ISSN: 2093-5374 / E-ISSN: 2093-5382

Energy-Aware Controller Load Distribution in Software-Defined Networking using Unsupervised Artificial Neural Networks

Poom Somwong[1], Karn Patanukhom[2], and Yuthapong Somchit[3*]

[1]OASYS Research Group, Department of Computer Engineering, Faculty of Engineering, Chiang Mai University, Chiang Mai 50200, Thailand. poom_somwong@cmu.ac.th, https://orcid.org/0009-0006-9384-4326

[2]Department of Computer Engineering, Faculty of Engineering, Chiang Mai University, Chiang Mai 50200, Thailand. karn@eng.cmu.ac.th, https://orcid.org/0000-0002-9292-7625

[3*]OASYS Research Group, Department of Computer Engineering, Faculty of Engineering, Chiang Mai University, Chiang Mai 50200, Thailand. yuthapong@eng.cmu.ac.th, https://orcid.org/0009-0002-9112-5134

Received: October 31, 2024; Revised: December 16, 2024; Accepted: January 27, 2025; Published: March 31, 2025

Abstract

Software-Defined Networking (SDN) enhances network management by separating the control and data planes into controllers and switches, allowing for centralized, programmable networks with multiple controllers. Switches are mapped to controllers and exchange control messages to manage the network, which leads to significant energy consumption. Managing energy in networks has become a critical issue, as dynamic changes in switch loads can cause controller overloads, necessitating the migration of switches to other controllers. As networks grow, energy consumed in control communications becomes a major concern. This paper proposes an unsupervised learning Artificial Neural Network (ANN) model to address controller overloads and optimize energy consumption, achieving faster execution times compared to conventional methods while maintaining manageable energy efficiency. The model considers dynamic switch loads and the hop distance between switches and controllers when remapping switches to optimize energy use. Experimental results demonstrate that the proposed unsupervised ANN model performs effectively in large networks, enabling efficient handling of controller overloads during variations in switch loads. The adaptability of the ANN model provides a robust strategy for energy-efficient load distribution, enhancing the scalability and efficiency of SDN environments.

Keywords: SDN, Artificial Neural Networks, Unsupervised Learning, Energy Consumption, Controller Overload.

1 Introduction

The traditional networking model integrates control logic and data forwarding within individual network devices (Sufiev & Haddad, 2016). In contrast, Software-Defined Networking (SDN) architecture separates these functions into control and data planes, offering centralized, programmable network control, enabling easier policy implementation, configuration updates, and rapid deployment of new

Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications (JoWUA), volume: 16, number: 1 (March), pp. 289-314. DOI: *10.58346/JOWUA.2025.I1.018*

*Corresponding author: OASYS Research Group, Department of Computer Engineering, Faculty of Engineering, Chiang Mai University, Chiang Mai 50200, Thailand.

services (Ahmad & Mir, 2020). The control plane includes a centralized controller that maintains a comprehensive network view and manages data packets, such as routing decisions (Isyaku et al., 2020). The data plane includes data packet forwarding devices called SDN switches, which use a table called a flow table to manage data packets, such as forwarding data packets according to the routes in the flow table (Abuarqoub, 2020). This flow table receives rules or instructions from the controller.

Communication between controllers and switches is called southbound communication using the OpenFlow protocol, which involves several types of OpenFlow messages, e.g., *packet_in*, *packet_out*, and *flow_mod* messages, representing the control traffic load on the network (Kim et al., 2013). These messages are exchanged between the controller and switches to manage data traffic, i.e., *packet_in* messages are sent from the switch to the controller when the packet does not match any rule in its flow table, while *packet_out* and *flow_mod* messages are sent from the controller to the switches to instruct how to handle packets and modify the flow table, respectively. The controller load is measured in packets per second (pps). In large-scale SDN deployments, it is common to employ multiple controllers to enhance performance, reliability, and fault tolerance (Blial et al., 2016). Typically, one switch maps to only one controller, while one controller can map to multiple switches depending on its capacity to handle the amount of control load.

In SDN, communication between controllers and switches can be achieved through out-of-band or in-band communication methods. Out-of-band communication uses a separate exclusive network for control communication, which allows administrators to have more control over network operations but acquires higher implementation costs due to the need for a separate network (Neves et al., 2022). Conversely, in-band communication requires sharing the same network infrastructure for the control and data traffic, allowing for greater flexibility, programmability, and simplified infrastructure (Carrascal et al., 2023). This research focuses on in-band communication within the control communication.

In in-band communication, not all switches are directly connected to their mapped controllers. To transmit control packets between switches and mapped controllers, switches that are not directly connected to their mapped controllers must forward packets through other switches to reach the mapped controller (Hussain et al., 2021). This requires multiple switches to forward packets from one switch to its mapped controller, and the number of these switches is counted as hops. The study in (Fröhlich et al., 2021) shows that when switches forward more packets, they consume more energy for data transmission. This means that when switches have to forward control messages for other switches, they consume more energy. Consequently, when the number of hops between the switch and controller increases, the energy consumption also increases. With the same load, if a switch is mapped to a closer controller, the network will consume less energy than when it is mapped to a further controller. Thus, the load on switches and the number of hops required for communication between controllers and switches notable impact the overall energy consumption of the network (Somwong & Somchit, 2024).

Energy consumption in data transmission networks is increasingly important due to the rising volume of traffic and the subsequent increase in operational costs and environmental impact (Rozite, Bertoli, & Reidenbach). This issue is related to the Sustainable Development Goal (SDG) that focuses on energy, specifically SDG 7: Affordable and Clean Energy. In SDN, where communication between controllers and switches involves multiple hops and extensive control packet exchanges, the energy consumed due to control packet transmission cannot be neglected, as it has a significant impact. Reducing this energy consumption is essential for improving network efficiency and sustainability (Shuja et al., 2017). By managing the mapping of switches to controllers while considering the loads on switches and the number of hops between switches and controllers, it is possible to minimize the energy required for packet transmission, leading to more energy-efficient network operations.

To minimize energy consumption, switches should initially be mapped to the nearest controller, considering the hop counts between switches and controllers (Choumas et al., 2020). However, as switches begin communicating with controllers, causing load on the controllers, and since controllers have limited capacity to handle loads from switches, excessive load can lead to controller overload (Jalili et al., 2020) . In such cases, switch migration, which involves remapping switches to other controllers, is required. Additionally, network load changes dynamically over time due to user activity (Zhao et al., 2021). Consequently, switch migration becomes necessary and must adapt to controller overload conditions. This process solves the controller overload problem but results in switches being remapped to potentially more distant controllers. This increased hop count increases energy consumption after migration. However, energy consumption must remain minimized after the migration. The challenge of switch migration for load balancing, while considering the energy consumption problem, is recognized as an NP-hard problem (Kumari & Sairam, 2024). This complexity arises from the need to assign multiple switches to potentially multiple controllers to optimize the mapping (Zhou et al., 2018)

Several researchers have proposed methods for addressing the controller overload problem, although these methods do not yet account for the energy consumption from control traffic (Alhilali & Montazerolghaem, 2023; Semong et al., 2020). Many of these methods focus on load balancing across multiple controllers using various algorithms, including nature-inspired approaches such as Particle Swarm Optimization (PSO) (Zhu et al., 2015) and Ant Colony Optimization (ACO) (Torkzadeh et al., 2021). The work in reference (Pathan et al., 2024) proposed using Mixed Integer Linear Programming (MILP) to balance data traffic loads while considering energy consumption. Nevertheless, this work does not focus on load balancing of controllers considering the energy consumption of control traffic.

There are works such as Energy Controller Load Distribution (ECLD) (Somwong & Somchit, 2024) and Self-Adaptive Load Balancing Scheme (SALB) (Priyadarsini et al., 2020) proposed methods to solve this load balancing with energy aware problem. However, they are still time intensive and not proper for dynamic load change network.

Meanwhile, machine learning approaches have shown potential in addressing NP-hard problems due to their ability to learn complex patterns and make predictions based on data (Martín et al., 2019). Among these, Artificial Neural Networks (ANNs) stand out for their flexibility and powerful modeling capabilities, inspired by the human brain's neural networks (Kouroshnia & Farokhi, 2018), which consist of interconnected neurons processing inputs and generating outputs through learned weights and biases (Zou et al., 2009). They can approximate complex functions and are useful for solving optimization problems (Villarrubia et al., 2018).

Previous research (Ruelas & Rothenberg, 2018; WilsonPrakash & Deepalakshmi, 2019) has explored the use of ANNs in supervised learning to achieve controller load balancing. However, these studies do not consider energy consumption optimization. Furthermore, collecting labeled data for supervised learning in large networks is challenging due to the extensive data requirements and the dynamic nature of network traffic. Thus, supervised learning might not be appropriate for addressing energy consumption (Giliberto et al., 2019), especially as network size increases.

These limitations emphasize significant challenges in current research, including the computational inefficiencies of existing methods and the reliance on labeled data in supervised learning-based approaches. Addressing these challenges requires a scalable and adaptable solution that minimizes energy consumption, dynamically adjusts to changing network conditions, and operates without the need for labeled data.

The proposed approach leverages the scalability of ANNs and the adaptability of unsupervised learning to optimize switch-to-controller mapping in SDN, addressing the computational inefficiencies of existing methods and eliminating the need for labeled data. By considering energy consumption during load distribution, this approach provides a practical and energy-efficient solution suitable for large and dynamic networks, where obtaining labeled data is often challenging due to the dynamic nature of network traffic and diverse mapping combinations. Our main contributions are:

- The proposed unsupervised ANN model achieves faster switch-to-controller mappings compared to existing algorithms, with significant efficiency gains in larger network topologies.
- The ANN model inputs include switch load, hop count to controllers, and controller capacity to determine switch-to-controller mappings that minimize energy consumption and perform load distribution to prevent controller overload.
- An objective function was developed to optimize energy consumption and prevent controller overload, and it was incorporated into the ANN model as a loss function for evaluation during training, which employs unsupervised learning techniques.

