2023 IEEE Symposium Series on Computational Intelligence (SSCI)
Mexico City, Mexico. December 5-8, 2023

Algorithm Package of AI-driven SDN Controller-Switch Load Balancing Strategies

Mahzabeen Emu[1], Md Yeakub Hassan[2], Zubair Md Fadlullah[3], Salimur Choudhury[4]
[1,4] School of Computing, Queen's University, ON, Canada
[2] Siemens Digital Industries Software, SK, Canada
[3] Department of Computer Science, Western University, ON, Canada
Emails: {20me21, s.choudhury}@queensu.ca, md-yeakub.hassan@siemens.com, zubair.fadlullah@uwo.ca

***Abstract*—Recently, Software-Defined Networking (SDN) is receiving much research attention due to its ability to decouple the data plane from the control architecture by associating the network switches to one (centralized) or more (distributed) controller(s). Traditionally, switches are assigned to the controllers in a static manner which results in under-utilization of the resources of the controllers and increased response delays to user requests. In this paper, we consider a practical load-balancing and agile scenario by formulating the dynamic associations of switches and controllers as an NP-hard optimization problem to minimize the maximum resource utilization of the controllers. Therefore, we propose an Ant Colony Optimization (ACO)-based algorithm to deal with the aforementioned request satisfiability issue in large SDN systems in polynomial-time. Furthermore, we envision a hybrid deep learning model consisting of Convolutional Neural Network (CNN) and Gated Recurrent Unit (GRU) structures to achieve near-optimal resource utilization for real-time SDN applications. Experimental results demonstrate that our customized CNN-GRU model outperforms the other techniques in terms of resource utilization ($15\% - 45\%$ optimality gap) within a significantly reduced computational running time (≤ 0.1s).**

***Index Terms*—Software-Defined Networking, Resource utilization, Convolutional Neural Network, Gated Recurrent Unit.**

I. Introduction

With the recent proliferation of digital communications, the limitations of conventional networks continue to be identified. A major shortcoming of legacy network devices is how both control and data planes are handled together [1], resulting in network manageability, extensibility, and flexibility issues. Aiming to resolve such issues, the concept of Software-Defined Networking (SDN) emerged where the control and data planes are decoupled using a centralized controller to handle all the operations [1]. However, for a large-scale SDN, a single controller cannot handle the growing number of requests from the data plane [1], and suffers from a single-point-of-failure contributing to network service disruption [1]. To alleviate this problem, distributed controllers have been considered to manage the SDN control plane functionalities [2], where switches are statically allocated to a single or multiple controllers [2]. However, network traffic changes dynamically, both spatially and temporally. Consequently, some controllers may experience a high traffic load and reach the maximum resource usage, while others might remain idle (i.e., under-utilized). Moreover, due to the static mapping of switches to controllers, the response time of controllers dramatically increases. Hence, dynamic mapping of switches to the controllers is necessary [1] for improving the utilization of resources while balancing the loads on the controller and getting better response time.

Several research works have been focused on the Dynamic Controller Assignment (DCA) for minimizing the message overhead, response time, or balancing the load among the controllers [2]–[6]. AI and machine learning approaches are being used to several approach SDN applications [7], [8]. In these existing works, the controller switch assignment creates an imbalance in the usage of the controllers' resources. Therefore, it can decrease the network's overall performance with a high response time. Moreover, the existing load balancing strategies can have higher communication overhead [9], and response time [10]. To eliminate these shortcomings, every controller needs to maintain a fixed processing capacity, and the maximum resource utilization of the controllers must be minimized as low as possible. While Filali et al. [11] proposed a one-to-many matching algorithm for the DCA problem, we show that this algorithm cannot return any solution for our optimization problem even though a solution exists. Finally, we explored a greedy approach for the DCA problem and elucidated that the algorithm may return no solution even if a solution exists. The motivation for our study comes from the limitations of existing approaches related to Assignment (DCA) solutions. The limitations of the existing approaches have been mentioned elaborately under Section III, which are further addressed by our proposed approaches. For example, MSDA and greedy (existing approaches used for similar DCA problems) MSDA may return no solution for an instance even though a solution may exist. Thus, the service request may not be satisfied even though the solution exists. However, our proposed ACO algorithm initially attempts to find a random feasible solution in an exhaustive manner and iteratively improves upon the solution, which eliminates the occurrences of request failure even when solutions exist. Furthermore, our proposed hybrid AI model can guarantee real-time service delivery for SDN infrastructure. The major contributions of this paper have been outlined as follows:

a) Although DCA resource allocation problem exists in the

literature, we have formulated a novel ILP model to minimize the maximum resource utilization for DCA optimization in this study with the intent of effective load balancing. We prove that the defined optimization problem is NP-hard, and the decision version of the problem is also NP-complete.

b) We apply a meta-heuristics based algorithm Ant Colony Optimization (ACO), which is capable of satisfying 100% request, unlike the algorithms existing in the literature.

c) Furthermore, we investigate a data-driven approach by proposing a hybrid deep learning model composing Convolutional Neural Networks (CNN) and Gated Recurrent Unit (GRU) to speed up the performance. CNN is an extremely well-known and preferred method for extracting pattern insights from data to deal with classification problems [8]. We have utilized GRU for gaining perception about capacity changing temporal effects in trend prediction. The primary intention of this hybrid model, named CNN-GRU hereafter, is to serve real-time SDN use cases.

