate is fixed. The y-axis indicates the average end-to-end response time in the system. The result shows that GA and SA outperform the greedy-based algorithm MG as well as SMDM. Compared to traditional optimizing algorithms, GA and SA enable a partial random process to create offspring or neighbor states, which helps avoid getting stuck in local optima.

Figure 6 shows the end-to-end response time with the overall switch request rate increasing. The simulation keeps running for 15 mins and the overall request rate increases 20% every 5 mins. The running result shows that compared to SMDM, GA and SA are more adaptive to network traffic.

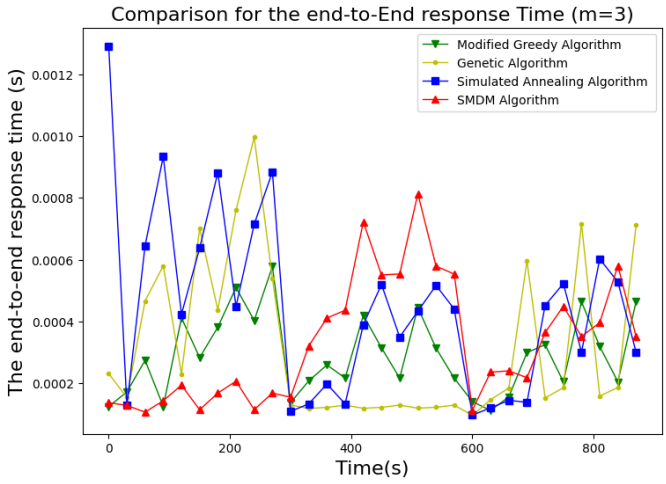

Fig. 6: The end-to-end response time in the system ($m = 3$).

B. Load Balancing

Figure 5 also shows the variance of controller loads with increasing number of controllers. SA has an excellent performance in load balancing while minimizing the end-to-end response time in the system. Figure 7 indicates that GA and SA are also capable of balancing the controller loads while the overall switch request rate is increasing. In GA and SA, we accept new offspring or neighbor states by using Eq. (5), which aims at minimizing both the end-to-end response time and the variance of controller loads. The result of Figure 7 also shows that the performance of MG and SMDM is not stable while experiencing the network traffic.

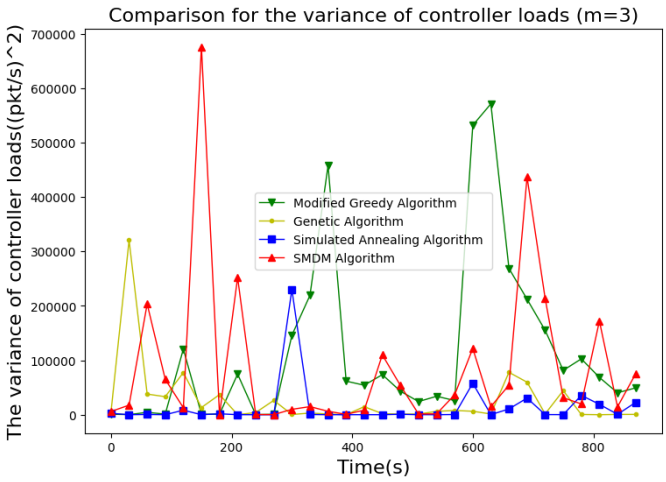

Fig. 7: The variance of controller loads ($m = 3$).

C. Switch Migration Cost

With increasing number of controllers, MG and SMDM have lower switch migration costs as shown in Figure 8. Moreover, MG outperforms SMDM in terms of switch migration cost when the overall switch request rates increase.

MG demonstrates better performance by always migrating the switch with the lowest request rate. Although GA and SA have higher migration costs, they can efficiently find a switch migration plan with minimum end-to-end response time and variance of controller loads.

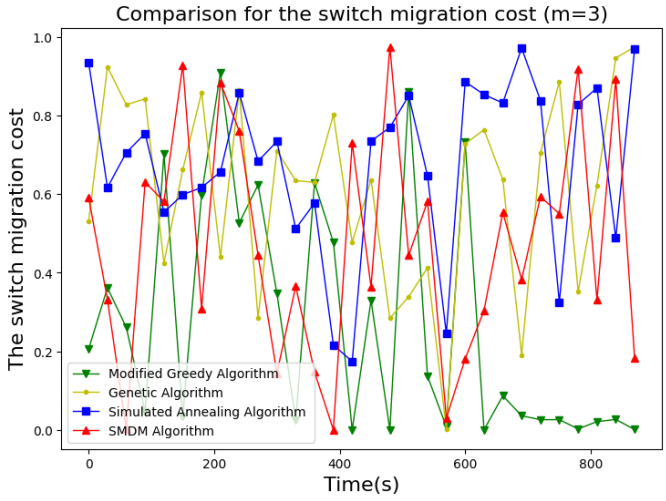

Fig. 8: The average switch migration cost ($m = 3$).