The remainder of this research is organized as follows: Section 2, Related Works, reviews the literature on methodologies and algorithms used for load balancing in SDN, including considerations of energy consumption. Section 3, Proposed Work, describes the problem and outlines the architecture of the proposed model. Section 4, Experimental Setup, details the environment used to evaluate the performance of the proposed model. Section 5, Experimental Results, discusses the evaluation results. Finally, Section 6, Conclusions, concludes this research work.

2 Related Works

Over the past several years, numerous research efforts have been conducted to solve problems related to switch migration, also known as switch-to-controller mapping, by performing load balancing in networks utilizing SDN capabilities. Different research works employ diverse criteria to evaluate and demonstrate their performance. In this section, the relevant literature is reviewed, categorizing the research into methods for load balancing and energy consumption management.

For research employing dynamic programming to solve the problem of switch migration and load balancing, the research by Somwong and Somchit (Somwong & Somchit, 2024) proposed an algorithm called Energy-aware Controller Load Distribution (ECLD). This algorithm is designed to solve controller overload by efficiently distributing the load among controllers while minimizing overall energy consumption. The research focuses on factors such as switch loads and the number of hops between switches and controllers to calculate energy costs, which are then used to select the mapping of switches and controllers. This research adapts dynamic programming techniques from the knapsack problem to solve the switch migration problem while considering energy consumption. However, it has been observed that using dynamic programming methods introduces issues with time complexity or computational overhead for the proposed algorithm. This is because solving the knapsack problem using dynamic programming can be computationally expensive, especially for large network topologies with numerous switches and controllers.

For load balancing using a greedy algorithm for switch migration that allows for rapid problem-solving, the work of (Priyadarsini et al., 2020) presents a framework called the Self-Adaptive Load Balancing scheme (SALB) for managing controller load balancing. This framework consists of various mechanisms that work together to determine the mapping of switches and controllers for migration, to optimize load distribution across controllers while minimizing energy usage. Specifically, this research

employs a heuristic greedy method to prioritize which switches should be mapped with new controllers by considering the switch load amounts and the hop count to the controllers. The approach prioritizes migrating the most heavily loaded switches to the nearest new controllers to reduce energy consumption from load balancing. However, the proposed approach may face issues related to computational overhead when the network size increases due to a higher number of switches and controllers. Additionally, the complexity of coordinating the involved mechanisms can pose practical challenges when implementing this framework in real-world settings.

Kurroliya et al., (2020) have presented a genetic algorithm designed to optimize the allocation of switches to controllers to specify active links and maximize inactive links between switches and controllers while satisfying latency requirements. This algorithm is called the Link-Based Genetic Algorithm (LBGA) and is used for load balancing. The operation of this algorithm involves randomly selecting link paths from switches to controllers using genetic processes like crossover, which combines elements of two good link paths, and mutation, which introduces random changes to explore new link paths. While the genetic algorithm is effective in exploring a wide range of solutions, it can be computationally intensive and complex to implement. Moreover, inactivating links or nodes can lead to longer paths for data transmission, thereby increasing latency. Additionally, having fewer active paths reduces redundancy, impacting the network's ability to efficiently reroute traffic in case of failures or congestion.

Due to the capabilities of using SDN in networks, recent research has integrated machine learning (ML) for load balancing. In (Ruelas & Rothenberg, 2018), Ruelas and Rothenberg introduced a method of load balancing in data center networks using ANNs. The model takes network traffic metrics as input, such as bandwidth and latency data for each network path. The output prediction of the model aims to predict network traffic demand and choose the least loaded paths between the source and destination to increase bandwidth utilization and reduce path latency in the network.

(WilsonPrakash & Deepalakshmi, 2019) have proposed a load balancing algorithm called Dynamic Agent-Based Load Balancing (DA-LB) that leverages a Back Propagation Artificial Neural Networks (BPANNs). This algorithm monitors the loads of virtual machines (VMs) in the network. When an overload in VMs is detected, the algorithm triggers a migration process. The BPANNs model is utilized to analyze current load metrics, such as CPU utilization, memory usage, and response times. The model's output prediction determines the optimal target VM for load migration. The proposed algorithm effectively reduces the time required for VM migrations.

Kumar et al., (2022) introduced the use of a Convolutional Neural Network (CNN) to manage and distribute network loads. The CNN model works by analyzing multiple traffic parameters, such as packet size, packet rate, and flow direction. The model employs convolutional layers to extract features, pooling layers to reduce data dimensions, and fully connected layers to make load distribution decisions. The output of the CNN model provides optimal paths for balancing network traffic loads among available pathways. The proposed CNN model achieves high accuracy and requires fewer resources compared to existing machine learning-based load balancing techniques.

In addition to using ANNs for predicting load balancing in networks, another technique that has been adopted is Reinforcement Learning (RL). This involves training an agent to manage network traffic loads. Tosounidis et al., (2020) propose using deep reinforcement learning, specifically Deep Q-Learning with a convolutional neural network (Al Sallami, & Al Alousi, 2013), for the task of load balancing traffic across servers in an SDN environment. This approach can dynamically adapt to changing network conditions and server loads by learning from the centralized visibility provided by the

SDN control plane. However, a potential limitation is that their evaluation is limited to a simulated and relatively small-scale environment, thus the scalability of their approach for very large network topologies remains unclear.

Xiang et al., (2022) present a Deep Reinforcement Learning-based Switch Migration Strategy (DRL-SMS) for addressing the load balancing problem in software-defined networking (SDN). The model is trained using a Double Deep Q-Network (DDQN) architecture that combines deep neural networks (Kooshki et al., 2016). with Q-learning to map network states to switch migration actions. The output of the model is a set of optimal switch migration actions. However, a limitation of this work is the high computational complexity of the training phase, which can become prohibitively expensive, especially for very large SDN topology deployments with many controllers and switches.

Nevertheless, none of the works (Ruelas & Rothenberg, 2018; WilsonPrakash & Deepalakshmi, 2019; Kumar et al., 2022; Tosounidis et al., 2020; Xiang et al., 2022) specifically mention energy consumption as a focus of the proposed algorithms. In contrast, research by (Kandil et al., 2022) addresses this by presenting an approach to balance energy efficiency and Quality of Service (QoS) in SDN routing using reinforcement learning techniques. They employ a simulated annealing exploration strategy, which provides better convergence and performance, named the Simulated annealing Q-learning (SAQL) routing algorithm. The SAQL algorithm effectively solves the routing problem by selecting paths that optimize both energy consumption and QoS parameters (latency and packet loss ratio). However, evaluating the algorithm primarily using small network topologies may limit the understanding of its performance in more complex or larger-scale environments. Moreover, the algorithm's performance appears to degrade under very high traffic loads, as the agent prioritizes QoS optimization over energy efficiency.

Sallami et al., (2013) proposed a load balancing technique based on ANNs. The main goal of this technique is to minimize energy consumption. The output of the model predicts how the current workload should be optimally balanced across servers, including turning off unutilized servers to save energy. While the general concept of using ANNs to balance workloads across servers and turn off unutilized servers is sensible for saving energy, the paper lacks energy modeling details, and comparisons to other energy-aware load balancing techniques.

In this research, we present the use of unsupervised learning ANNs for network load balancing with an awareness of energy consumption, highlighting the advantages of neural network integration. The integration of ANNs with SDN supports scalability in managing large-scale network topologies by adapting to changes in network load. While previous research using ANNs by (Ruelas & Rothenberg, 2018; WilsonPrakash & Deepalakshmi, 2019; Kumar et al., 2022; Sallami et al., 2013) employs supervised learning models that rely on pre-labeled datasets, which can be time-consuming and resource-intensive to obtain, especially in complex and dynamic SDN environments, the unsupervised learning techniques used in this research address the challenge of unlabeled datasets. These techniques allow for the use of extensive datasets, including generated data, without the need for labeling. This expands the possibility of having a larger dataset, enabling the model to be trained more efficiently and making it adaptable to various network environments.

Moreover, a significant aspect of this research is the consideration of energy consumption from the mapping between switches and controllers. This research does not use inactive paths or devices to save energy because disabling interfaces would disrupt data packet transmission. Additionally, controllers must remain active; reactivating them involves reconnecting and sharing information with switches and other controllers, a time-consuming process that may affect network QoS (Somwong & Somchit,

2024).This research focuses on load balancing in the control plane, the connection between switches and controllers as an in-band communication, a form of communication commonly used in real-world communication.

3 Proposed Work

This section explains the energy consumption problem within a network model, detailing the formulation of equations to optimize energy usage. Additionally, we introduce the architecture of ANN model designed to solve this problem. The performance of the model is evaluated using a specific loss function equation.

3.1 Network Model

The network model is described by the set of connected graphs $G = \{N, L\}$, where N represents the set of nodes and L represents the set of edges. The set of nodes is a union of a set of switches $S = \{s_1, s_2, s_3, \dots, s_p\}$ and a set of controllers $C = c_1, c_2, c_3, \dots, c_k$, with $|S| = p$ and $|C| = k$. As illustrated in Figure 1, the network topology is a connected graph where each switch and controller can find paths to reach each other. Each controller has a domain of controlled switches, representing the switches mapped to that controller. Each controller j is defined by a capacity threshold b_j and current loads u_j. The variable $y_{(i,j)} \in \{0,1\}$ indicates that switch i is mapped to controller j when the value is 1.

