CONFLICT FLOW AVOIDED PROACTIVE REROUTING ALGORITHM USING ONLINE ACTIVE LEARNING FOR EFFICIENT TRANSMISSION OF DATASTREAM IN SOFTWARE DEFINED NETWORKS

Kalaivani Subramaniam [1] and Sumathi Arumugam [2]

[1] Department of Computer Science, KPR College of Arts Science and Research, Coimbatore-641407, Tamilnadu, India
[2] Department of Information Technology, KPR College of Arts Science and Research, Coimbatore-641407, Tamilnadu, India

ABSTRACT

Traditional or default Software-Defined Networking (SDN) generally relies on reactive or static routing, where a new flow entry is added for a new arriving packet in a switch. This can create inefficiencies in processing dynamic network conditions. To solve this issue, a Machine Learning-based Proactive Rerouting Scheme (MLPRS) has been developed to dynamically balance load in real-time networks by rerouting flow and enhancing Quality of Service (QoS) compared to the default SDN. However, SDN can face challenges from conflicting flows occurring due to priority and action adjustments that may affect network throughput and bandwidth. Therefore, this article introduces an Online Active ML-based Conflict Flow Avoided Proactive Rerouting Algorithm (OAMLCFAPRA) to discriminate between normal and conflict flows in SDN based on the behavioural changes of flows over time. To achieve this, an online learning algorithm with a customized weighting scheme called Iterative Confidence-Weighted Learning (ICWL) is executed in the controller plane. Initially, it preprocesses the generated flows and trains the ICWL to identify them as normal and conflicting based on their behavioural features. Then, the priorities of each flow are assigned by the ICWL, and the normal flows are directed to OpenFlow. Additionally, the ICWL classifies the conflicting flows into several types based on their features such as priority, IP address, and action. Moreover, the betweenness centrality algorithm determines each link's significance, and the load on each link is monitored. When the most significant link is overloaded, the flow is rerouted to prevent congestion in real-time. Finally, experimental results show that the ICWL achieves high accuracy in classifying conflict flow types and the OAMLCFAPRA increases network throughput and bandwidth compared to conventional algorithms.

KEYWORDS

SDN, Flow classification, Conflict flows, Online active learning, Rerouting

1. INTRODUCTION

The demands of contemporary data centre environments and network applications cannot be fully accommodated by conventional network design. Therefore, SDN was created to allow engineers, cloud, and network managers to stay up to date with constantly evolving business needs through a centralized management console [1-2]. Additionally, SDN incorporates a number of network technologies to ensure the network is resilient and extensible, meeting the needs of storage systems and virtualized servers seen in today's data centres [3-4]. Furthermore, the initial purpose of SDN was to design, construct, and manage networks [5-6]. Because the network control and

forwarding planes are separated, this enables direct network programmability and independence of the basic planning of network services and applications [7-8]. SDN is perfect for the dynamic nature of contemporary high-bandwidth applications since it is often affordable, controllable, dynamic, and adaptable [9-10].

SDN offers a framework for virtualized execution that separates network control operations from network forwarding traffic [11]. The SDN controller permits complicated network configuration in addition to the integration of several network devices (such as routers, switches, and access points) that facilitate the execution of diverse network management functions [12-14]. SDN attempts to give users more choices over their management setup while preserving network performance criteria [15-16]. This viewpoint has led to the development of MLPRS [17], which dynamically balances load in real-time network topology to improve QoS. Applications were categorized and flow priorities were assigned based on their category using ML techniques. Each link's significance in the network was determined using the betweenness centrality algorithm, and the connections were arranged in the betweenness set. To prevent congestion, it also tracked the traffic on all connections and diverted it in real-time when the most crucial links were overloaded.

1.1. Problem Statement

SDN applications can manage systems by providing load balancing, access control, and routing, which are the most relevant features of SDN. On the other hand, SDN is impacted by a variety of conflicting flows, which can reduce network efficiency. Furthermore, conflicts in SDN arise from the influence and change of specified aspects, including priority and actions. Improving network speed and bandwidth also requires detecting and classifying competing flows.

1.2. Contributions of this Study

In this manuscript, an OAMLCFAPRA is proposed for SDN. In order to improve the load classification model, an efficient model is introduced to handle the classification of both flow types and conflicting flow types. This model combines normal flow routing, conflict flow removal, and rerouting of normal flows from congested paths. While standard ML techniques are scalable, they may not be robust to changes in the statistical properties of flow variables over time. To address this, an ICWL with a customized weighting scheme is introduced to adapt to evolving network conditions and rule changes. The key contributions of this study are: balancing and quality of service, this study primarily aims to distinguish between regular and conflict flows in SDN by observing how their behaviour evolves over time. A unified ML

- First, the generated flows are pre-processed and verified by the ICWL executed in the controller plane to obtain flow behavior features, including MAC address, IP address, and action. Based on these features, flows are recognized as normal and conflicting.
- Then, the priorities of each normal and conflicting flow are assigned by the ICWL in the controller plane, and the normal flows are directed to OpenFlow. Additionally, the conflicting flows are further categorized into several types by the ICWL in the controller plane based on their priority, IP address, and actions.
- Moreover, the betweenness centrality scheme is utilized to determine the significance of all links, and the load on each link is monitored. When the most significant link is overloaded, the flow is rerouted to prevent congestion in real time.
- Thus, this ICWL demonstrates the ability to achieve superior performance in the identification and categorization of conflicting flows in SDN, resulting in increased throughput and bandwidth usage.