D. Performance Trade-offs

Considering the different performance of our switch migration heuristics, we present the trade-offs among our algorithms in Figure 9 and Figure 10. GA and SA show better performance in minimizing the variance in the controller loads while having higher migration costs. With the overall request rate increasing, MG has the lowest migration cost while the requests are experiencing longer processing time in the controller.

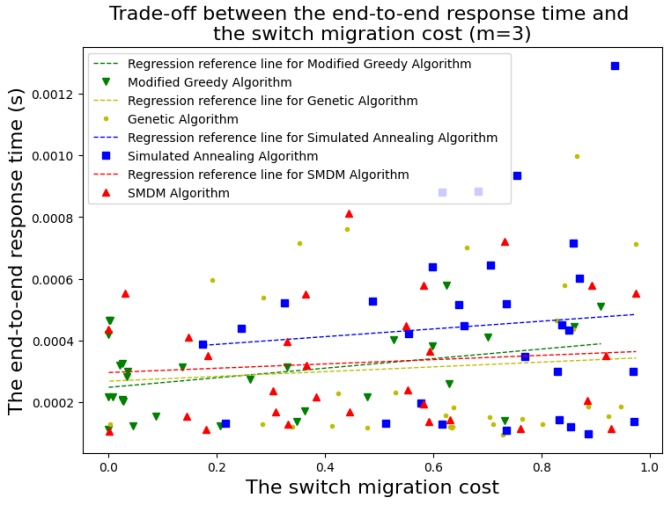

Fig. 9: The average switch migration cost with the end-to-end response time increasing ($m = 3$).

E. Summary

Overall, we observed that our heuristic algorithms exhibit good performance when considering the end-to-end response time, the variance in controller loads, and the switch migration cost. Our result indicates that GA and SA are excellent decisions for the IoT network, which experiences an increasing traffic load while MG helps the large-scale IoT based network. Essentially, the results show they are effective approaches for solving SMP while adapting to dynamic network conditions.

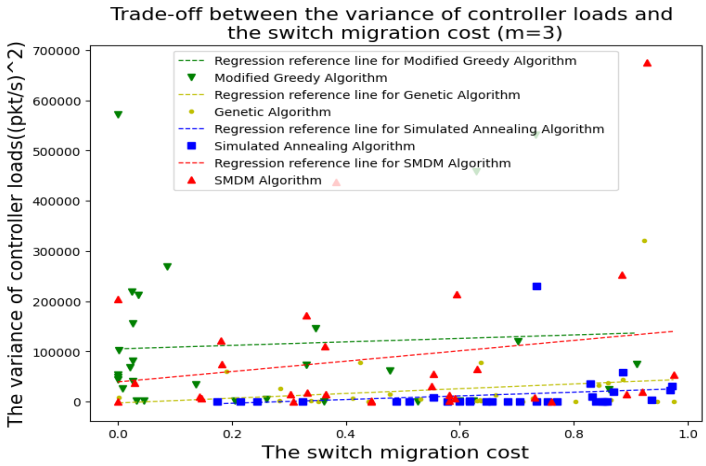

Fig. 10: The average switch migration cost with variance increasing ($m = 3$).

VII. Conclusions and Future Work

This paper presents an autonomic approach to deal with the multi-objective dynamic Switch Migration Problem (SMP) for software defined networks (SDN) in IoT. Compared with the previous single-objective research efforts, our study considered not only the control latency between controller and switch but also runtime controller loads, load balancing among controllers and the switch migration costs in the system. Moreover, compared with the previous multi-objective research efforts, our algorithm is able to adapt to dynamic changes in IoT. The NP-hard nature of the optimization problem and the need for effective runtime solutions motivates the heuristics developed in this paper based on genetic algorithms (GA), simulated annealing (SA) and modified greedy (MG). Simulation results conducted using Mininet show that the GA and SA are able to balance the controller loads while minimizing the end-to-end response time in the system. Moreover, MG can significantly reduce the switch migration cost under dynamic traffic changes.