All the methodologies proposed in this paper are considered to be packaged together to facilitate the SDN infrastructure by enabling a module selector for load balancing or resource utilization approaches. The service providers are considered responsible for activating the algorithm from the package as per the Quality of Service (QoS) requirements, Service Level Agreement (SLA), and nature of the application. The rest of the paper is structured as follows. The network model and problem formulation is described in Section II. Section III covers some existing approaches to the problem and issues regarding those methodologies. Our proposed AI-driven approaches (ACO and CNN-GRU) are explained in Sections IV and V, respectively. Finally, Section VII concludes the paper.

II. System Model & Problem Formulation

In a data-center network, the physical topology may vary depending on the network infrastructure. However, the communication between controllers and switches can be realized in a two-tier structure between the control and data plane with an algorithm module selector, as shown in Figure 1. Based on the QoS requirements of the application and SLA, algorithm module selector is configured by the provider to apply one of our proposed approaches for controller-switch mapping. In this paper, we consider a physically distributed SDN architecture within a data-center where the control plane comprises a set $C = \{c_1, c_2, \ldots, c_n\}$ of n controllers. The processing capacity of each controller $c_i \in C$ is denoted by α_i with respect to the number of requests c_i can handle in one time unit. Let $S = \{s_1, s_2, \ldots, s_m\}$ be a set of m switches in the data plane. At any time t, the load generated by a switch s_j, i.e., the request demand of s_j, is denoted by $\lambda_j(t)$. We formulate a SDN controller-switch assignment problem for the set C of n controllers and the set S of m switches. At any time t, the assignment between controllers and switches is denoted as a binary $n \times m$ matrix $\mathcal{X}(t)$, where

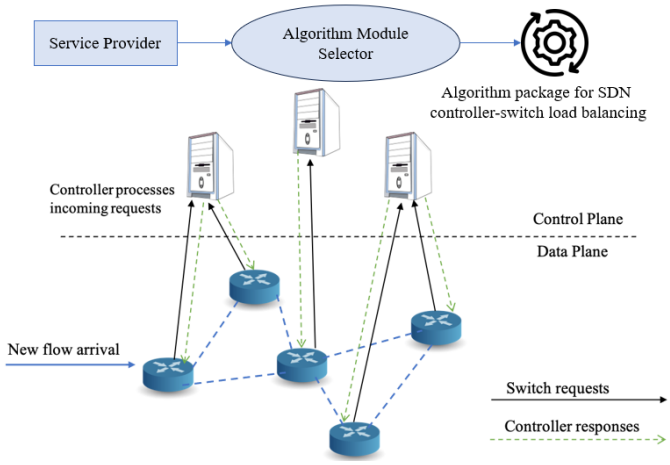


Fig. 1: A data-center network model for SDN deployment with algorithm package configured by service provider

$$x_{ij}(t) = \begin{cases} 1, & \text{switch } s_j \text{ is assigned to the controller } c_i; \\ 0, & \text{otherwise.} \end{cases}$$

We consider the controller response time model where the arrival times of the requests follow a Poisson process with $\lambda_j(t) \leq \alpha_i$, $\forall c_i \in C$. Here, switch requests are aggregated in the connected controller's processing queue. Therefore, at any time t, the load of the controller c_i, denoted by $\theta_i(t)$, is represented as:$\theta_i(t) = \sum_{j=1}^{m} \lambda_j(t) x_{ij}(t)$.

We assume that a controller is modelled as an M/M/1 queue, and the loads generated at the switches are independent [11]. Therefore, according to Little's law, the average sojourn time for the controller c_i is $\frac{1}{\alpha_i - \theta_i(t)}$. In our model, the processing capacity of the controllers may differ from one controller to another. The resource utilization of a controller is the utilization rate of its processing capacity. Hence, at time t, the resource utilization of a controller c_i, denoted by $\mathcal{R}_i(t)$, is represented as:$\mathcal{R}_i(t) = \frac{\theta_i(t)}{\alpha_i}$. In this paper, our primary objective is to minimize the maximum resource utilization in each time slot by assigning controllers to the switches. Hence, for any time slot t, we formulate the following optimization problem:

$$\textbf{Minimize} \quad \arg\max_{i,j} \ \mathcal{R}_i(t) \tag{1}$$

subject to,

$$\theta_i(t) \leq \alpha_i \ ; \qquad \forall \ c_i \in C \tag{2}$$

$$\sum_{j=1}^{m} x_{ij}(t) \geq 0 \ ; \quad \forall \ c_i \in C \tag{3}$$

$$\sum_{i=1}^{n} x_{ij}(t) = 1 \ ; \quad \forall \ s_j \in S \tag{4}$$

$$x_{ij}(t) \in \{0, 1\} \ ; \quad \forall \ c_i \in C \text{ and } \forall \ s_j \in S \tag{5}$$

The constraint in equation (2) ensures that the controller's total load does not exceed that controller's processing capacity. The constraints in equations (3) and (4) ensure that each controller can be associated with any number of switches and every switch maintains connectivity with exactly one controller, respectively. Equation (4) also ensures that all switches are assigned to the controllers. Here, $x_{ij}(t)$ in equation (5) is the decision variable.

In what follows, we show that the SDN controller-switch assignment problem is NP-hard, and we obtain the result by reduction from the load balancing problem, which is NP-hard [12].