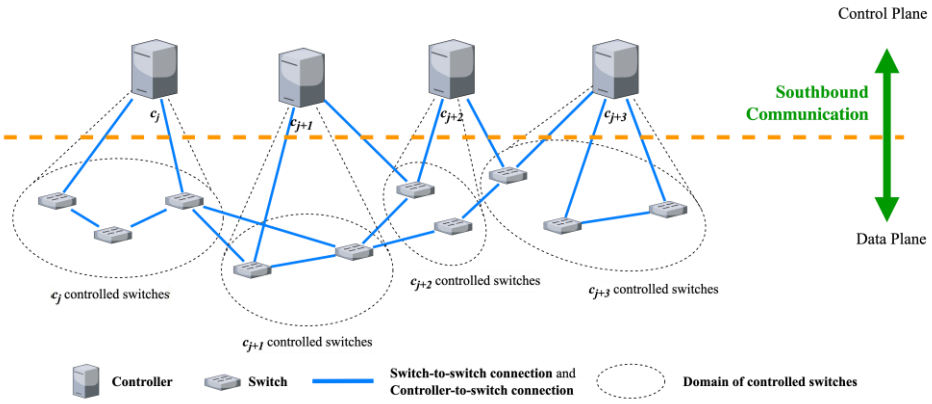

Figure 1: SDN Network Topology

3.2 Energy Consumption

The energy consumption is represented by the energy cost. The energy cost from switch i when mapped to controller j is denoted as $e_{(i,j)}$, and the total energy cost e from all switches and controllers in the network can be determined using the following equation:

$$e_{(i,j)} = l_i h_{(i,j)} y_{(i,j)} \quad (1)$$

$$e = \sum_{j=1}^{k} \sum_{i=1}^{p} e_{(i,j)} \quad (2)$$

where l_i is the load of switch i and $h_{(i,j)}$ are the number of hops from switch i to controller j.

In network operations, switch load varies depending on the traffic during different periods of the day. When a controller, which is mapped to switches generating a large load, exceeds its threshold capacity, switch migration becomes necessary. Initially, switches are mapped to the nearest controller to minimize energy consumption. However, after migration, switches may be remapped to more distant controllers, potentially increasing total energy consumption.

This problem of mapping switches with controllers for load balancing can be translated into an optimization problem. The objective is to ensure that switch migration results in minimized energy consumption and that no controller becomes overloaded. This can be formulated with an objective function and constraints as follows:

$$Minimize\ e = \sum_{j=1}^{k}\sum_{i=1}^{p} e_{(i,j)} \tag{3}$$

Subject to,

$$\sum_{j=1}^{k} y_{(i,j)} = 1, \quad \forall_i \in \{1,2,3,\dots,p\} \tag{4}$$

$$\sum_{i=1}^{p} l_i y_{(i,j)} \leq b_j, \quad \forall_j \in \{1,2,3,\dots,k\} \tag{5}$$

The aim is to minimize the total energy cost, as defined in Equation (3). This objective is subject to constraints that each switch must be mapped to exactly one controller as specified in Equation (4). Equation (5) includes a capacity constraint to ensure that no controller is overloaded beyond its capacity by the switches mapped to. Here, all variables are integers. This kind of problem can be solved using Integer Linear Programming (ILP). However, ILP is very time-consuming. When the switch loads change, the calculations must be recomputed, which is not suitable for networks with dynamic load changes. Many methods have been proposed to reduce the time required to solve the problem, trading off with near-optimal solutions, such as Constraint Programming Satisfiability Solver (CP-SAT) (Perron et al., 2023), ECLD, and SALB. However, these methods are still time-intensive and therefore not suitable for networks with dynamic load changes.

Artificial Neural Networks (ANNs) are a type of machine learning that creates models to solve problems. Various datasets with diverse input values are used to train these models. Once trained, the model can solve problems even with previously unseen data. ANNs offer the significant advantage of solving problems much faster than conventional methods. Therefore, in this work, we propose using ANNs to solve this optimization problem instead of traditional approaches. In addition, when variables such as switch loads are not integers, the problem must be solved by Mixed Integer Linear Programming (MILP), which is more complex than ILP. However, ANNs can still solve this problem effectively, regardless of whether the variables are integers or not.

3.3 Artificial Neural Networks

ANNs are widely used in SDN for tasks such as load balancing within networks. However, previous research has not focused on load balancing that considers the energy consumption resulting from switch migration, as discussed in the related works section. Therefore, this research employs ANNs using unsupervised learning to solve the problem of switches-to-controller mapping, achieving load balancing

without overloading controllers while also considering energy consumption. The architecture of the model used in this research is shown in Figure 2. The model includes three inputs:

- **Energy Matrix:** A matrix representing the energy cost of mapping each switch to each controller, with a dimension of $k \times p$.
- **Switch Load Vector:** A vector representing the loads of switches, with a length of p.
- **Controller Capacity Vector:** A vector representing the capacities of controllers, with a length of k.

These model inputs will be explained in subsection 3.4.

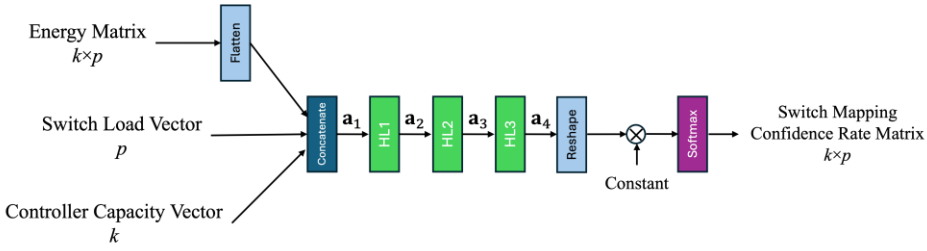

Figure 2: Architecture of ANN Model

The model comprises three hidden layers as Dense layers, denoted as HL1, HL2, and HL3 in Figure 2, each with a different number of nodes. Each hidden layer employs the Scaled Exponential Linear Unit (SELU) as its activation function. The equation of SELU is shown in Equation (6).

$$\mathrm{SELU}(x) = \begin{cases} \lambda x & \text{if } x > 0 \\ \lambda(\alpha \exp(x) - \alpha) & \text{if } x \leq 0 \end{cases} \quad (6)$$

Here, λ and α are predefined constants. Specifically, λ is approximately 1.0507 and α is approximately 1.67326.

Before being fed into the model, the Energy Matrix is flattened into a vector of length $k \times p$. Then, all three inputs are concatenated to form the initial input vector, denoted as $\mathbf{a_1}$, which has a length of $k \times p + p + k$. The input passes sequentially through these layers, from the first to the third, resulting in an output denoted as $\mathbf{a_4}$. This output is then reshaped back a matrix with the dimension of $k \times p$ and scaled by a constant to adjust the input values appropriately for the SoftMax activation function. This constant enhances the clarity of the probabilities and assists the model in more accurately differentiating between predicted classes. The final output of the model is a confidence rate that maps each switch to every controller in the network, with values ranging from 0 to 1, where 1 represents the highest likelihood that the switch should be mapped to a particular controller.

The mathematical equation describing the operation of the hidden layers in Figure 2 is given in Equation (7)

$$\mathbf{a}_{i+1} = SELU(\mathbf{a}_i \mathbf{w}_i + \mathbf{z}_i) \quad (7)$$

Denotes $\mathbf{a}_i$ represents the activations or outputs from the previous layer i. These inputs are transformed through a linear operation, specifically a weighted sum, where $\mathbf{w}_i$ denotes the weight matrix. These weights adjust the importance or influence of the input values on the resultant output. The bias vector $\mathbf{z}_i$ is added to this weighted sum, which helps the model make better decisions by allowing it to fit the data more flexibly.

3.4 Model Input

In this research, the model employs unsupervised learning techniques to analyze network states, which represent the load of each switch across different periods for each network topology. Additionally, the model considers the load of each switch and the capacity of each controller as key parameters. These parameters serve as inputs to the model. To ensure consistency, all input values are normalized by dividing them by the maximum switch load observed during the respective period. The model inputs are as follows:

- **Energy Consumption Cost**

$$\text{Energy Matrix} = \begin{bmatrix} l'_1 h_{(1,1)} & l'_2 h_{(2,1)} & \cdots & l'_p h_{(p,1)} \\ l'_1 h_{(1,2)} & l'_2 h_{(2,2)} & \cdots & l'_p h_{(p,2)} \\ \vdots & \vdots & \ddots & \vdots \\ l'_1 h_{(1,k)} & l'_2 h_{(2,k)} & \cdots & l'_p h_{(p,k)} \end{bmatrix} \quad (8)$$

This matrix represents the energy cost of mapping each switch to each controller, with dimensions $k \times p$.

- **Load of Each Switch**

$$\text{Switch Load Vector} = [l'_1 \quad l'_2 \quad \cdots \quad l'_p] \quad (9)$$

This vector represents the normalized loads of the switches, with a length of p .

- **Capacity Threshold of Each Controller**

$$\text{Controller Capacity Vector} = [b'_1 \quad b'_2 \quad \cdots \quad b'_k] \quad (10)$$

This vector represents the capacities of the controllers, with a length of k.

Here, l'_i represents the normalized load of switch i, and b'_j represents the normalized load of controller j . They can be calculated from Equations (11) and (12).

$$l'_i = \frac{l_i}{\max_i l_i} \quad (11)$$

$$b'_j = \frac{b_j}{\max_i l_i} \quad (12)$$

3.5 Loss Function

We evaluate the performance of the model during the training process using a loss function. This loss function is derived from the objective function described in Equation (3) and the constraints in Equations (4)-(5). We define a dataset as a network state used as the model input, which includes the Energy Matrix, Switch Load Vector, and Controller Capacity Vector. To train the model for a specific network topology, multiple network states are used as datasets. The formulation of the loss function for one network state combines each variable as shown in Equations (13)-(17) as follows:

- **Confidence Mapping**

$$\mathrm{Z} = \sum_{i=1}^{p} \left(1 - \max_{j} \hat{y}_{(i,j)}\right)^2 \tag{13}$$

As explained in subsection 3.1, $y_{(i,j)}$ can only be 0 or 1, indicating that switch i is mapped with controller j if the value is 1. However, the value obtained from $\hat{y}_{(i,j)}$ is a continuous value represents the confidence interval for mapping switch i to controller j. The value ranges from 0 to 1. The aim of this term is to make the characteristics of $\hat{y}_{(i,j)}$ similar to those of $y_{(i,j)}$.