The remaining sections are prepared as follows: Section 2 covers the literature survey. Section 3 explains the OAMLCFAPRA and Section 4 illustrates its effectiveness. Section 5 concludes the study and suggests future enhancements.

2. LITERATURE SURVEY

ML techniques are a widely acknowledged option for SDN traffic categorization since they leverage collected statistical data to categorize data flows. Statistical data are mostly used to categorize flows, and the various flows are divided into multiple types or groups depending on these data [18]. For SDN network traffic classification, ML algorithms are developed at the control layer and data layer. In the initial scenario, the SDN controller gathers data and, working with the application layer, uses it to execute access or forwarding strategies [19]. This task is carried out as follows: (i) the controller receives the information via OpenFlow Protocol and mines the required characteristics, and (ii) the classifier learns those characteristics and categorizes the traffic. Another such setup is data-level classification using the P4 programming language to embed ML methods into flow-level switches. Most research methodologies that address network traffic classification issues in SDN use OpenFlow protocol data to extract relevant characteristics to feed the classification process's input. This section reviews current studies on the ML-based traffic flow classification in SDN.

Liang and Su [20] presented the SDN flow instruction conflict recognition knowledge graph to accumulate the system data with flow rules. Construction instructions were set according to the definition of flow instruction conflicts, which were conflicts in one table and conflicts among multiple tables. However, its complexity was high when considering multiple conflicts or a large number of flows. Aqdus et al. [21] analysed Feed Forward Neural Network (FFNN), K-Means, and Decision Tree (DT), to identify and categorize low-rate collision flows in SDN. However, number of dropped packets and latency were high. Mohammadi et al. [22] developed a flow category discriminator and a greedy mechanism for optimal resource distribution according to the flow categories in SDN. However, it cannot discriminate conflicting flows, resulting in low scalability and reliability of the network.

During the load balancing phase, Ananth [23] created a DeepQ Residue method to examine both typical and problematic flow patterns. A hybrid Support Vector Machine (SVM) with an improved DT was used to predict accuracy and performance. However, precision and recall were low. Khairi et al. [24] developed an Extremely Fast Decision Tree (EFDT) classification method to recognize and categorize multiple conflicts inside the flow table using varying numbers of flows in SDN. However, its complexity was high and accuracy was low while using a large amount of flow data.

Han et al. [25] developed a comprehensive flow rule conflict identification technique that enhances real-time detection of explicit conflicts (both static and dynamic) and reduces false positives in implicit (dependency) conflict detection. They presented a real-time explicit conflict detection algorithm based on the Protocol-Divided Trie (PDT), which classifies flow rules by protocol type and utilizes a prefix tree for rapid matching. However, the detection time was high. Serag et al. [26] investigated ML techniques to categorize SDN traffic flows according to factors such as packet size, protocol type, and application behaviour. On the contrary, the efficiency of such techniques depends on the quality and quantity of information. For large-scale datasets, these techniques achieved low accuracy and high detection time.Abdulqadder and Aziz [27] investigated Q-learning techniques to categorize SDN traffic flows and provide security using block chaining to allow any flow inside SDN network. However, this technique implemented only for 5G network. It has to be enhanced to support 6G enabled networks. .

2.1. Research Gap

Despite significant advancements in SDN traffic classification using ML methods, existing methods face issues in handling conflicting flows effectively, leading to a degradation in network performance. Knowledge graphs and DT-based approaches to conflict flow identification have been investigated before, however these approaches have issues with real-time conflict flow categorization, scalability, and adaptability. Most methods have been shown to yield high detection times, lower accuracy, or increased computational complexity while handling large-scale networks with dynamic traffic. In addition, conventional ML models also often do not possess the ability to adapt to the evolving statistical properties of network flows, hence degrading their effectiveness in real-time proactive rerouting. To solve these limitations, this study presents the OAMLCFAPRA aimed at enhancing SDN effectiveness by dynamically classifying, prioritizing, and rerouting flows while also minimizing real-time conflicts.

3. PROPOSED METHODOLOGY

This section describes the OAMLCFAPRA in SDN for flow classification and rerouting. Figure 1 portrays the architecture of this proposed study. This proposed study applies the ICWL to classify normal and conflicting flows and allocate flow priorities according to the flow types. Then, it determines the betweenness of all routes in the network and ranks routes in decreasing order of betweenness. Each route from the betweenness set is examined individually, and the traffic load is assessed. If the load exceeds 50% of the link capacity, packet priority is evaluated. Flows with packets of high priority are instantly rerouted to avoid congestion and packet loss.