Theorem II.1. *The SDN controller-switch assignment problem is NP-hard.*

Proof. Consider any instance of the load balancing problem where $\mathcal{J}$ be a set of m jobs, and $\mathcal{M}$ be a set of n identical machines. The processing time of each job $j \in \mathcal{J}$ is represented as t_j^p. Now, we describe how to construct a corresponding instance of the SDN controller-switch assignment problem. Each job in $\mathcal{J}$ is considered a switch in the data plane, and each machine in $\mathcal{M}$ is considered a controller in the control plane. The load generated by a switch s_j is t_j^p, i.e., the load of each switch is equal to the processing time of the corresponding job in $\mathcal{J}$. We assume that the processing capacity of each controller is $\sum_{j=1}^{m} t_j^p$. Therefore, any controller can be assigned to all the switches. The SDN controller assignment problem's objective is to minimize any controller's maximum resource utilization. Since the controller's processing capacity is the same, the resource utilization of any controller would directly depend on the load generated at the switches managed by that controller. Therefore, each machine's load in the load balancing problem becomes equivalent to the resource utilization of the corresponding controller in the SDN controller-switch assignment problem. In the instance of the load balancing problem, the maximum load of any machine would be minimized if and only the corresponding instance of the SDN controller-switch assignment problem minimizes the maximum resource utilization of any controller. □

We also find that the decision version of the SDN controller-switch assignment problem is NP-complete. Given a set S of m switches and a set C of n controllers, the decision version of the SDN controller-switch assignment problem asks to determine whether the controllers in C can be assigned to all the switches in S. We prove that the decision version of SDN controller-switch assignment problem is NP-complete. We obtain the result by reduction from the decision version of the bin packing problem, which is NP-complete [13].

Theorem II.2. *The decision version of the SDN controller-switch assignment problem is NP-complete.*

Proof. Consider an instance of the decision version of bin packing problem where we are given a set B of n bins, and a set $W = \{w_1, w_2, \ldots, w_m\}$ of m weights. Let Δ be the capacity of each bin. Now, we describe how to construct a corresponding instance of the decision version of the SDN controller-switch assignment problem. Each bin in B is considered as a controller in the control plane. The processing capacity of all the controllers is Δ. Let S be a set of m switches in the data plane and the load generated by a switch $s_j \in S$ is w_j. Since the processing capacity at all the controllers is the same, the resource utilization of any controller would directly depend on the load generated by the switches managed by that controller. Therefore, the total weight of each bin in the decision version of the bin packing problem becomes equivalent to the resource utilization of the corresponding controller in the decision version of the SDN controller-switch assignment problem. In the instance of the decision version of the bin packing problem, all the weights can be packed into D or fewer bins of capacity Δ if and only if the corresponding instance of SDN controller-switch assignment problem can assign the controllers to all the switches. □

III. Exploring Limitations of Existing Approaches

In this section, we first discuss an existing one-to-many matching algorithm for the SDN controller-switch assignment problem. We show that the algorithm can return no solution for an instance even if a solution exists. Then, we provide a greedy approach and show that it also fails to return a solution for an instance even though a solution exists. This approach is intuitively applicable, i.e., natural choices to solve our considered problem. We provide its preliminaries and identify its limitations, which will reinforce the need to devise more effective solutions.

A. One-to-many Matching Algorithm

A one-to-many matching game-based approach, also named the Multi-Stage Deferred Acceptance (MSDA) algorithm, was proposed to dynamically assign the switches among the controllers dynamically [11]. The approach dynamically assigns switches to controllers to minimize the response time. In the MSDA algorithm, each switch chooses one controller as its master controller based on its preference list. Each controller can be associated with multiple switches concerning its processing capacity and a minimum quota. However, in some cases, MSDA algorithm can return no solution for an instance even though a solution exists. Consider an example of an SDN with two controllers $(c_1, c_2,)$, and three switches $(s_1, s_2, s_3,)$. The capacity of c_1 and c_2 are 10 and 4, respectively. The load generated at the switches $s_1, s_2,$ and s_3 are $5, 4, 5$, respectively. The preference list of all the switches is $\{c_1, c_2\}$. Therefore, the algorithm assigns s_1 and s_2 to the controller c_1. Hence, MSDA algorithm cannot find a feasible solution for this instance as it fails to find a controller for s_3. However, an optimal solution exists where controller c_1 is mapped to s_1 and s_3, and c_2 is mapped to s_2.

Lemma III.1. *MSDA algorithm may return no solution for an instance even though a solution may exist.*

B. Greedy Algorithm

We provide a greedy technique that minimizes the maximum resource utilization while ensuring the constraints (2), (3), and (4). Initially, the algorithm sorts all the switches in descending order according to their load. According to the sorted list of switches, we take each switch s_j and calculate each controller's resource utilization rate for s_j by following the equation (II). Then, we choose a controller c_i that has the minimum resource utilization rate among all other controllers. If the capacity of c_i is not exceeding and has the minimum resource utilization rate, then s_j is mapped to c_i. The load of the controller c_i is updated afterward. In some instances, the greedy algorithm may return no solution even though a solution exists. Consider an instance of the SDN controller-switch assignment problem where we have two controllers c_1 and c_2, and four switches s_1, s_2, s_3 and s_4. The processing capacities of c_1 and c_2 are 110 and 11, respectively. The load generated by the switches s_1, s_2, s_3 and s_4 are $5, 6, 10, 100$, respectively. The greedy algorithm cannot find a feasible solution for this instance because the algorithm fails to find a controller for s_1. However, there exists a solution where we can assign the following: $c_1 : s_4, s_3$ and $c_2 : s_2, s_1$.