- **Switch and Controller Mapping Energy Consumption**

$$E = \left(\frac{1}{pk} \sum_{j=1}^{k} \sum_{i=1}^{p} l_i' h_{(i,j)} \hat{y}_{(i,j)}\right)^2 \tag{14}$$

This term calculates the total energy consumption across the network, considering the load of each switch and the hop count to the mapped controller.

- **Overload Capacity**

$$\Gamma = \sum_{j=1}^{k} \left(\max\left(0, u_j' - b_j'\right)\right)^2 \tag{15}$$

This term adjusts the importance of ensuring that the load u_j' of any controller j does not exceed its capacity b_j'. Here, u_j' is defined as the normalized total load on controller j, calculated by Equation (16).

$$u_j' = \sum_{i=1}^{p} l_i' \hat{y}_{(i,j)} \tag{16}$$

- **Load Balancing**

$$B = \left(\max_{j} u_j' - \min_{j} u_j'\right)^2 \tag{17}$$

This term is the square of the difference between the highest and lowest loads among the controllers.

By integrating the terms in Equations(14)-(17), the loss function used for this model from all network states in the training dataset is the Mean Square Error (MSE), formulated as Equation (18).

$$Loss = \frac{1}{N} \sum_{s=1}^{N} (Z_s + \varepsilon E_s + \gamma \Gamma_s + \beta B_s) \tag{18}$$

where N is the number of trained datasets, and ε, γ, and β are weighting factors that adjust the importance of each term. This loss function aims to achieve an optimal balance by minimizing total

energy consumption, preventing any controller from becoming overloaded, and ensuring even load distribution across the network.

3.6 Model Output

The switch mapping confidence rate matrix from the model prediction, denoted as $\hat{\mathbf{y}}$, consists of elements $\hat{y}_{(i,j)}$ that range from 0 to 1. A value of 1 indicates a high likelihood that switch i is appropriately mapped to controller j. To determine the mapping, the prediction output for each switch-controller pair is rounded to identify the argument maximum of the confidence rates. The controller corresponding to the argument maximum is assigned a value of 1, indicating selection, while other controllers for that switch are assigned a value to 0. The matrix for the output prediction $\hat{\boldsymbol{y}}$ before rounding is provided in Equation (19).

$$\hat{\mathbf{y}} = \begin{bmatrix} \hat{y}_{(1,1)} & \hat{y}_{(2,1)} & \cdots & \hat{y}_{(p,1)} \\ \hat{y}_{(1,2)} & \hat{y}_{(2,2)} & \cdots & \hat{y}_{(p,2)} \\ \vdots & \vdots & \ddots & \vdots \\ \hat{y}_{(1,k)} & \hat{y}_{(2,k)} & \cdots & \hat{y}_{(p,k)} \end{bmatrix} \tag{19}$$

3.7 Model Training

The models are created to fit a specific topology. For each topology, training requires a large dataset with varying network states, i.e., the load of each switch varies in each network state. Additionally, we leverage the benefit of unsupervised learning, where datasets do not need to be labeled in advance, allowing for the creation of a very large dataset for training purposes. This approach ensures that the models are finely tuned to accommodate the specific network topology and distribution of loads, enabling them to predict the mapping of unseen network states.

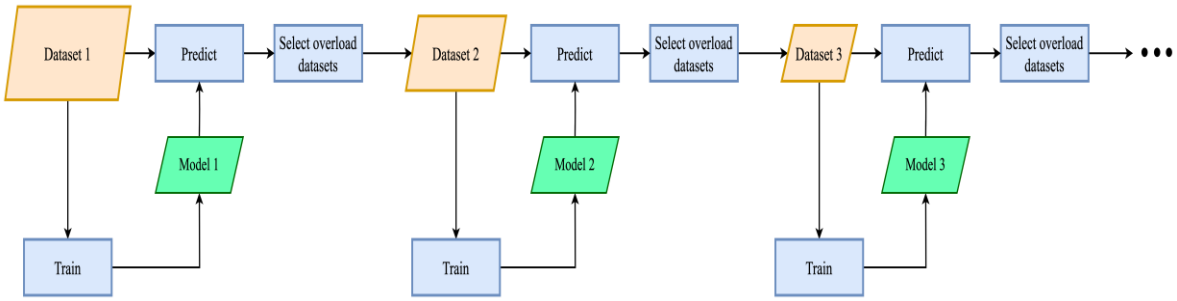

Figure 3: Model Training Process

In our proposed method, for one network topology, there might be multiple models used during the inference phase, as explained in subsection 3.8. The procedure to create the models is as follows: First, all datasets from all network states are used to train a model. Then, this model is used to predict the mapping for the same network states. Based on the mapping prediction output of each network state, we map the switches and controllers according to this output and check if there are any network state datasets where controllers are overloaded. These controller overload network states are then used to create and train the next model. This process is repeated until the number of controller overloads is less than the threshold of acceptable network states with controller overloads. This results in r models for each specific topology, where the value of r varies. The process of training multiple models is shown in Figure 3.

3.8 Inference Phase

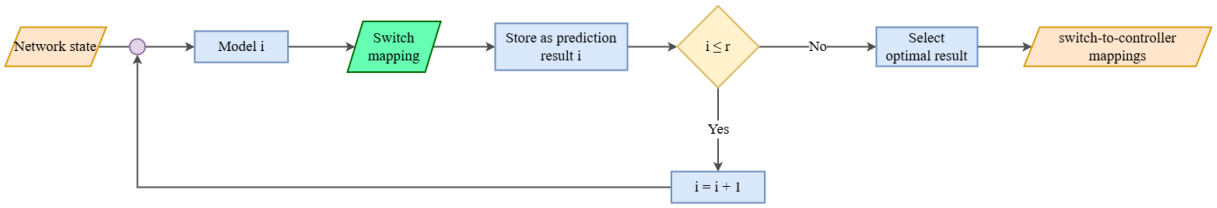

Figure 4: Proposed ANN-Full

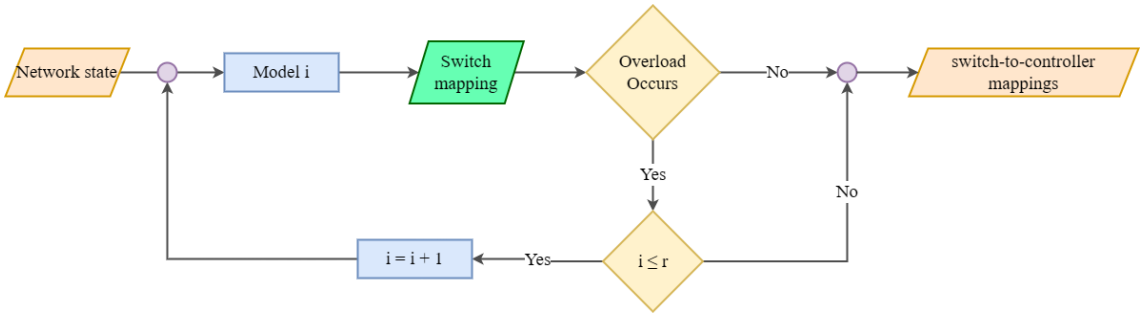

Figure 5: Proposed ANN-Fast, in which the Models are Sorted

This research proposes methods for the inference of models to predict the mapping solution between switches and controllers. To perform inference on the model, a network state serves as the model input, and the mapping matrix prediction is the output. Additionally, we define the result from the mapping as the total energy cost and controller overload status when switches are mapped to controllers according to the model's output prediction.

As previously mentioned, there are multiple models for one specific topology. To perform inference on these multiple models, we propose two methods as follows.

- **Proposed ANN-Full:** This method involves predicting with all models and selecting the optimal result, which is the result with the least total energy cost without controller overloads, from one model among all r models for a given dataset. Although this approach ensures the best possible outcome, it requires some time because predictions from all models are obtained sequentially, as illustrated in Figure 4. Another way to use this method is to predict with all models in parallel to reduce time. This would reduce the total time to the time taken to predict with all models sequentially divided by the number of models. However, predicting with all models in parallel requires more computing resources.
- **Proposed ANN – Fast:** This method predicts the optimal output using the predictions from one or multiple models through a statistical process. Several new network state sets are generated to make predictions using the full method. From the results of each dataset, we keep statistics on which model is selected for that dataset, as it provides the most optimal result. Based on these, we rank the models from the most frequently selected to the least. When implementing this method, we first input the network state dataset into the top-ranked model and check whether the result still indicates controller overload. If overloads still occur, we use the next-ranked model for prediction. This process is repeated until no overloads occur, as shown in Figure 5.

3.9 Time Complexity Analysis

The training phase of an ANN model has a time complexity of $O(r \times N \times M \times K)$, where r is the number of trained models, N is the number of trained datasets, M is the number of model parameters, and K is the number of epochs. This complexity grows linearly with the increasing number of training examples, model parameters, and epochs, highlighting the computational demands of the training process. In contrast, the prediction phase, which involves making inferences using the trained model, has a time complexity of $O(r \times M)$. This lower complexity during the prediction phase allows for efficient real-time inference, making ANNs particularly suitable for load balancing in SDN environments, were network loads change dynamically.