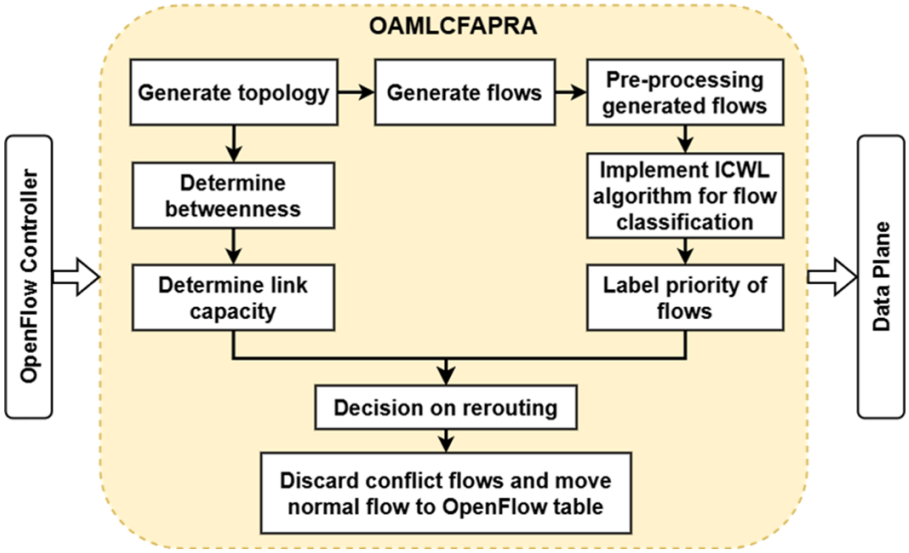

Figure 1. Architecture of the Proposed Study

3.1. Problem Formulation

Assume a network $G = (V, E)$, where V and E are nodes and links or edges, respectively. Here, $E = \{E_i\}, 1 \leq i \leq e$, where E_i is the i^{th} link and e represents a sum of links. Consider $C = \{C_i\}, 1 \leq i \leq e$ is the set of capacities of each E, where C_i is the i^{th} link capacity. F denotes the set of flows with a source and destination, represented as $F = \{F_j\}, 1 \leq j$, where F_j is the j^{th} flow. T is the set of routes, where every route in T refers to a sequence of links joining

a source and destination nodes, denoted as $T = \{T_k\}, 1 \leq k \leq p$, where T_k is the k^{th} route and p indicates a sum of routes. Every route T_k in T holds a betweenness score $B(T_k)$, indicating how frequently the route is used in the network.

The objective is to maximize network throughput by assigning priorities and paths to each flow in F. The optimization problem is defined as $\max(number\ of\ flows\ routed)$:

1. Flow priority: Flows with high priority are rerouted immediately when a route's load capacity is greater than 50%. Low-priority flows are positioned in a ranked list for later rerouting.
2. Capacity: A sum of traffic on E_i in E should not be greater than its C_i.
3. Routing criteria: F_j in F should be directed between its source and destination via a route in T.
4. Betweenness: Routes in the betweenness set are prioritized for routing based on their betweenness score in decreasing order.

As a result, increasing the network's total flow is the goal of the objective function. This is subject to constraints that prioritize and route flows effectively, while also ensuring that the network is capacity and routing criteria are maintained.

3.2. Flow Generation and Pre-processing

This study developed an SDN dataset by generating and collecting normal and conflicting flows from the OpenFlow table. The procedure involves three stages, as illustrated in Figure 2: generating normal flows from the running topology, creating and implementing conflict rules in the OpenFlow table, and generating conflicting flows.