Lemma III.2. *The greedy algorithm may return no solution for an instance even though a solution may exist.*

IV. Meta-heuristic Controller-Switch Assignment

This section proposes a meta-heuristic Ant Colony Optimization (ACO) approach for the SDN controller switch allocation problem. Meta-heuristics are problem-independent methods, and the master strategy is relatively easy to adapt according to other heuristics. These methods are commonly utilized to invade the search space effectively for generating sub-optimal solutions within polynomial time. he key reason behind proposing this ACO algorithm has been to ensure request satisfiability, dissimilar to previously discussed state-of-the-art algorithms. The ACO algorithm has been designed to generate a feasible initial solution by randomly mapping the SDN controllers to various switches in line number 3 of Algorithm 1. The algorithm initially attempts to find a random but feasible solution in an exhaustive manner and iteratively improves upon the solution quality, which addresses the request satisfiability issue. Hence, instead of the objective function, the initial solution generation scheme instead contributes towards the assurance of request satisfiability (feasible solution) whenever exists. Our proposed ACO algorithm for the SDN controller switch assignment problem has been exhibited in Algorithm 1. At first, the algorithm initializes a set of system parameters and virtual ants in lines 1 and 2, respectively. After that, it generates a feasible initial solution by randomly mapping the SDN controllers to various switches in line number 3. The ACO algorithm initially attempts to find a random feasible solution in an exhaustive manner, which eliminates the occurrences of the mentioned cases in Lemma III.1 and III.2. Thus, the ACO algorithm ensures a feasible solution, and this is one of the strongest motivations for proposing this algorithm. The details of the rest of the algorithm have been discussed in the following subsections.

Algorithm 1: ACO driven SDN controller-switch assignment

```
Input: C, S, λ_j(t), α_i
Result: A set of SDN controller-switch pairs
Initialize system parameters μ, λ, ω_l, ω_g
Initialize a set of ants A
Construct an initial feasible solution by randomly
  assigning switches to controllers
Compute initial value of pheromone ζ_0 using Eq. (6)
Set the value of total_iterations
while (iteration ≤ total_iterations) do
    foreach ant a_z ∈ A do
        foreach switch s_j ∈ S do
            Assign switch s_j ∈ S to a controller c_i ∈ C
              using Eq. (7)
            Update local pheromone using Eq. (8)
    Update value of global pheromone using Eq. (9)
    iteration = iteration + 1
return the set of assigned SDN controller-switch pairs
```

A. Calculation of Initial Pheromone Value

We utilize the randomly generated initial solution to determine the initial pheromone value. In practice, pheromone trails are the deposit of chemical factors functioning as communication signals among the ants. Real ants employ pheromones leading others to food sources while traversing their surroundings. In the SDN controller-switch assignment problem, the pheromone values indicate the possibility of allocating a switch $s_j \in S$ to some controller $c_i \in C$. Being a minimization problem, the lower the pheromone value, the higher the chances are for a specific controller-switch assignment. Thus, the initial pheromone value for this problem is determined by the maximum resource utilization of the corresponding initial solution using the following equation:

$$\zeta_0 = max(R_i) \tag{6}$$

B. Heuristic Value Determination

In collaboration with the pheromone value, the heuristic value plays a crucial role in assigning switches to a controller. The objective function is to minimize the maximum resource utilization. Hence, the heuristic function has been designed in a way that contributes to minimizing the overall objective function. The function for calculating the heuristic value of a switch $s_j \in S$ assigned to controller $c_i \in C$ has been defined as: $\mathcal{H}_{i,j} = \frac{\lambda_j(t)}{\alpha_i}$. This equation makes it possible to ensure that the solutions with less resource utilization will have a lower heuristic value. It is desirable since we ultimately aim to favour the SDN controller-switch mapping solutions that minimize the maximum resource utilization. Moreover, it is noteworthy that we are purposely using lesser heuristic values for more preferable solutions as this is a minimization problem.

C. Controller Selection

An ant $a_z \in A$ selects the best suitable controller for a switch following the pseudo-random-proportional action rule, as mentioned in line 9 of the algorithm. The pseudo-random-proportional rule has been defined as the following.

$$q = \begin{cases} \arg\min_{c_i \in \tilde{C}} \ ([\zeta_{i,j}]^{\mu} \times [\mathcal{H}_{i,j}]^{\lambda}), \text{if } \eta \leq \epsilon \text{ (exploitation)} \\ \text{randomly choose a } c_i \in \tilde{C}, \text{otherwise (exploration)} \end{cases} \tag{7}$$

Here, ϵ represents a random variable uniformly distributed in the $[0, 1]$ range, while η is a pre-defined system parameter. Moreover, η can be any fractional number between the closed range of 0 and 1. The notation $\tilde{C}$ is a set of candidate controllers that has enough available capacity for a switch $s_j \in S$, where $\tilde{C} \subseteq C$. $\zeta_{i,j}$ indicates the pheromone trail values for placing a switch $s_j \in S$ to controller $c_i \in \tilde{C}$. Therefore, an ant $a_z \in A$ assigns a switch to a controller that produces the lowest quantity of $[\zeta_{i,j}]^{\mu} \times [\mathcal{H}_{i,j}]^{\lambda}$, whenever $\eta \leq \epsilon$. Here, μ and λ are coefficients indicating the relative importance of the pheromone trail and heuristic values, respectively. Thus, selecting the most appropriate controller from the candidate set depends on combining heuristic and pheromone trail values. These actions with η probability encourage intensification of the exploitation of the search space. On the contrary, to stimulate exploration, the action rule randomly assigns a controller $c_i \in \tilde{C}$ for a switch $s_j \in S$.