4 Experimental Setup

The network environment used for simulating and evaluating this research utilizes Python version 3.10.12 with the NetworkX library (Hagberg et al., 2008). The topology for the network is sourced from the Internet Topology Zoo. (Knight et al., 2011) which is widely used in network research as it reflects real-world scenarios by combining various network topologies from around the world. In this research, the Internet Topology Zoo is utilized for small to medium-sized networks. For larger networks, the Barabási–Albert model is employed to effectively model systems such as the Internet and the World Wide Web (Barabási & Albert, 1999), with a configurable parameter for the number of connections added during node creation, offering flexibility in generating realistic network structures. Additionally, the Waxman Graph and the Watts–Strogatz model are used to provide more diverse network topologies, as they are well-suited for modeling Internet backbone structures and small-world networks, respectively (Waxman, 1988; Duncan J Watts, 1998). Watts & Strogatz (1998) The Watts–Strogatz model, applied with a larger number of nodes, creates a challenging network topology. Its properties, including high clustering, low path length, and limited scale-free characteristics, make it appropriate for analyzing complex scenarios. The resources used for simulating the network operations in this experiment include a Google Colab environment with an Intel Xeon 2vCPU @ 2.2GHz, 13GB RAM, and an NVIDIA Tesla K80 GPU for the inference phase and execution time evaluation. For the training process, an HPC system with AMD EPYC 7742 @ 2.25GHz (64 cores x 2) and 128GB x 16 RAM was used.

4.1 Assignment of Switch and Controller

In this research, the method used for assigning the roles of nodes within the network graph as switches or controllers is based on the procedure from (Alowa & Fevens, 2020), with the following steps:

1. For the nodes that are not yet assigned as controllers or switches, assign the node with the highest number of connected neighboring nodes as the controller. In cases where multiple nodes have the same highest number of neighboring nodes, choose the node with the lowest ID to be the controller.
2. Assign all nodes that are directly connected to the controller chosen in step 1 as switches.
3. If the number of controllers is still less than the required number of controllers, go back to Step 1. Otherwise, assign the remaining nodes as switches.

4.2 Network Environment

This research focuses on the in-band communication of the control plane in network simulations. In this experimental network, the generated loads are measured in pps. Each controller operates with an equal

capacity of 44,700 pps. A threshold is set at 80 percent of this capacity, which is 38,000 pps, to trigger various algorithms for load balancing. Whenever a controller's load exceeds this threshold, load balancing is initiated. The details of the network graph are presented in Table 1.

Table 1: Network Topology Information

Network size	**Topology**	**No. of switches**	**No. of controller**	**Summation load of all switches (pps)**
Small	TW	64	7	212,800
Medium	Colt Telecom	133	13	395,200
Medium	Waxman Graph (n = 170)	150	16	486,400
Large	Barabási–Albert ($n = 210, m = 5$)	190	20	608,000
Extra-large	Barabási–Albert ($n = 300, m = 10$)	272	28	851,200
Extra-large	Watts-Strogatz (n = 400, k = 10, p = 0.1)	363	37	1,124,800

This table divides the network topology into different sizes based on the total number of switches and controllers present. In the graphs generated by the graph models, n represents the total number of nodes. Additional parameters vary by graph model, i.e., m is used in the Barabási–Albert model to denote the number of edges a new node attaches to existing nodes, while in the Watts–Strogatz model, k represents the number of nearest neighbors each node is initially connected to in a ring topology, and p is the probability of edge rewiring to introduce randomness, creating a small-world network.

Each network topology has distinct characteristics, such as network diameter and the number of neighboring nodes. Additionally, the difference in the number of hops between a switch and its nearest, next-nearest, and subsequent controllers can vary significantly across topologies. In some topologies, the number of hops increases sharply between successive controllers, while in others, the increase is more gradual. These topological differences result in different performance of mapping methods across various network structures.

In each topology, the maximum total load of all switches in the network is set to approximately 80 percent of the combined threshold capacity of all controllers. For example, in the TW topology, there are a total of 64 switches and 7 controllers. The total load that all controllers can support without overloading is calculated as $7 \times 38{,}000 = 266{,}000$ pps. Therefore, the maximum total load that the 64 switches in the network can generate is $0.8 \times 266{,}000 = 212{,}800$ pps.

Switches generate loads that vary among each switch using a uniform distribution. The range of values to be randomly selected for each switch's load is specified, along with the probabilities as presented in Table 2. This setup introduces variability while reflecting realistic network conditions.

Table 2: Range of Generated Switch Load Value

Load type	**Range of load value (pps)**	**Probability**
Low	1,000 - 1,400	0.3167
Medium	1,900 - 2,400	0.3167
High	5,500 - 6,000	0.3167
Very high	17,500 - 18,500	0.0500

According to the research in (Lai et al., 2022), the probability of switches generating varying loads for each load category is not uniform. Switches categorized as high load types exhibit the lowest probability of occurrence, which aligns with typical network traffic patterns where high-load devices are less common. After all switches generate their loads, these values are forwarded to the corresponding controllers, enabling the calculation of the total load per controller. This determines if switch migration is necessary for load balancing.

Each complete round of load generation across the network is regarded as one period. Following each period, new load values are randomly generated for the switches, while maintaining consistency with previous load types to reflect gradual traffic changes as outlined in Table 3.

Table 3: Switch Load Type Change Probability

Load type earlier period	Load type next period	Probability (respectively)
Low	Low, Medium	0.500, 0.500
Medium	Medium, Low, High	0.333, 0.333, 0.333
High	High, Medium, very high	0.475, 0.475, 0.050
Very high	Very high, High	0.050, 0.950

The load type categorization for each period in this research is based on the characteristics of real-world network traffic used by Internet Service Providers (ISPs) (Mozo et al., 2018) and internal organizational networks (Information Technology Service Center (ITSC), Chiang Mai University, n.d.). Network traffic is observed to gradually change, increasing or decreasing over time, without sudden changes.

For example, if switch s_1 generates a load of 5,875 pps, categorized as a high load in the current period, in the subsequent period, it will generate a load within the same high category with a probability of 0.475, a medium category with a probability of 0.475, or a very high category with a probability of 0.050.

4.3 ANN Models

The ANN models were trained and implemented using TensorFlow library (Abadi et al., 2016) version 2.13.0. Details regarding the model utilized for evaluation are provided in the table below.

Table 4: ANN Models Configuration using in Each Topology

Topologies	Number of nodes			Number of trained model (r)
	HL1	HL2	HL3	
TW	210	200	170	12
Colt Telecom	80	70	50	24
Waxman Graph ($n = 170$)	120	110	90	7
Barabási–Albert ($n = 210, m = 5$)	120	110	90	10
Barabási–Albert ($n = 300, m = 10$)	120	110	90	13
Watts–Strogatz ($n = 400,\ k = 10,\ p = 0.1$)	120	110	90	73

From Table 4, the number of models trained to predict results varies across different topologies. This variation is due to the unequal number of datasets in each training process, where the models' predictions result in controller overload. For example, the Colt Telecom topology requires the highest number of models trained because numerous datasets result in the model predicting controller overload in each training process as referenced from Figure 3. This situation requires multiple model training iterations to ensure that the remaining datasets constitute less than 10 percent of the initial datasets.

The number of nodes in each hidden layer for the various topologies was determined through the experimental selection of values that minimize the occurrence of controller overloads across all 40,000 datasets while maintaining a low total energy cost (e), as predicted by the model. Hyperparameters, including 8,000 epochs and a learning rate of 0.0003 for the model's optimizer, were selected through experimental tuning to optimize performance. In this experiment, each hidden setting was tested with varied values of $\varepsilon,\ \gamma$, and β, selecting the model that achieved the best results in terms of the lowest total energy cost e and the lowest number of overloaded datasets. These criteria, along with the specified

hyperparameters, guided the setup of the number of nodes in each hidden layer, as outlined in Table 5 through Table 10.

TW Topology

Table 5: Number of Nodes in Hidden Layers of Model using in TW Topology

Number of nodes			Total e	Overload datasets
HL1	HL2	HL3		
80	70	50	12,976,295,116	16,635
120	110	90	13,115,276,297	7,412
210	200	170	13,277,765,746	7,098
310	300	270	13,258,454,933	8,094

From Table 5, the configuration of 210, 200, and 170 nodes in each hidden layer was selected. Although total e is higher than other options, it can be decreased by increasing the number of epochs in the hyperparameter settings during the training phase.

Colt Telecom Topology

Table 6: Number of Nodes in Hidden Layers of Model using in Colt Telecom Topology.

Number of nodes			Total e	Overload datasets
HL1	HL2	HL3		
80	70	50	29,438,977,157	30,423
120	110	90	28,891,409,270	31,812
210	200	170	28,677,252,582	32,510
310	300	270	29,244,661,516	35,243

From Table 6, the configuration of 80, 70, and 50 nodes in each hidden layer was selected because it results in the lowest number of controller overloads according to the model's predictions. Additionally, adjusting the model's hyperparameter to increase the epochs during training can further reduce the value of total e.

Waxman Graph Topology

Table 7: Number of Nodes in Hidden Layers of Model using in Waxman Graph Topology.

Number of nodes			Total e	Overload datasets
HL1	HL2	HL3		
80	70	50	27,690,138,917	29,079
120	110	90	28,392,972,907	22,181
210	200	170	28,218,805,087	23,321
310	300	270	28,527,886,758	22,449

From Table 7, the configuration of 120, 110, and 90 nodes in each hidden layer was selected because these configurations of hidden layers have the lowest number of controller overloads as represented in the overload datasets.

Barabási–Albert Topologies

Table 8: Number of Nodes in Hidden Layers of Model using in Barabási–Albert ($n = 210, m = 5$) Topology

Number of nodes			Total e	Overload datasets
HL1	HL2	HL3		
80	70	50	28,295,501,223	30,961
120	110	90	27,705,551,999	17,555
210	200	170	27,798,538,117	17,785
310	300	270	27,790,129,832	20,401

Table 9: Number of Nodes in Hidden Layers of Model using in Barabási–Albert ($n = 300, m = 10$) Topology

Number of nodes			Total e	Overload datasets
HL1	HL2	HL3		
80	70	50	29,438,977,157	30,423
120	110	90	28,891,409,270	31,812
210	200	170	28,677,252,582	32,510
310	300	270	29,244,661,516	35,243

Table 8 and Table 9 select the same number of nodes in each hidden layer 120, 110, and 90 for models used in the Barabási–Albert topology with parameters ($n = 210, m = 5$) and ($n = 300, m = 10$), respectively. These two topologies, categorized as large and extra-large network sizes, respectively, demonstrate that using the maximum number of nodes in ANNs is not always necessary to achieve best results.