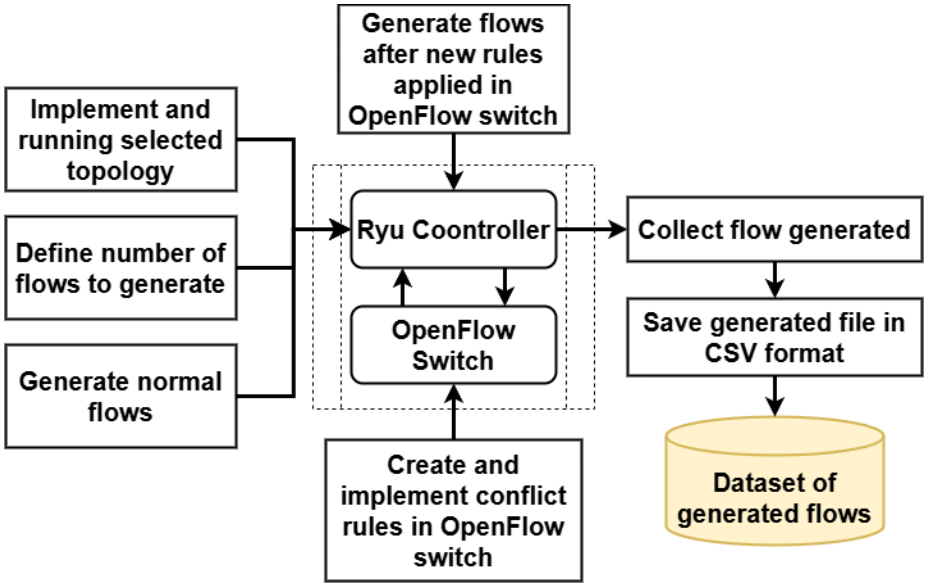

Figure 2. Generation of Normal and Conflicting Flows

Flows are considered conflicting if they conflict with the flow instructions in the switch. Repetition, overlap, shadowing, correlation A, correlation B, generalization, and imbrication are the seven types of competing flows found in this dataset. The pre-processing is used to create flows from the OpenFlow switch for classification. Relevant characteristics like action, protocol,

MAC address, and IP address are extracted to train the ICWL. Conflicts can be classified based on these characteristics. Pre-processing deletes missing data points where missed attributes are more than three otherwise applying mean imputation methods to handle them. Remove data duplicates. The outliers are distorted by the Z-score normalization model.

In this study, Fat Tree Topology (FAT) is simulated for an SDN network. The experiment employed the Ryu controller to connect with an OpenFlow switch version 1.3 for data evaluation in both topologies. The topologies were set up in Mininet and linked to the Ryu for traffic generation. The FAT consisted of 7 switches and 8 hosts, whereas the STT comprised 3 switches and 4 hosts. The Ryu was connected to every switch and host using the Topo.py Python app. Traffic generation involved creating 10000 flows, with each host initiating 10 iperf servers on different ports (8089, 8082, 8081). A basic switch was required for flow entry creation, with the L4 Match app serving as the foundation for this setup.

The controller used source/destination IP, source/destination port, and protocols to create various flows. Each packet received by the controller triggered the creation of a fresh flow in the switch. After setting up the network topologies with the Topo app, a specific quantity of flows was chosen, and the Ryu supervisor app was launched to create normal flows. Once the desired quantity of flows was established, the conflicts flow app was executed to introduce conflicting rules in Ryu. After generating normal and conflict flows, the flowstat app was used to gather and store all the flows in a CSV document.

3.3. Online Active Learning Algorithm for Classification and Priority Assignment of Flows

The CSV file contains instances or observations (x_i, y_i), where x_i is the characteristics (e.g., IP address, MAC address, protocol, priority, and action) and y_i represents the label (flow types).

3.3.1. Online Learning

The problem of flow classification is formulated as an online classification and it is executed in a sequence of successive rounds. At round t, the classifier initially gets an instance $x_i \in X$. The classifier predicts the label $\hat{y}_i$ based on the prediction function $h(x_i)$ and calculates the loss function $L(y_i, \hat{y}_i)$. Afterward, the classifier updates the prediction rule for consecutive rounds utilizing the present instance (x_i, y_i). The aim is to decrease the loss rate, which is crucial in classification tasks within the learning approach.

Initially, the prediction phase of online learning is described. At round t, an online algorithm uses a binary discriminant based on a linear hypothesis having an inner product as:

$$f_t(x) = \phi(x)^\top w_t \tag{1}$$

In Eq. (1), $w_t \in \mathbb{R}^d$ is the weight vector, and $\phi: X \to \mathbb{R}^d$ serves as feature extraction that maps instances into the desired feature space. $\mathrm{sign}(\phi(x_t)^\top w_t)$ forecasts the flow label of the feature x_i and $\phi(x)^\top w_t = 0$ denotes a rectilinear resolution border for features space. The magnitude $|\phi(x)^\top w_t|$ is denoted as the confidence of this prediction. According to this hypothesis, a Linearized Confidence-Weighted (LCW) algorithm is adopted, which applies a Gaussian distribution over the weight vector as:

$$w_t \sim \mathcal{N}(\mu_t, \Sigma_t) \tag{2}$$

In Eq. (2), $\mu_t \in \mathbb{R}^d$ is the mean vector and $\Sigma_t \in S_{++}^d$ is the covariance matrix, where S_{++}^d is the set of symmetric positive definite $d \times d$ matrices. Entries of μ_t signify the knowledge of features and diagonal entries of Σ_t refer to the confidence in the weights. The greater the diagonal entry, the less confidence in the associated weight, and vice versa. Off-diagonal entries record information about feature interactions.