D. Update Procedure for Pheromone Values

As suggested by the line number 10 of algorithm 1, whenever an ant assigns a switch to a controller, it updates the local pheromone trail value with respect to the initial pheromone using the following equation:

$$\zeta_{i,j}(t+1) = \omega_l^{-1} \times \zeta_0 + (1 - \omega_l)^{-1} \times \zeta_{i,j}(t) \tag{8}$$

Here, ω_l upholds the relative importance of historical and current pheromone trail values. Once an ant selects a controller $c_i \in \tilde{C}$ for some switch $s_j \in S$, the corresponding pheromone trail value is increased, making it less desirable for other ants. Hence, this local pheromone update phenomenon promotes the ants to construct diverse solutions in the colony. The global pheromone value update occurs once all the ants construct local optimal solutions according to the line number 11 of algorithm 1. The global solution $\mathcal{G}$ is formed by recording the best solution among the locally produced solutions by all the ants. Equation (9) includes ω_g acting as a system weight parameter.

$$\zeta_{i,j}(t+1) = \omega_g^{-1} \times \Delta\zeta_{i,j} + (1 - \omega_g)^{-1} \times \zeta_{i,j}(t) \tag{9}$$

$\Delta\zeta_{i,j}$ reflects the pheromone value effect for the updated global solution, which is determined by the following equation: $\Delta\zeta_{i,j} = \begin{cases} \zeta_{i,j}, & \text{if } (c_i, s_j) \in \mathcal{G} \\ 0, & \text{otherwise} \end{cases}$. In summary, for each iteration, the historical learning experiences of the ants are utilized for intending to assign switches to SDN controllers optimally.

V. SDN Controller-Switch Assignment Leveraging Deep Learning

Future SDN management researchers are approaching various AI-enabled techniques [14]. The primary intention behind employing an intelligent load balancer for SDN is to improve efficiency, deliver greater convenience, and handle a high number of real-time vertical use cases (e.g., video surveillance, smart city, and industrial automation). Besides, deep learning emerges as a robust design capable of eliminating the lock by design issues with traditional non-programmable load balancers [14]. Hence, we propose a deep learning-empowered hybrid model containing CNN and GRU. We propose hybrid CNN-GRU model with the intention to ensure linear prediction time complexity and effectively serve time-sensitive SDN use cases.

A. Dataset Generation

We utilize an ILP solver to generate labelled datasets for the training process of our proposed hybrid CNN-GRU model. The entire dataset generation and training process has been addressed in Algorithm 2. Firstly, we create a randomly initialized unlabeled system and define features as a quadruple consisting $\mathcal{D} = (\{C_i | 1 \leq i \leq n\}, \{S_j | 1 \leq j \leq m\}, \lambda_j(t), \alpha_i)$ in line number 3 of Algorithm 2. Next, we solve the random system through the ILP solver and exploit the solution to create a labelled dataset. To apply one-hot encoding for the label generation, we define the target variable size for each feature sample same as the number of controllers. Then, we represent the optimal target variable/controller for a feature sample to be 1, and define all others to be 0. Thus, through line numbers $6-9$, we iteratively generate the target variables set $\mathcal{T}$. We also continually update the capacities of controllers for each feature sample in line 10. Likewise, we simulate several feature sets and compile those with target variables to generate a labelled dataset $\mathcal{L}$ for training the CNN-GRU model. The training of the model is performed using the historical dataset captured in the previous subsection. The training is supposed to be done in a sequential manner. Thus, the training model is updated after frequent intervals by training on the new addition of experienced data. By re-training the model, the parameters are adjusted to make accurate decisions. This binding process ensures that the model remains accurate and effective over time. Usually, the training is performed by the SDN provider's server responsible for controller-switch mapping and load-balancing computation.

B. Proposed Hybrid Deep Learning Model

The hybrid CNN-GRU model is composed of one dimensional CNN layer and GRU layer combined by fully connected layers. The GRU layer has been incorporated to capture the capacity update temporal effect of the system into the final deep learning model prediction. We have Rectified Linear Unit (ReLU) as an activation function for all the layers except the output layer that utilizes the softmax function. The filter size of layers gets reduced by a factor. We use the dropout technique

Algorithm 2: Labeled dataset generation and training

Input: number of controllers: n, number of switches: m, *total_simulation_limit*
Output: $\mathcal{L}$
$\mathcal{L} \leftarrow \varnothing$
for $epoch <$ *total_simulation_limit* **do**
  Generate a random system $\mathcal{D} = (\{C_i | 1 \leq i \leq n\}, \{S_j | 1 \leq j \leq m\}, \lambda_j(t), \alpha_i)$
  $\mathcal{T} \leftarrow \varnothing$
  $y_{ij} \leftarrow$ Solve the randomly generated problem using the ILP optimizer
  for $j = 1, 2, ..., m$ **do**
    $i \leftarrow \arg\max_i \ y_{ij}$
    $k \leftarrow \{1, 2, ..., n\} - \{i\}$
    $\mathcal{T} \leftarrow \mathcal{T} \cup \{(y_{ij}, 1), (y_{kj}, 0)\}$
    $\alpha_i \leftarrow \alpha_i - \lambda_j(t)$
  $\mathcal{L} \leftarrow \mathcal{D} \cup \mathcal{T}$
Train CNN-GRU model on labeled dataset $\mathcal{L}$

to avoid overfitting. Then, the model is followed by a 1D max-pooling layer. Finally, we utilize the gradient descent approach to minimize the loss in the training process. The loss function is defined as the mean squared error between the ground truth (ILP solutions) and model predictions. The training of the model is performed using the historical dataset captured in the previous subsection. The training is supposed to done in sequential manner. Thus, the training model is updated after frequent intervals by training on the newly addition data. By re-training the model, the parameters are adjusted to make accurate decisions. This is an important process to make sure the model remains accurate and effective over time.