A. Discussion

Considering the different strengths of our proposed algorithms, we could potentially swap our heuristics to adapt to these dynamic changes. Besides, when we scale our architecture to very large and distributed IoT networks, we may need a hierarchical/partitioned solution and multiple MAPE loops. Then, we will manage the network monitor state to handle the consistency issues.

B. Future Work

There are several potential extensions to this work. First, we will develop our approach for a larger network topology than the one we considered in this work. Second, we will consider more objectives that play pivotal roles in IoT, such as security, cost and energy savings. Finally, we will expand the application domain from wired networks to the fifth generation (5G) networks, where the SDN technology is widely applied to build different types of network slicing based on different user requirements.

REFERENCES

[1] Y. B. Zikria, R. Ali, M. K. Afzal, and S. W. Kim, "Next-generation internet of things (iot): Opportunities, challenges, and solutions," 2021.

[2] P. Middleton, "Forecast analysis: Internet of things — endpoints, worldwide, 2017 update." https://www.gartner.com/en/documents/3841268.

[3] CISCO, "Internet of things." https://www.cisco.com/c/en/us/products/collateral/se/internet-of-things/at-a-glance-c45-731471.pdf?dtid=osscdc000283.

[4] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM computer communication review*, vol. 38, no. 2, pp. 69–74, 2008.

[5] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker, "Ethane: Taking control of the enterprise," *ACM SIGCOMM computer communication review*, vol. 37, no. 4, pp. 1–12, 2007.

[6] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "Nox: towards an operating system for networks," *ACM SIGCOMM computer communication review*, vol. 38, no. 3, pp. 105–110, 2008.

[7] Y. Cao, J. Guo, and Y. Wu, "Sdn enabled content distribution in vehicular networks," in *Fourth edition of the International Conference on the Innovative Computing Technology (INTECH 2014)*, pp. 164–169, IEEE, 2014.

[8] M. Azizian, S. Cherkaoui, and A. S. Hafid, "Vehicle software updates distribution with sdn and cloud computing," *IEEE Communications Magazine*, vol. 55, no. 8, pp. 74–79, 2017.

[9] R. Muñoz, R. Vilalta, N. Yoshikane, R. Casellas, R. Martínez, T. Tsuritani, and I. Morita, "Integration of iot, transport sdn, and edge/cloud computing for dynamic distribution of iot analytics and efficient use of network resources," *Journal of Lightwave Technology*, vol. 36, no. 7, pp. 1420–1428, 2018.

[10] B. Heller, R. Sherwood, and N. McKeown, "The controller placement problem," *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 4, pp. 473–478, 2012.

[11] G. Wang, Y. Zhao, J. Huang, and Y. Wu, "An effective approach to controller placement in software defined wide area networks," *IEEE Transactions on Network and Service Management*, vol. 15, no. 1, pp. 344–355, 2017.

[12] B. Zhang, X. Wang, L. Ma, and M. Huang, "Optimal controller placement problem in internet-oriented software defined network," in *2016 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC)*, pp. 481–488, IEEE, 2016.

[13] M. Malboubi, L. Wang, C.-N. Chuah, and P. Sharma, "Intelligent sdn based traffic (de) aggregation and measurement paradigm (istamp)," in *IEEE INFOCOM 2014-IEEE Conference on Computer Communications*, pp. 934–942, IEEE, 2014.

[14] W. Queiroz, M. A. Capretz, and M. Dantas, "An approach for sdn traffic monitoring based on big data techniques," *Journal of Network and Computer Applications*, vol. 131, pp. 28–39, 2019.

[15] A. S. da Silva, J. A. Wickboldt, L. Z. Granville, and A. Schaeffer-Filho, "Atlantic: A framework for anomaly traffic detection, classification, and mitigation in sdn," in *NOMS 2016-2016 IEEE/IFIP Network Operations and Management Symposium*, pp. 27–35, IEEE, 2016.

[16] C. Wang, B. Hu, S. Chen, D. Li, and B. Liu, "A switch migration-based decision-making scheme for balancing load in sdn," *IEEE Access*, vol. 5, pp. 4537–4544, 2017.

[17] A. Dixit, F. Hao, S. Mukherjee, T. Lakshman, and R. R. Kompella, "Elasticon; an elastic distributed sdn controller," in *2014 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, pp. 17–27, IEEE, 2014.

[18] M. Belyaev and S. Gaivoronski, "Towards load balancing in sdn-networks during ddos-attacks," in *2014 International Science and Technology Conference (Modern Networking Technologies)(MoNeTeC)*, pp. 1–6, IEEE, 2014.