Using the Gaussian distribution, the LCW algorithm predict the class label of x_i based on the mean value of $\phi(x)^\top w_t$ as follows:

$$\hat{y}_t = \text{sign}(\mathbb{E}[\phi_t^\top w_t]) = \text{sign}(\phi_t^\top \mu_t) \tag{3}$$

In Eq. (3), $\phi_t^\top w_t \sim \mathcal{N}(\phi_t^\top \mu_t, \phi_t^\top \Sigma_t \phi_t)$. The variance of $\phi_t^\top w_t$ and $\phi_t^\top \Sigma_t \phi_t$ are utilized to define the confidence in this prediction: a smaller variance leads to higher confidence, and vice versa. After the correct label y_i is obtained, the classification efficiency is assessed by determining the error value as:

$$L(y_i, \hat{y}_i) = \begin{cases} 1, & if\ \hat{y}_i \neq y_t \\ 0, & if\ \hat{y}_i = y_t \end{cases} \tag{4}$$

Before initiating the next round, the LCW algorithm determine the update of the linear hypothesis for consecutive rounds by resolving a constrained optimization dilemma. The new hypothesis is defined as:

$$f_{t+1}(x) = \phi(x)^\top w_{t+1}, w_{t+1} \sim \mathcal{N}(\mu_{t+1}, \Sigma_{t+1}) \tag{5}$$

In Eq. (5), $(\mu_{t+1}, \Sigma_{t+1})$ is the solution of this optimization dilemma. This is the learning phase of online learning.

3.3.2. Linearized Confidence-Weighted (LCW) Algorithm

Since f_t in this algorithm is parameterized by a weight vector w_t following a Gaussian distribution $\mathcal{N}(\mu_t, \Sigma_t)$, it is equal to that f_t is determined by $\mathcal{N}(\mu_t, \Sigma_t)$. It is natural that the updated hypothesis f_{t+1} is needed to be parameterized by a new weight vector w_{t+1} following a new Gaussian distribution $\mathcal{N}(\mu_{t+1}, \Sigma_{t+1})$, or equivalently, that f_{t+1} is determined by $\mathcal{N}(\mu_{t+1}, \Sigma_{t+1})$. Therefore, the update of this algorithm is determined by the solution of the below constrained optimization dilemma containing two Gaussian distributions:

$$\min_{\mu,\Sigma} D_{KL}(\mathcal{N}(\mu, \Sigma) \| \mathcal{N}(\mu_{t+1}, \Sigma_{t+1})) \tag{6}$$
$$\text{s.t. } \mathbb{P}[\gamma_t(\phi_t^\top w) \geq 0] \geq \eta$$

In Eq. (6), $D_{KL}(\cdot \| \cdot)$ is the Kullback-Leibler (KL) divergence between two distributions and $\mathbb{P}[\cdot]$ measures the probability that relies on $w \sim \mathcal{N}(\mu, \Sigma)$. Also, $\eta \in (0.5, 1]$ is a predefined probability threshold. This optimization issue seeks to identify the new Gaussian distribution with the least amount of distributional change while maintaining a sufficient probability of a right prediction on the given case, since the KL divergence may be seen as a measure of the difference between two distributions. So, Eq. (6) can be rewritten as follows:

$$\min_{\mu,\Sigma} \frac{1}{2}\log\left(\frac{\det \Sigma_t}{\det \Sigma}\right) + \frac{1}{2} tr(\Sigma_t^{-1}\Sigma) + \frac{1}{2}(\mu_t - \mu)^\top \Sigma_t^{-1}(\mu_t - \mu) \quad (7)$$
s.t. $\gamma_t(\phi_t^\top \mu) \geq \psi(\phi_t^\top \Sigma_t \phi_t)$

By utilizing the variance $\phi_t^\top \Sigma_t \phi_t$ of $\phi_t^\top w_t$, it seeks to identify the new Gaussian distribution with the least amount of change to the distribution while maintaining a sufficiently high margin. The constraint of Eq. (7) is rewritten as the following probability constraint:

$$\mathbb{P}[\gamma_t(\phi_t^\top w) \geq 0] \geq \Psi\left(\Psi^{-1}(\eta)\sqrt{\phi_t^\top \Sigma_t \phi_t}\right) \quad (8)$$

In contrast to η, the updated formula of the confidence parameter is represented as $\eta' = \Psi\left(\Psi^{-1}(\eta)\sqrt{\phi_t^\top \Sigma_t \phi_t}\right)$. If the standard deviation is more than 1 or less than 1, the LCW algorithm will either amplify or reduce the associated variables. At last, a closed-form solution exists for Eq. (6) and determines the update rule for this LCW algorithm:

$$\mu_{t+1} = \mu_t + \tau_t \gamma_t \Sigma_t \phi_t \quad (9a)$$
$$\Sigma_{t+1} = \Sigma_t - \lambda_t \Sigma_t \phi_t \phi_t^\top \Sigma_t \quad (9b)$$
Where $u^t = \gamma_t(\phi_t^\top \mu_t), v^t = \phi_t^\top \Sigma_t \phi_t$ (9c)
$$\tau_t = \max\left\{0, \frac{-(1+2\psi u^t) + \sqrt{(1+2\psi u^t)^2 - 8\psi(u^t - \psi v^t)}}{4\psi v^t}\right\} \quad (9d)$$
$$\lambda_t = \frac{2\tau_t \psi}{1 + 2\tau_t \psi v^t} \quad (9e)$$

Observe that $\tau_t = 0$ if and only if (μ_t, Σ_t) satisfies the constraint of Eq. (7). Eq. (9a) indicates that weights with low confidence require a more aggressive update to their mean value, while Eq. (9b) suggests that weights with higher confidence have less aggressive updates to their variance.