Algorithm 3: SDN controller-switch mapping using trained CNN-GRU model

Input: $\{C_i | 1 \leq i \leq n\}, \{S_j | 1 \leq j \leq m\}, \lambda_j(t), \alpha_i$
Output: $\mathcal{P}$
$\mathcal{P} \leftarrow \varnothing$
for $j = 1, 2, ..., m$ **do**
  $M \leftarrow$ array of n elements
  for $i = 1, 2, ..., n$ **do**
    if $\lambda_j(t) > \alpha_i$ **then**
      $M_i \leftarrow 0$
    else
      $M_i \leftarrow$ Predict the score between 0.0 and 1.0 using trained CNN-GRU model for assigning s_j to c_i
  $k \leftarrow \arg\max_i \ M_i$
  $\mathcal{P} \leftarrow \mathcal{P} \cup (c_k, s_j)$
  $\alpha_k \leftarrow \alpha_k - \lambda_j(t)$

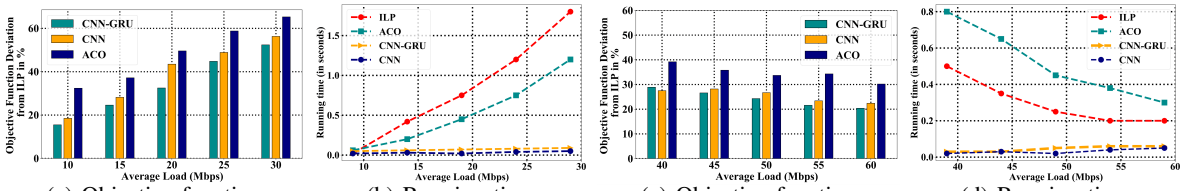
The pre-trained CNN-GRU model is later employed to predict the SDN controller-switch allocation to avoid training time during decision making, as demonstrated in Section VII. As described by the Algorithm 3, for any input system, it records the assigned or predicted controller-switch pairs into a final solution set $\mathcal{P}$. For every feature sample, the trained model generates a confidence score between 0.0 and 1.0 against all the feasible candidate sets of controllers. By a feasible candidate set of controllers, we specify the ones that have enough remaining capacity to handle the load of the corresponding sample or switch. Then, the algorithm retrieves the controller with the highest confidence score produced by the hybrid CNN-GRU model for an individual switch. Finally, the algorithm allocates all the switches to various controllers and generates the solution for the entire system input.

VI. Performance Evaluation

In this paper, we have compared our proposed CNN-GRU (hybrid ML) against standalone CNN to justify the necessity of using hybrid ML models. Moreover, ACO has been considered as one of the meta-heuristic approaches to show the performance trade-off in terms of feasibility rate (request satisfiability rate) and solution quality (optimality gap). To the best of our knowledge, no other hybrid model exists in the literature specifically designed for this problem. Thus, we have not found any other hybrid model to be considered as one of the baselines for comparison purpose. Introducing ACO sacrifices optimality gap and running time compared to hybrid ML models yet ensures 100% request satisfiability with guaranteed feasible solutions for all cases (if there exists any). Overall, ILP has been considered the primary baseline for performance comparison against our proposed approaches since it ensures optimal (best) results in retrieval.

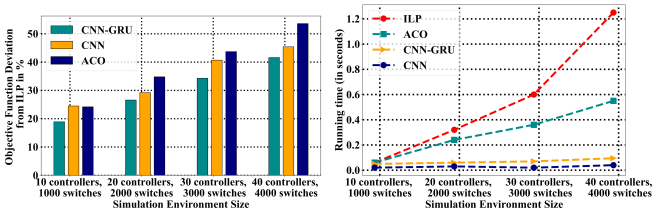
All the experiments have been carried out on DELL ALIENWARE m15 R3 machine of Intel core i7-10750H CPU @2.6 GHz equipped with 16 GB RAM and Windows 10 Home. We have used Gurobi [15] optimization solver to solve the ILP model. The CNN-GRU model has been implemented in TensorFlow [16]. Python binding has been used as the programming language for all the models, including ACO. Regardless of the implementation infrastructure used in this simulation study, the proposed models' solutions are transferable to OpenFlow [17] via a customized interface. For ACO, we use 20 ants for all experiments. The CNN-GRU model utilizes the split of 60/20/20 for training, validation, and testing purposes. The size of the simulated dataset depends on the number of controllers and switches. The number of training instances and target variables can be defined as $|S_j|$, $1 \leq j \leq m$ and $|C_i|$, $1 \leq i \leq n$, respectively. We let the model run through 20 epochs as the simulation limit, which resulted in $2-3$ hours of training time depending on the simulation environment size. The presented simulation results have been averaged over 10 different runs. To demonstrate the performances of the proposed models, we have considered two sets of experiments. In the first experiment, we solely focus on increasing the average load, and the second experiment is focused on the impacts of simulation environment size.