[19] M. Jarschel, S. Oechsner, D. Schlosser, R. Pries, S. Goll, and P. Tran-Gia, "Modeling and performance evaluation of an openflow architecture," in *2011 23rd International Teletraffic Congress (ITC)*, pp. 1–7, IEEE, 2011.

[20] A. Leon-Garcia, *(2008). Probability, statistics, and random processes for electrical engineering.Harlow: Prentice Hall.* 2008.

[21] A. A. Neghabi, N. Jafari Navimipour, M. Hosseinzadeh, and A. Rezaee, "Load balancing mechanisms in the software defined networks: A systematic and comprehensive review of the literature," *IEEE Access*, vol. 6, pp. 14159–14178, 2018.

[22] T. Das, V. Sridharan, and M. Gurusamy, "A survey on controller placement in sdn," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 1, pp. 472–503, 2019.

[23] M. Guo and P. Bhattacharya, "Controller placement for improving resilience of software-defined networks," in *2013 Fourth International Conference on Networking and Distributed Computing*, pp. 23–27, 2013.

[24] G. Yao, J. Bi, Y. Li, and L. Guo, "On the capacitated controller placement problem in software defined networks," *IEEE Communications Letters*, vol. 18, no. 8, pp. 1339–1342, 2014.

[25] H. Bo, W. Youke, W. Chuan'an, and W. Ying, "The controller placement problem for software-defined networks," in *2016 2nd IEEE International Conference on Computer and Communications (ICCC)*, pp. 2435–2439, IEEE, 2016.

[26] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood, "On controller performance in software-defined networks," in *2nd {USENIX} Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (Hot-ICE 12)*, 2012.

[27] T. Hu, Z. Guo, P. Yi, T. Baker, and J. Lan, "Multi-controller based software-defined networking: A survey," *IEEE Access*, vol. 6, pp. 15980–15996, 2018.

[28] M. D. Priya, T. Suganya, A. C. J. Malar, E. Dhivyaprabha, P. K. Prasad, and L. V. Vardhan, "An efficient scheduling algorithm for sensor-based iot networks," in *Inventive communication and computational technologies*, pp. 1323–1331, Springer, 2020.

[29] M. S. Munir, S. F. Abedin, M. G. R. Alam, N. H. Tran, and C. S. Hong, "Intelligent service fulfillment for software defined networks in smart city," in *2018 International Conference on Information Networking (ICOIN)*, pp. 516–521, 2018.

[30] F. Metzger, T. Hoßfeld, A. Bauer, S. Kounev, and P. E. Heegaard, "Modeling of aggregated iot traffic and its application to an iot cloud," *Proceedings of the IEEE*, vol. 107, no. 4, pp. 679–694, 2019.

[31] X. Qin, W. Zhang, W. Wang, J. Wei, *et al.*, "Enabling elasticity of key-value stores in the cloud using cost-aware live data migration," *J. Softw.*, vol. 24, no. 6, pp. 1403–1417, 2014.

[32] M. Mitchell, *An introduction to genetic algorithms*. MIT press, 1998.

[33] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *science*, vol. 220, no. 4598, pp. 671–680, 1983.

[34] A. Gelman, G. O. Roberts, W. R. Gilks, *et al.*, "Efficient metropolis jumping rules," *Bayesian statistics*, vol. 5, no. 599-608, p. 42, 1996.

[35] C. H. Papadimitriou and K. Steiglitz, *Combinatorial optimization: algorithms and complexity*. Courier Corporation, 1998.

[36] K. Kaur, J. Singh, and N. S. Ghumman, "Mininet as software defined networking testing platform," in *International Conference on Communication, Computing & Systems (ICCCS)*, pp. 139–42, 2014.

[37] Ryu, "Ryu sdn framework." https://ryu-sdn.org.

[38] P. Hintjens, *ZeroMQ: messaging for many applications*. " O'Reilly Media, Inc.", 2013.

[39] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The internet topology zoo," *IEEE Journal on Selected Areas in Communications*, vol. 29, no. 9, pp. 1765–1775, 2011.

[40] S. Avallone, S. Guadagno, D. Emma, A. Pescapè, and G. Ventre, "D-itg distributed internet traffic generator," in *First International Conference on the Quantitative Evaluation of Systems, 2004. QEST 2004. Proceedings.*, pp. 316–317, IEEE, 2004.