3.3.3. Iterative Linearized Confidence-Weighted Algorithm

LCW's aggressive weight update policy can lead to instability when there is low confidence in some features, and thus large variations in weight values. When feature distributions change over time in dynamic contexts, this can lead to overfitting and poor generalization. In addition, classification performance could suffer if weight adjustments are made with a single update per instance, which is not ideal.

In this study, a customized weight-updating scheme is introduced with the LCW, which involves updating the weights for each instance multiple times rather than a single update, ensuring the optimal weight value. This approach is simple and effective in minimizing loss. The weights are trained optimally by iterating through each instance individually. The results show a lower mistake rate at $m = 2$ and a consistent mistake rate from $m = 8$ onwards. The weight updating process improves the mistake rate starting from $m = 2$, with some datasets achieving $m = 0$. This customized weight updating scheme does not involve any changes to conventional approaches, except for the introduction of a loop. There are no modifications to the feature vector or projected label in all iterations. The weights are modified in all iterations, leading to improved performance.

Consider w_i and w_i^k are the update for i^{th} instance before and after k iterations, respectively. Δw_i and Δw_i^k are the update rule value for i^{th} instance before and after k iteration. Generally, a weight update is as follows:

$$w_i = w_{i-1} + \Delta w_i \tag{10}$$

According to this, the weight update for k^{th} iteration of i^{th} instance is as follows:

$$w_i^k = w_{i-1}^k + \Delta w_i^k \tag{11}$$

Consider w_i^* is the optimal weight at w_i^k; so,

$$\begin{aligned} w_i^* &= w_i^{k-1} + \Delta w_i^k \\ &= w_i^{k-2} + w_i^{k-1} + \Delta w_i^k \\ &= w_i^0 + \Delta w_i^1 + \Delta w_i^2 + \cdots + \Delta w_i^k \end{aligned} \tag{12}$$

$$w_i^* - w_i^0 = \Delta w_i^1 + \Delta w_i^2 + \cdots + \Delta w_i^k \tag{13}$$

Simplify Eq. (13) by taking the norm and squaring both sides as follows:

$$\left\| w_i^* - w_i^0 \right\|^2 = \left\| \Delta w_i^1 + \Delta w_i^2 + \cdots + \Delta w_i^k \right\|^2 \tag{14}$$

$$\left\| w_i^* - w_i^0 \right\|^2 = \left\| \Delta w_i^1 \times 1 + \Delta w_i^2 \times 1 + \cdots + \Delta w_i^k \times 1 \right\|^2 \tag{15}$$

By applying the Cauchy-Schwarz inequality, as given in Eq. (16), to simplify Eq. (15) as:

$$\left(\sum_{i=1}^n a_i b_i\right)^2 \leq \left(\sum_{i=1}^n a_i^2\right)\left(\sum_{i=1}^n b_i^2\right) \tag{16}$$

$$\begin{aligned} \left\| w_i^* - w_i^0 \right\|^2 &\leq \left(\left\| \Delta w_i^1 \right\|^2 + \left\| \Delta w_i^2 \right\|^2 + \cdots + \left\| \Delta w_i^k \right\|^2 \right)\left(1^2 + 1^2 + \cdots + 1^2\right) \\ &\leq \left(\sum_{j=1}^K \|\Delta w_i\|^2 \right) \times K \end{aligned} \tag{17}$$

$$\text{If } M \geq K, \leq M\left(\sum_{j=1}^M \|\Delta w_i\|^2\right) - \left\| w_i^0 \right\| \leq \|w_i^*\| - \left\| w_i^0 \right\| \leq \sqrt{M}\left(\sum_{j=1}^M \left\| \Delta w_i^j \right\|^2\right)^{1/2} \tag{18}$$

Adding $\left\| w_i^0 \right\|$ in Eq. (18) to obtain

$$0 \leq \|w_i^*\| \leq \left\| w_i^0 \right\| + \sqrt{M}\left(\sum_{j=1}^M \left\| \Delta w_i^j \right\|^2\right)^{1/2} \tag{19}$$

The ideal weight w_i^* for x_i is determined by several iterations and is constrained by Eq. (19). A pseudocode for this weight updating mechanism is described in Algorithm 1.The algorithm clearly mentioned how the proposed for LCW update the weight of each features dynamically. Based on the weight updating the flow is prioritized dynamically which improve bandwidth utilization, reduce delay and increase the throughput. Overall, the network performance is improved.