Since this problem has been formally proven as NP-hard, The computational complexity can increase exponentially with the growing size of problem instances. For such problems, even a marginal increase in the problem size can affect the



(a) Objective function (b) Running time (c) Objective function (d) Running time

Fig. 2: Performance of the proposed models due to varying average load in capacity tight problem scenarios



(a) Objective function (b) Running time

Fig. 3: Performance of the proposed models due to varying problem instance sizes

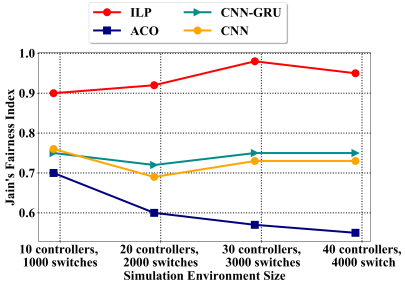


Fig. 4: Comparison based on Jain's Fairness Index

computational complexity in a significant manner. In this research study, we have considered up to 40 controllers and 4000 switches, whereas the existing research studies use considerably smaller simulation environment size (e.g., 5 controllers, 300 switches), [11]. We have considered 3 controllers with capacities randomly selected from 1000 to 2000 Mbps, and up to 100 switches for this simulation. The average traffic load of the system is varied between 10 and 60 Mbps to justify the performances of the proposed models due to processing load variance. Similar experimental settings have been adopted in the literature [11]. Under this experimental setting, we consider two types of problem scenarios: a) capacity relaxed (average load $10-30$ Mbps) and b) capacity tight ($40-60$ Mbps) instances. We adopt these two scenarios to observe the performances of our proposed models in both lightly and heavily loaded SDN controller environments. Then, we employ two preliminary performance metrics to evaluate our proposed methodologies against ILP (optimal) approach. Firstly, the objective function deviation metric gives us an idea concerning the quality of the solutions provided by the model considering ILP as the baseline. Secondly, the running time demonstrates how fast the proposed model can extend its services. However, we utilize another performance metric called Jain's fairness index to reflect the load balancing among all the controllers. Suppose the total amount of load assigned to n controllers are $\{l_1, l_2, l_3, ..., l_n\}$. Then, Jain's fairness index is calculated as follows:

$$\mathcal{J}(l_1, l_2, l_3, ..., l_n) = \frac{(\sum_{i=1}^{n} l_i)^2}{n \times \sum_{i=1}^{n} l_i^2}. \tag{10}$$

The value of this metric is 1 when all the controllers receive the same amount of load allocation. Thus, a higher Jain's fairness index metric indicates more reasonable SDN controller-switch placement solutions in terms of load balancing. It is noteworthy that all the optimal solutions do not require equal load allocation at the end in practical scenarios. Hence, the value of $\mathcal{J}$ may not be exactly 1 even for optimal solutions in most cases.

Fig. 2a illustrates that the optimality gap of the CNN-GRU model can vary from 18% to 42%, while ACO can produce an optimality gap of $30\%-65\%$. The hybrid CNN-GRU model improves resource utilization compared to the standalone CNN model by $2\% - 6\%$. In relaxed capacity instances, where the controllers have enough available capacity, we observe an increasing optimality gap with the growing amount of average load in the system. On the contrary, the feasible region reduces in tightly capacitated controller-switch mapping with a higher average load. Next, from Fig. 2b, it is noteworthy that irrespective of load size in the system, CNN-GRU and the standalone CNN model can always ensure running time approximately below 0.1 seconds, highly desirable for real-time SDN use cases. On the other hand, ACO demonstrates moderately higher running time, while the execution time of ILP continues to increase exponentially with the increased average load. Surprisingly, from Fig. 2d, we can observe that with the increasing average load, the running time of ILP and ACO starts decreasing. In the tightly capacitated system, the number of feasible solutions reduces significantly. Thus, it is easier to invade smaller feasible search space by ILP and ACO in comparatively lower running time. Nonetheless, both CNN-GRU and CNN demonstrate exceptionally low running time in tightly capacitated problems as well. For the second set of experiments, we studied the impacts of simulation environment size on the performance of the models. This study is very significant since scalability is a key factor for the SDN environment. For this purpose, we randomly assign capacity limits to the controllers between 1000 to 2000 Mbps, and the load of the switches randomly varies from

$15 - 25$ Mbps. As shown in Fig. 3a, CNN-GRU outperforms ACO by approximately at most 12%. The optimality gap of CNN-GRU varies from $20\% - 42\%$ and improves the performance of CNN by $4\% - 7\%$. Overall, the optimality deviation rate of the proposed models is not extreme with the growing size of simulation environments. However, with the progressively increasing simulation environment, unlike ILP and ACO, the running time of the hybrid CNN-GRU model remains consistently low, as shown in Fig. 3b.

Method	Satisfiability	Solution Quality	Time Complexity
ILP	100%	Optimal	Exponential
ACO	100%	Sub-optimal; Optimality gap ($22\% - 65\%$)	Quadratic
CNN	Not-guaranteed	Sub-optimal; Optimality gap ($20\% - 49\%$)	Linear (Prediction time)
CNN-GRU	Not-guaranteed	Sub-optimal; Optimality gap ($18\% - 42\%$)	Linear (Prediction time)

TABLE I: Comparative analysis on different methodologies

On the other hand, the running of ILP and ACO continue to keep extending significantly with the expansion of the simulation environment. Afterward, we evaluate a metric for the overall impression of the load balancing among all the controllers in the SDN context portrayed as Jain's fairness index. As suggested by Fig. 4, CNN-GRU is the fairest in terms of equally balancing load in every controller as much as possible right after ILP. The standalone CNN model is quite close to the hybrid model, while the performance of ACO remains notably lower. This complimentary adopted metric is an implied by-product of our considered objective function.