Watts-Strogatz Model Topology

Table 10: Number of Nodes in hidden Layers of Model using in Watts–Strogatz Topology

Number of nodes			Total e	Overload datasets
HL1	HL2	HL3		
80	70	50	60,772,667,983	38,475
120	110	90	59,320,592,180	35,879
210	200	170	76,807,740,488	39,910
310	300	270	124,811,901,599	39,945

From Table 10, the configuration of 120, 110, and 90 nodes in each hidden layer was selected because it represents the lowest total e while maintaining the lowest number of controller overloads compared to other hidden layer configurations. In addition, For the Proposed ANN-Fast method, to rank models, a newly generated dataset of 1,000 network states is created for each topology. Overall, the hidden layer configurations were selected to minimize controller overloads while balancing the model's computational efficiency. In cases where the total error was slightly higher, increasing the number of training epochs was considered to improve performance further.

5 Experimental Results

The performance evaluation of the Proposed ANN model is assessed based on several metrics: the total energy cost from all switches and controllers as defined by Equation (2), the occurrence of total controller overloads, and the execution time of the switch-to-controllers mapping process. The dataset used for evaluation contains a total of 1,000 datasets, which are datasets that the model has never seen, representing the network state of each topology, including the load of switches at 1,000 periods.

5.1 Load Balancing Algorithms in Comparison

In this research, the performance of the Proposed ANN model is benchmarked against traditional approaches. The CP-SAT solver from the Google OR-Tools library (Perron et al., 2023), with our objective function and constraints detailed in Equations (3)-(5), is employed as a baseline to evaluate the performance of various algorithms. The CP-SAT solver is suitable for integer optimization problems, including decision or assignment problems such as our switch mapping problem (Developers, Google, n.d.). For a comprehensive comparison, the performance of the Proposed ANN model is also assessed against the ECLD and SALB algorithms, which tackle the challenge of switch migration for load balancing by employing dynamic programming and greedy methods, respectively. Note that in the tables and figures of the evaluation results, ANN-Full refers to the Proposed ANN-Full, and ANN-Fast refers to the Proposed ANN-Fast.

5.1.1 Total Energy Cost

Table 11: Total Energy Cost from all Topologies

Algorithms	Topologies					
	TW	Colt Telecom	Waxman Graph ($n = 170$)	Barabási–Albert ($n = 210$, $m = 5$)	Barabási–Albert ($n = 300$, $m = 10$)	Watts–Strogatz ($n = 400$, $k = 10$, $p = 0.1$)
CP-SAT	310,702,164	659,336,836	625,527,931	650,253,737	798,589,112	1,177,774,971
ECLD	313,275,750	675,208,591	641,709,846	677,367,560	823,926,141	1,206,805,679
SALB	314,709,836	708,711,510	643,212,480	687,104,697	835,885,041	1,204,468,524
ANN-Full	321,370,169	718,001,997	647,205,014	673,282,095	829,960,867	1,280,337,854
ANN-Fast	323,682,289	725,604,167	649,556,435	675,593,793	839,817,258	1,289,799,319

Table 12: Ratio of Total Energy Cost from all Topologies Compare with CP-SAT

Algorithms	Topologies					
	TW	Colt Telecom	Waxman Graph ($n = 170$)	Barabási–Albert ($n = 210$, $m = 5$)	Barabási–Albert ($n = 300$, $m = 10$)	Watts–Strogatz ($n = 400$, $k = 10$, $p = 0.1$)
CP-SAT	1.00	1.00	1.00	1.00	1.00	1.00
ECLD	1.00	1.02	1.03	1.04	1.03	1.02
SALB	1.01	1.07	1.03	1.06	1.05	1.02
ANN-Full	1.03	1.08	1.03	1.04	1.04	1.09
ANN-Fast	1.04	1.10	1.04	1.04	1.05	1.10

In this evaluation metric, if a model's prediction from any dataset leads to controller overload, the results from that dataset are excluded from the records, irrespective of the algorithm employed. The Proposed ANN-Full and Proposed ANN-Fast models, as shown in Table 11, demonstrate total energy costs comparable to those of other algorithms evaluated. Their performance, when compared to the CP-SAT method, is expressed as ratios and presented in Table 12.

The ratio of total energy cost, when comparing the efficiency of Proposed ANN-Full and Proposed ANN-Fast, indicates that their performance is similar to that of other compared algorithms, particularly in a large topology, i.e., Barabási–Albert ($n = 210, m = 5$) and extra-large topologies, i.e., Barabási–Albert ($n = 300, m = 10$) and Watts-Strogatz. It was observed that both models could predict outcomes that yield total energy costs comparable to those achieved by the ECLD algorithm, which utilizes dynamic programming for its solutions.

The Proposed ANN-Fast shows a higher ratio value than Proposed ANN-Full. This is because when the model predicts results that do not lead to controller overload, it discontinues input to other models, even if the results might not be the most optimized. This strategy reduces processing time but may not always achieve the minimized energy cost.

5.1.2 Overload Controllers

Table 13: Number of Controller Overload Datasets out of 1,000 Datasets

Algorithms	Topologies					
	TW	Colt Telecom	Waxman Graph ($n = 170$)	Barabási–Albert ($n = 210$, $m = 5$)	Barabási–Albert ($n = 300$, $m = 10$)	Watts–Strogatz ($n = 400$, $k = 10$, $p = 0.1$)
CP-SAT	0	0	0	0	0	0
ECLD	0	0	0	0	0	0
SALB	0	0	0	0	0	0
ANN-Full	2	54	82	29	91	128
ANN-Fast	2	54	82	29	91	128

This metric evaluates the Proposed ANN model using 1,000 datasets. In some cases, predictions from the model result in controller overload. Table 13 shows the number of data instances that led to controller overload across each network topology.

Analysis of the datasets where controller overload occurred revealed that no cases of overload were detected with traditional algorithms, whereas the Proposed ANN models predicted some cases of overload. However, these cases can be considered within acceptable rates when compared to the total number of datasets. To further enhance the efficiency of the model, one could increase the number of trained models used in aggregation to select predictions that avoid causing controller overload.

In a limited number of cases, as considered in this study, overload in the Proposed ANN models can be attributed to how the model maps switches to controllers. The variable $y_{i,j}$ is used to map which controller each switch should connect to, where the value is either 0 or 1. However, the ANN produces confidence values $\hat{y}_{i,j}$, which are continuous values between 0 and 1, generated by the Softmax function commonly used in multi-class classification. The ANN generates varying confidence values for multiple controllers associated with the same switch. To determine the final mapping, the controller with the highest confidence value is selected and assigned a value of 1, while all other controllers are assigned a value of 0. This assignment step can lead to certain controllers receiving a higher load than the ANN originally calculated.

This issue is more likely in environments with a high number of switches carrying very high load types, where precise load distribution is critical. To mitigate this issue, we incorporated the confidence mapping into the loss function as shown in Equation (13), which drives $\hat{y}_{i,j}$ values to approach 1. This adjustment improves the model's ability to make more decisive and accurate assignments, but overload can still occur due to the trade-off between faster execution time and optimal load distribution. Addressing this limitation remains an open challenge for future research, where more adaptive or dynamic load balancing mechanisms could further minimize controller overload.

5.1.3 Execution Time of Switch Migration Selection

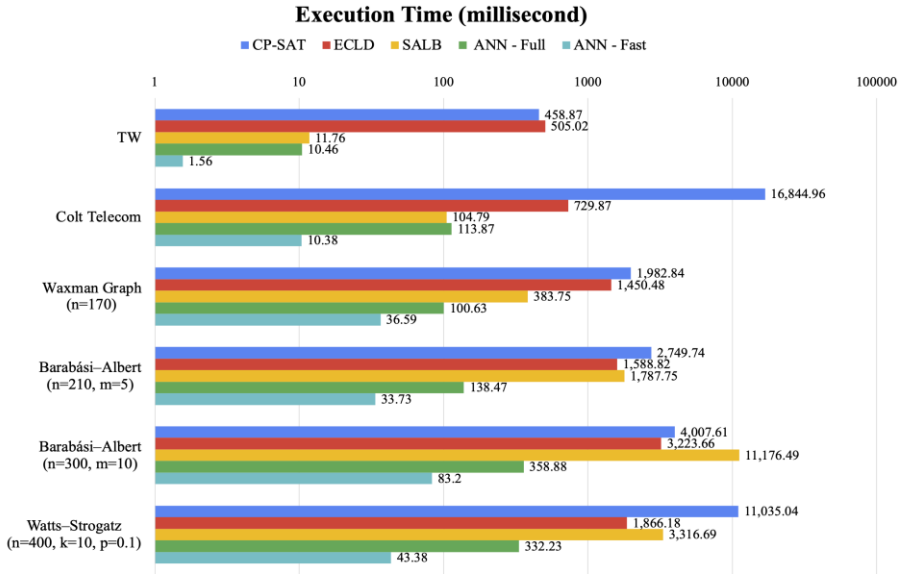

Figure 6: Average Execution Time Per Data Result

From the graph presented in Figure 6, it is evident that the Proposed ANN-Full and Proposed ANN-Fast require the least amount of time compared to other algorithms. This observation confirms that utilizing

neural networks for load balancing can enhance network efficiency. The advantage lies in the ability of these models to initiate the switch migration process immediately upon detecting a controller overload, thereby utilizing a shorter response time to solve the problem compared to traditional algorithms.

The ECLD algorithm, which uses dynamic programming, also shows better performance compared to CP-SAT but is still significantly slower than the Proposed ANN models. Similarly, the SALB algorithm, which is a greedy algorithm, offers some improvements in execution time but does not match the efficiency of the ANN models.