Algorithm 1: Proposed Weight Updating Mechanism for LCW

1. **Begin**
2. Initialize $w_1 = 0$;
3. $\boldsymbol{for}(i = 1{:}n)$ //n is the sum of instances in the dataset
4. $\boldsymbol{for}(k = 1{:}m)$ //$m = 1{:}32$ in this study
5. Predict $\hat{y}_i = \langle w_i, x_i \rangle$;
6. Calculate loss as $L(y_i, \hat{y}_i)$;
7. $\boldsymbol{if}(L(y_i, \hat{y}_i) > 0)$
8. $w_{i+1} = w_i + \langle update\ rule \rangle$; //Update rule depends on the LCW
9. $\boldsymbol{end\ if}$
10. $\boldsymbol{end\ for}$
11. $\boldsymbol{end\ for}$
12. **End**

Thus, this ICWL with a customized weight updating scheme is used to classify flows as normal or conflicting. The flows are then prioritized based on features like Flow ID, source, source port, destination IP address, destination port, and protocol (TCP layer protocol number). Normal flows are sent to OpenFlow while conflicting flows are further classified into seven categories according to the priority and IP address features. The classification and prioritization of flows is portrayed in Figure 3.

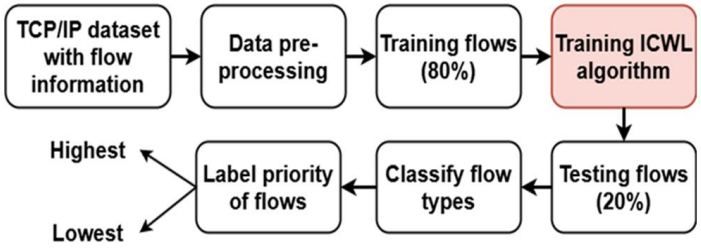

Figure 3. Processes of Classification and Priority Assignment of Flows Using ICWL

3.4. Rerouting for Congestion Avoidance

One important metric for determining the significance of nodes is centrality in G. Edge betweenness centrality quantifies the sum of shortest routes that traverse through a specific e in G. Each node is assigned an edge betweenness centrality value. A maximum value suggests that the link plays a crucial role in connecting nodes, and removing it could disrupt communication in the network. The between centrality $B(x)$ of an edge e in G is defined by

$$(x) = \sum_{a,b \in V} \frac{\sigma(a, b|e)}{\sigma(a,b)} \quad (20)$$

In Eq. (20), $\sigma(a, b|e)$ is the sum of shortest paths between a and b that pass through e, while $\sigma(a, b)$ is the sum of shortest routes between a and b. Thus, $B(x)$ for each route in G is determined and the routes are positioned in the descending value of $B(x)$. For data transfer, routes are considered one by one from $B(x)$ and the load on each route is checked. When the load on a route is greater than 50% of the link capacity, the highest priority flow is rerouted immediately. To reroute the highest priority flows, the source and destination are identified. The

top-K routes, where $K = \frac{N}{2}$ and N denotes the sum of routes from the source and destination nodes, are considered in increasing value of hop count. The route with the least traffic is selected for flow rerouting.

Low-priority flows are listed in a ranked order. Each route is evaluated based on traffic volume on the link and rerouted accordingly. The most cost-effective route is selected, and static flows are pushed to the Ryu via the REST API after communication with the load-balancing app. These flows are refreshed every minute to ensure dynamic load balancing. Figure 4 presents a flowchart of OAMLCFAPRA, while Algorithm 2 outlines its entire process.

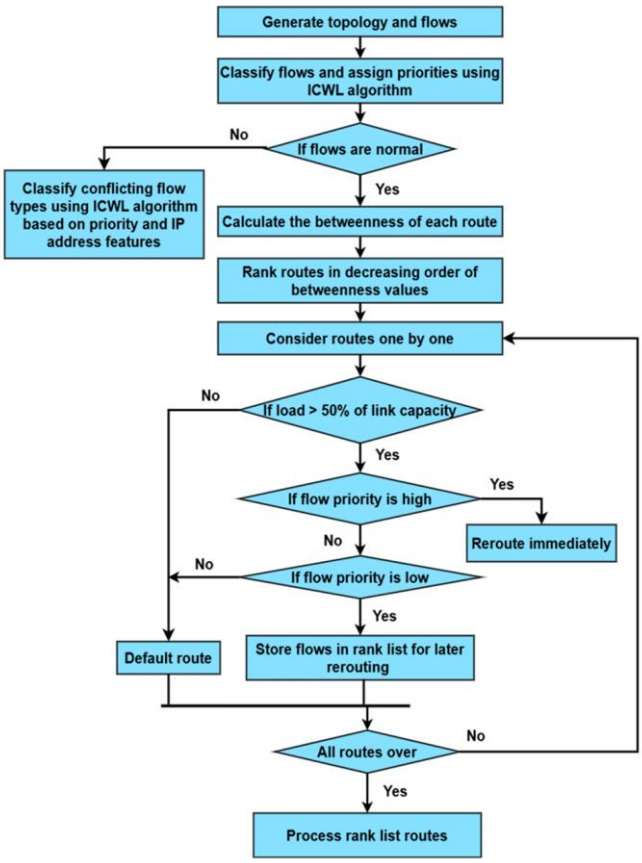

Figure 4. Flowchart of OAMLCFAPRA for Rerouting in SDN

Algorithm 2: OAMLCFAPRA for Rerouting

1. **Begin**
2. Construct and simulate the topology;
3. Obtain the topology data from the Ryu controller to build G;
4. Classify the flow types using ***Algorithm 1***;
5. Label the priority of flows;
6. Classify the conflict flow types using ***Algorithm 1***;
7. Pass the normal flows into the OpenFlow;
8. $\boldsymbol{for}(links\ or\ routes\ in\ G)$
9. Compute the betweenness of all links;
10. Store the betweenness value in B;
11. ***end for***