To summarize the absolute and potential strengths of the aforementioned methods, we outline the key performance indicators for each method in Table I. CNN-GRU outperforms both CNN and ACO by finding the closest sub-optimal solutions in relatively low running time. Moreover, ILP and ACO offer guaranteed request satisfiability by delivering solutions whenever at least one (feasible) exists, regardless of solution quality and response time. Yet, the hybrid CNN-GRU model can also reduce the chances of delivering no solution even though a solution exists to a great extent by carefully building the dataset with similar worst-case training instances.

VII. Conclusion

The traditional SDN controller-switch load balancer strategies lack the capability to support the real-time solution demanding use cases. To overcome the shortcomings and as a step closer towards futuristic SDN applications, in this paper, we proposed two AI-driven techniques: ACO and hybrid CNN-GRU model. Our proposed ACO can resolve the request satisfiability issues related to existing approaches and guarantee the deliverable of solutions wherever possible. The CNN-GRU model outperforms ACO in terms of optimality gap and demonstrates exceptionally low running time. Thus, the CNN-GRU model emerges as the most suitable and potential candidate for delay-sensitive SDN use cases. Furthermore, it is possible to train the CNN-GRU model with known possible worst-case instance patterns to increase the possibility of request satisfiability. Through an extensive simulation study, it is evident that deep learning techniques can be employed to solve various combinatorial optimization research problems aimed at emerging communication services. The future research direction is to extend the mathematical programming model and study the trade-off between load balancing and SDN migration costs incurred by load balancing in a dynamic environment.

References

[1] Y. Shi, Q. Yang, X. Huang, D. Li, and X. Huang, "An sdn-enabled framework for a load-balanced and qos-aware internet of underwater things," *IEEE Internet of Things Journal*, vol. 10, no. 9, pp. 7824–7834, 2023.

[2] E. Hajian, M. R. Khayyambashi, and N. Movahhedinia, "A mechanism for load balancing routing and virtualization based on sdwsn for iot applications," *IEEE Access*, vol. 10, pp. 37 457–37 476, 2022.

[3] B. Li, X. Deng, X. Chen, Y. Deng, and J. Yin, "Mec-based dynamic controller placement in sd-iov: A deep reinforcement learning approach," *IEEE Transactions on Vehicular Technology*, vol. 71, no. 9, pp. 10 044–10 058, 2022.

[4] K. Lu, Z. Du, J. Li, and G. Min, "Resource-efficient distributed deep neural networks empowered by intelligent software-defined networking," *IEEE Transactions on Network and Service Management*, vol. 19, no. 4, pp. 4069–4081, 2022.

[5] D. Adami, S. Giordano, and M. Pagano, "Elastic load balancing in software defined satellite networks," in *2022 IEEE 27th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*, 2022, pp. 207–212.

[6] J. L. Herrera, J. Galán-Jiménez, L. Foschini, P. Bellavista, J. Berrocal, and J. M. Murillo, "Qos-aware fog node placement for intensive iot applications in sdn-fog scenarios," *IEEE Internet of Things Journal*, vol. 9, no. 15, pp. 13 725–13 739, 2022.

[7] N. Ahmed and S. Misra, "Collaborative flow-identification mechanism for software-defined internet of things," *IEEE Internet of Things Journal*, vol. 9, no. 5, pp. 3457–3464, 2022.

[8] M. Emu and S. Choudhury, "Iot ecosystem on exploiting dynamic vnf orchestration and service chaining: Ai to the rescue?" *IEEE Internet of Things Magazine*, vol. 3, no. 4, pp. 30–35, 2020.

[9] P. P. Shahrbabaki, R. W. L. Coutinho, and Y. R. Shayan, "A novel sdn-enabled edge computing load balancing scheme for iot video analytics," in *2022 IEEE Global Communications Conference*, Rio de Janeiro, Brazil, 2022, pp. 5025–5030.

[10] I. Maity, R. Dhiman, and S. Misra, "Enplace: Energy-aware network partitioning for controller placement in sdn," *IEEE Transactions on Green Communications and Networking*, vol. 7, no. 1, pp. 183–193, 2023.

[11] A. Filali, A. Kobbane, M. Elmachkour, and S. Cherkaoui, "Sdn controller assignment and load balancing with minimum quota of processing capacity," in *IEEE International Conference on Communications (ICC)*, MO, USA, Oct. 2018, pp. 1–6.

[12] G. Aggarwal, R. Motwani, and A. Zhu, "The load rebalancing problem," in *Proceedings of the fifteenth annual ACM symposium on Parallel algorithms and architectures*, Jun. 2003, pp. 258–265.

[13] E. man Jr, M. Garey, and D. Johnson, "Approximation algorithms for bin packing: A survey," in *Approximation algorithms for NP-hard problems*, 1996, pp. 46–93.

[14] Y. Zhao, Y. Li, X. Zhang, G. Geng, W. Zhang, and Y. Sun, "A survey of networking applications applying the software defined networking concept based on machine learning," *IEEE Access*, vol. 7, pp. 95 397–95 417, 2019.

[15] G. optimizer, "Gurobi optimizer reference manual, 2020," 2014.

[16] N. Ketkar, "Introduction to tensorflow," in *Deep Learning with Python*. Springer, 2017, pp. 159–194.

[17] 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, Apr. 2008.