In the TW topology, CP-SAT takes significantly longer, with the Proposed ANN-Fast model being approximately 294 times faster. While ECLD and SALB also take longer than the ANN models, their execution times are still less than that of CP-SAT. Similarly, in the Colt Telecom topology, CP-SAT is approximately 1,623 times slower than the Proposed ANN-Fast, with ECLD and SALB being slower as well, but less significantly.

Waxman, (1988). In the Waxman Graph topology, CP-SAT takes the longest time, followed by ECLD and SALB, which are based on dynamic programming and greedy algorithms. The Proposed ANN-Fast is 54 times faster than CP-SAT, while the Proposed ANN-Full executes faster than CP-SAT by more than 19 times.

For the Barabási–Albert (n=210, m=5) topology, CP-SAT is approximately 81 times slower than the proposed ANN-Fast model, while ECLD and SALB demonstrate better performance than CP-SAT but remain significantly slower than the ANN models. Similarly, for the Barabási–Albert (n=300, m=10) topology, CP-SAT is approximately 48 times slower than the proposed ANN-Fast model, with ECLD and SALB performing faster than CP-SAT but still slower than the ANN models.

Finally, for the Watts-Strogatz topology, CP-SAT is over 254 times slower than the Proposed ANN-Fast. The Proposed ANN-Full takes longer to execute due to the number of trained models required to find an optimal solution for balancing controller loads and reducing energy consumption.

However, this improvement in average execution time introduces a trade-off between computational efficiency and energy consumption. The shorter execution time of the Proposed ANN models may result in slightly higher energy consumption due to suboptimal switch-to-controller mappings. While the models prioritize faster migration decisions to manage controller overloads, these decisions might not always result in the most energy-efficient configurations. Nonetheless, this trade-off is acceptable in dynamic SDN environments where network adaptability and responsiveness are critical. The reduced average execution time significantly enhances overall network performance, making the minor increase in energy consumption a reasonable compromise.

These considerable differences highlight the computational efficiency of both ANN models, with the Proposed ANN-Fast being the most efficient. The faster execution time of the Proposed ANN models enables quick adaptation to network changes, making them well-suited for real-time load balancing in dynamic SDN environments. Although this improvement comes with a slight increase in energy consumption compared to the optimal solution, this trade-off remains reasonable and justifiable. The balance between faster execution and slightly higher energy usage makes both models highly practical and effective for maintaining network performance.

6 Conclusions

This research proposes the Artificial Neural Network models with unsupervised learning to address the switch migration problem in load balancing with a focus on energy awareness in SDN. In large networks,

the complexity of the algorithm impacts the processing time required to find an optimal solution. This model predicts the mapping of switches to controllers during occurrences of controller overload. It considers factors such as load amount and hop count, which affect the network's energy consumption.

The models were trained using an unsupervised learning technique with a dataset that includes each switch's load, the hop count from each switch to every controller, and the controller's capacity. During the training phase, the performance of the models involves formulating equations to serve as the loss function, corresponding to the problem's objective function.

For model inference, two methods were proposed. The first method, called Proposed ANN-Full, performs inference on all models to predict results from the same input. It selects the best solution based on the model that optimizes energy consumption without causing controller overload. Additionally, the research introduces the Proposed ANN-Fast method, which prioritizes the most optimal models based on their predicted results. This method prioritizes the top-ranked model for inference. If controller overload still occurs, the next-ranked model is used for inference, and so on, until no controller overload is detected. This approach is expected to require inference from fewer models compared to the Proposed ANN-Full method, resulting in faster processing times but with the potential trade-off of slightly less optimal energy consumption (Santhosh & Prasad 2023).

In terms of performance, the proposed model achieves comparable results to conventional algorithms such as CP-SAT, dynamic programming, and greedy algorithms in switch-controller mapping while minimizing energy consumption. However, a significant advantage of this model is the reduced computation time needed for selecting switches and controllers, particularly in large network topologies. This reduction enhances network efficiency by rapidly resolving problems when controller overload occurs.

Moreover, these proposed unsupervised ANNs can be easily implemented for various SDN network topologies. Since unsupervised learning eliminates the need for labeled datasets, the only data required for training the model is the network state. This state information includes switch loads, hop counts, and controller capacities.

Future research should explore expanding the model to support networks with variable controller capacities and even more diverse switch load distributions, improving its applicability to real-world scenarios. Additionally, refining both the model itself and the Proposed ANN-Fast inference method to further enhance speed and energy efficiency represents a promising direction for continued study.

Acknowledgement

This work was supported by Erawan HPC Project, Information Technology Service Center (ITSC), Chiang Mai University, Chiang Mai, Thailand. The authors would like to thank ITSC for providing the computing resources used during the model training process.

References

[1] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., … Zheng, X. (2016). TensorFlow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*. https://doi.org/10.48550/arXiv.1603.04467.

[2] Abuarqoub, A. (2020). A Review of the Control Plane Scalability Approaches in Software Defined Networking. *Future Internet, 12*(3), 49. https://doi.org/10.3390/fi12030049.
[3] Ahmad, S., & Mir, A. H. (2021). Scalability, consistency, reliability, and security in SDN controllers: A survey of diverse SDN controllers. *Journal of Network and Systems Management, 29*(9). https://doi.org/10.1007/s10922-020-09575-4.
[4] Al Sallami, N. M., & Al Alousi, S. A. (2013). Load balancing with neural network. *International Journal of Advanced Computer Science and Applications*, *4*(10).
[5] Alhilali, A. H., & Montazerolghaem, A. (2023). Artificial intelligence-based load balancing in SDN: A comprehensive survey. *Internet of Things*, 22, 100814. https://doi.org/10.1016/j.iot.2023.100814.
[6] Alowa, A., & Fevens, T. (2020). Towards minimum inter-controller delay time in software-defined networking. *Procedia Computer Science, 175*, 395–402. https://doi.org/10.1016/j.procs.2020.07.056.
[7] Barabási, A.-L., & Albert, R. (1999). Emergence of scaling in random networks. *Science, 286*(5439), 509–512. https://doi.org/10.1126/science.286.5439.509.
[8] Blial, O., Ben Mamoun, M., & Benaini, R. (2016). An overview on SDN architectures with multiple controllers. *Journal of Computer Networks and Communications*, *2016*(1), 9396525. https://doi.org/10.1155/2016/9396525
[9] Carrascal, D., Rojas, E., Arco, J. M., Lopez-Pajares, D., Alvarez-Horcajo, J., & Carral, J. A. (2023). A Comprehensive Survey of In-Band Control in SDN: Challenges and Opportunities. *Electronics, 12*(6), 1265. https://doi.org/10.3390/electronics12061265
[10] Choumas, K., Giatsios, D., Flegkas, P., & Korakis, T. (2020). SDN Controller Placement and Switch Assignment for Low Power IoT. *Electronics, 9*(2), 325. https://doi.org/10.3390/electronics9020325.
[11] Fröhlich, P., Gelenbe, E., Fiołka, J., Chęciński, J., Nowak, M., & Filus, Z. (2021). Smart SDN Management of Fog Services to Optimize QoS and Energy. *Sensors, 21*(9), 3105. https://doi.org/10.3390/s21093105.
[12] Giliberto, M., Arena, F., & Pau, G. (2019). A fuzzy-based Solution for Optimized Management of Energy Consumption in e-bikes. *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications, 10*(3), 45-64.
[13] Google Developers. (n.d.). Integer optimization. Retrieved November 23, 2023, from https://developers.google.com/optimization/mip.
[14] Hagberg, A. A., Swart, P. J., & Schult, D. A. (2008). Exploring network structure, dynamics, and function using NetworkX. In *Proceedings of the 7th Python in Science Conference (SciPy 2008)*. https://doi.org/10.25080/TCWV9851.
[15] Hussain, M. W., Reddy, K. H. K., Rodrigues, J. J. P. C., & Roy, D. S. (2021). An indirect controller-legacy switch forwarding scheme for link discovery in hybrid SDN. *IEEE Systems Journal, 15*(2), 3142–3149. https://doi.org/10.1109/JSYST.2020.3011902.
[16] Information Technology Service Center (ITSC), Chiang Mai University. (2024). All Traffic ENG. Retrieved May 10, 2024, from https://cmu.to/h0E9h.
[17] International Energy Agency (IEA). (2023). Tracking clean energy progress 2023. IEA, Paris. Retrieved from https://www.iea.org/reports/tracking-clean-energy-progress-2023 (Licensed under CC BY 4.0).
[18] Isyaku, B., Mohd Zahid, M. S., Bte Kamat, M., Abu Bakar, K., & Ghaleb, F. A. (2020). Software Defined Networking Flow Table Management of OpenFlow Switches Performance and Security Challenges: A Survey. *Future Internet, 12*(9), 147. https://doi.org/10.3390/fi12090147.
[19] Jalili, A., Keshtgari, M., & Akbari, R. (2020). A new framework for reliable control placement in software-defined networks based on a multi-criteria clustering approach. *Soft Computing, 24*(4), 2897–2916. https://doi.org/10.1007/s00500-019-04070-8.

[20] Jyothi, B., & Mary Gladence, L. (2024). Enhanced Accuracy for Lung Adenocarcinoma (LUAD) Prediction based UMAP Feature Using Artificial Neural Network. *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications, 15*(4), 380-394. https://doi.org/10.58346/JOWUA.2024.I4.026

[21] Kandil, M., Awad, M. K., Alotaibi, E. M., & Mohammadi, R. (2022, December). Q-learning and simulated annealing-based routing for software-defined networks. In *2022 International conference on computer and applications (ICCA)* .1-10. IEEE. https://doi.org/10.1109/ICCA56443.2022.10039651.

[22] Kim, T., Lee, T., Kim, K. H., Yeh, H., & Hong, M. (2013, October). An efficient packet processing protocol based on exchanging messages between switches and controller in OpenFlow networks. In *2013 10th International Conference and Expo on Emerging Technologies for a Smarter World (CEWIT)* (1-5). IEEE. DOI: 10.1109/CEWIT.2013.6713746

[23] Knight, S., Nguyen, H. X., Falkner, N., Bowden, R., & Roughan, M. (2011). The Internet Topology Zoo. *IEEE Journal on Selected Areas in Communications, 29*(9), 1765–1775. https://doi.org/10.1109/JSAC.2011.111002.

[24] Kooshki, S. A., Zeinabadi, H., Jafarnezhad, A., & Kooshki, H. A. (2016). Applications of Fuzzy Logic and Artificial Neural Networks in Evaluation and Ranking of Teachers Based on "Framework for Teaching" Model. *International Academic Journal of Innovative Research, 3*(1), 1–10.

[25] Kouroshnia, H., & Farokhi, F. (2018). Adapting and optimization in small signal modeling of nanoscale MOS-FET Using artificial recurrent neural networks. *International Academic Journal of Science and Engineering, 5*(2), 227–234. https://doi.org/10.9756/IAJSE/V5I1/1810038

[26] Kumar, A., Anand, D., Jha, S., Joshi, G.P., Cho, W. (2022). Optimized load balancing technique for software defined network. *Computers, Materials & Continua, 72*(1), 1409–1426. https://doi.org/10.32604/cmc.2022.024970.

[27] Kumari, A., & Sairam, A. S. (2024). Look ahead before you leap: SDN switch migration scheduling for load balancing. *Wireless Personal Communications*, 1-30.

[28] Kurroliya, K., Mohanty, S., Sahoo, B., & Kanodia, K. (2020, February). Minimizing energy consumption in software defined networks. In *2020 7th International Conference on Signal Processing and Integrated Networks (SPIN)*. 885-890. IEEE. https://doi.org/10.1109/SPIN48934.2020.9070985.

[29] Lai, W.-K., Wang, Y.-C., Chen, Y.-C., & Tsai, Z.-T. (2022). TSSM: Time-sharing switch migration to balance loads of distributed SDN controllers. *IEEE Transactions on Network and Service Management, 19*(2), 1585–1597. https://doi.org/10.1109/TNSM.2022.3146834.

[30] Martín, I., Troia, S., Hernández, J. A., Rodríguez, A., Musumeci, F., Maier, G., Alvizu, R., & Gonzalez de Dios, O. (2019). Machine learning-based routing and wavelength assignment in software-defined optical networks. *IEEE Transactions on Network and Service Management, 16*(3), 871–883. https://doi.org/10.1109/TNSM.2019.2927867.

[31] Mozo, A., Ordozgoiti, B., & Gómez-Canaval, S. (2018). Forecasting short-term data center network traffic load with convolutional neural networks. *PLOS ONE, 13*, 1–31. https://doi.org/10.1371/journal.pone.0191939.

[32] Neves, R. H., Silva, A. A. A., Gava, V., Azevedo, M. T., Sandoval, J. F. R., Oliveira, F. S., Guelfi, A. E., & Kofuji, S. T. (2022). DoS attack on SDN: A study on control plane strategies in-band and out-of-band (Version 1). *Research Square.* https://doi.org/10.21203/rs.3.rs-1816989/v1.

[33] Pathan, M. N., Muntaha, M., Sharmin, S., Saha, S., Uddin, M. A., Nur, F. N., & Aryal, S. (2024). Priority-based energy and load-aware routing algorithms for SDN-enabled data center network. *Computer Networks, 240*, 110166. https://doi.org/10.1016/j.comnet.2023.110166.

[34] Perron, N., Chemnitz, M., Fischer, B., Junaid, S., Schmidt, M., & Morandotti, R. (2023, May). All-optical digital processing in Carbon Disulfide liquid-core optical fiber. In *2023 Conference on Lasers and Electro-Optics (CLEO)*. 1-2. IEEE. https://doi.org/10.4230/LIPIcs.CP.2023.3.

[35] Priyadarsini, M., Kumar, S., Bera, P., & Rahman, M. A. (2020). An energy-efficient load distribution framework for SDN controllers. *Computing, 102*(9), 2073–2098. https://doi.org/10.1007/s00607-019-00751-2.

[36] Ruelas, A. M., & Rothenberg, C. E. (2018, October). A load balancing method based on artificial neural networks for knowledge-defined data center networking. In *Proceedings of the 10th Latin America Networking Conference* (106-109). https://doi.org/10.1145/3277103.3277135.

[37] Santhosh, G., & Prasad, K. V. (2023). Energy Saving Scheme for Compressed Data Sensing Towards Improving Network Lifetime for Cluster based WSN. *Journal of Internet Services and Information Security, 13*(1), 64-77. https://doi.org/10.58346/JISIS.2023.I1.007

[38] Semong, T., Maupong, T., Anokye, S., Kehulakae, K., Dimakatso, S., Boipelo, G., & Sarefo, S. (2020). Intelligent Load Balancing Techniques in Software Defined Networks: A Survey. *Electronics, 9*(7), 1091. https://doi.org/10.3390/electronics9071091.

[39] Shuja, J., Ahmad, R. W., Gani, A., Ahmed, A. I. A., Siddiqa, A., Nisar, K., Khan, S. U., & Zomaya, A. Y. (2017). Greening emerging IT technologies: Techniques and practices. *Journal of Internet Services and Applications, 8*(9). https://doi.org/10.1186/s13174-017-0060-5.

[40] Somwong, P., & Somchit, Y. (2024, January). Energy-Aware Controller Load Distribution in Software-Defined Networking. In *2024 International Conference on Electronics, Information, and Communication (ICEIC)* (1-4). IEEE. DOI: 10.1109/ICEIC61013.2024.10457141

[41] Sufiev, H., & Haddad, Y. (2016, November). A dynamic load balancing architecture for SDN. In *2016 IEEE International conference on the science of electrical engineering (ICSEE)* (1-3). IEEE.

[42] Torkzadeh, S., Soltanizadeh, H., & Orouji, A. A. (2021). Energy-aware routing considering load balancing for SDN: A minimum graph-based Ant Colony Optimization. *Cluster Computing, 24*(3), 2293–2312. https://doi.org/10.1007/s10586-021-03263-x.

[43] Tosounidis, V., Pavlidis, G., & Sakellariou, I. (2020, September). Deep Q-learning for load balancing traffic in SDN networks. In *11th Hellenic Conference on Artificial Intelligence.* 135-143. https://doi.org/10.1145/3411408.3411423.

[44] Villarrubia, G., De Paz, J. F., Chamoso, P., & De la Prieta, F. (2018). Artificial neural networks used in optimization problems. *Neurocomputing, 272*, 10–16. https://doi.org/10.1016/j.neucom.2017.04.075.

[45] Watts, D. J., & Strogatz, S. H. (1998). Collective dynamics of 'small-world' networks. *Nature, 393*, 440–442. https://doi.org/10.1038/30918.

[46] Waxman, B. M. (1988). Routing of multipoint connections. *IEEE Journal on Selected Areas in Communications, 6*(9), 1617–1622. https://doi.org/10.1109/49.12889.

[47] WilsonPrakash, S., & Deepalakshmi, P. (2019, April). Artificial neural network-based load balancing on software defined networking. In *2019 IEEE International Conference on Intelligent Techniques in Control, Optimization and Signal Processing (INCOS)*. 1-4. IEEE. https://doi.org/10.1109/INCOS45849.2019.8951365.

[48] Xiang, M., Chen, M., Wang, D., & Luo, Z. (2022). Deep reinforcement learning-based load balancing strategy for multiple controllers in SDN. *e-Prime - Advances in Electrical Engineering, Electronics and Energy, 2*, 100038. https://doi.org/10.1016/j.prime.2022.100038.

[49] Zhao, Y., Wang, X., He, Q., Zhang, C., & Huang, M. (2021). PLOFR: An online flow route framework for power saving and load balance in SDN. *IEEE Systems Journal, 15*(1), 526–537. https://doi.org/10.1109/JSYST.2020.3010971

[50] Zhou, Y., Zheng, K., Ni, W., & Liu, R. P. (2018). Elastic switch migration for control plane load balancing in SDN. *IEEE Access, 6*, 3909–3919. https://doi.org/10.1109/ACCESS.2018.2795576.

[51] Zhu, R., Wang, H., Gao, Y., Yi, S., & Zhu, F. (2015). Energy saving and load balancing for SDN based on multi-objective particle swarm optimization. In *Algorithms and Architectures for Parallel Processing: 15th International Conference, ICA3PP 2015, Zhangjiajie, China, November 18-20, 2015, Proceedings, Part III 15*. 176-189. Springer International Publishing. https://doi.org/10.1007/978-3-319-27137-8_14

[52] Zou, J., Han, Y., & So, S. S. (2009). Overview of artificial neural networks. *Artificial neural networks: methods and applications*, 14-22. https://doi.org/10.1007/978-1-60327-101-1_2.

Authors Biography

Poom Somwong received a B.Eng. in Information Systems and Network Engineering and an M.Eng. in Computer Engineering from Chiang Mai University, Thailand. His research interests include computer networks, software-defined networking, and machine learning.

Karn Patanukhom received a B.Eng. in Electrical Engineering from Chiang Mai University, Thailand, in 2003. He received an M.Eng. and a Ph.D. in Electronics from the Tokyo Institute of Technology, Japan, in 2006 and 2009, respectively. He is currently an Assistant Professor at Chiang Mai University. His research interests include image processing, machine learning, computer vision, and natural language processing.

Yuthapong Somchit received a B.Eng. in Electrical Engineering from Chiang Mai University, Thailand, in 2000. He received an M.Eng. and a Ph.D. in Communications and Integrated Systems from the Tokyo Institute of Technology, Japan, in 2003 and 2006, respectively. He is currently an Assistant Professor in the Department of Computer Engineering, Faculty of Engineering, at Chiang Mai University. His research focuses on computer networking, network security, software-defined networking, and machine learning.