12. Rank the betweenness value B in decreasing order;
13. $\boldsymbol{for}(each\ link\ or\ route\ with\ betweenness\ value)$
14. $\boldsymbol{if}(load > 50\%\ of\ link\ capacity)$
15. $\boldsymbol{if}(flows\ with\ highest\ priority)$
16. Reroute the flow immediately via top-K routes with lowest traffic;
17. $\boldsymbol{else}$
18. Insert flow in the rank list for later rerouting;
19. $\boldsymbol{end\ if}$
20. $\boldsymbol{end\ if}$
21. $\boldsymbol{end\ for}$
22. **End**

4. SIMULATION RESULTS

On a Windows 10 machine with 256 GB of RAM and an Intel Core i7 CPU, the OAMLCFAPRA has been implemented as a Python module. Table 1 lists the prerequisites for running the simulation. The simulation parameters are listed in Table 2. The specifications are listed here after applying simulation for the parameters and obtaining best results among the simulations. The results for different inputs are measured through the performance metrics. These parameters are considered for FAT topology. The descriptions of topology is briefly given in section 3.2.

Table 1. Simulation Requirements

Hardware/Software	Framework	Specifications
Hardware	CPU	Intel Core i7 – 2.4 GHz
	RAM	12 GB
	Hard disk	2 TB
Software	OS	Windows 10-64 bit
	Programming language	Python 2.7
	Programming IDE	Spyder 3.3.3
	Machine learning software	Tensorflow/Python
	Machine learning library	Scikit-learn v0.21.3
	Controller	Pox 0.2.0

Table 2. Simulation Parameters

Parameters	Value
No. of nodes	15
No. of links	19
Topology	FAT
No. of flow requests	40
Maximum bandwidth	100 Mbps
Minimum bandwidth	10 Mbps
Minimum and maximum delay	10 and 100 ms
Link jitter	10-20 ms
Simulation time	600 sec
Packet category	TCP
Packet size	64 bytes
No. of packets transferred	250
Packets interval	20 sec
Link PLR	1-7%

4.1. Performance Evaluation Metrics

The performance of classifying conflicting flows is measured based on the following metrics:

- Accuracy: It assesses a model's capability to classify conflict flows among a total number of generated flows.

$$Accuracy = \frac{True\ Positive\ (TP)+True\ Negative\ (TN)}{TP+TN+False\ Positive\ (FP)+False\ Negative\ (FN)} \quad (21)$$

Here, TP indicates the amount of conflicting flows exactly categorized, TN indicates the amount of exactly classified normal flows, FP represents the amount of normal flows inexactly classified as conflicting flows, and FN refers to the amount of conflicting flows inexactly classified as normal flows.

- Precision: It is determined by

$$Precision = \frac{TP}{TP+FP} \quad (22)$$

- Recall: It is computed as:

$$Recall = \frac{TP}{TP+FN} \quad (23)$$

- F1-score: It is determined as:

$$F1 = \frac{2\times Precision\times Recall}{Precision+Recall} \quad (24)$$

The performance of rerouting normal flows to prevent congestion is measured using below measures:

- Throughput: It refers to the overall quantity of packets transferred from the source to destination per unit time.
- Bandwidth utilization: It is the percentage of utilized bandwidth to total available bandwidth.
- Round Trip Delay (RTT): It is the total interval spent to transfer a packet between the source and destination nodes and vice versa.
- Delay: Duration of a packet's journey from its origin node to its final destination node. It involves the propagation, processing, queuing, and transmission delays.
- Jitter: It is the variance of delay.
- Packet Loss Rate (PLR): It is the ratio of packets lost in unit interval.

4.2. Performance Analysis of Flow Classification

Figure 5 illustrates the performance analysis of flow classification using the proposed ICWL and existing DT [10], SVM [12], and EFDT [13]. The precision of ICWL is increased by 8.1%, 5.75%, and 3.19% compared to DT, SVM, and EFDT, respectively. The recall is 7.81%, 5.83%, and 3.12% higher than the same algorithms, respectively. The F1-score is increased by 7.96%, 5.79%, and 3.16% compared to the other algorithms, respectively. The accuracy is 7.95%, 5.76%, and 3.2% greater than the same algorithms, respectively. This improvement is achieved because ICWL continuously updates its model based on real-time network traffic, unlike DT, SVM, and EFDT algorithms, which rely on static training datasets. In addition, ICWL updates weights multiple times per instance, unlike existing algorithms that update only once.

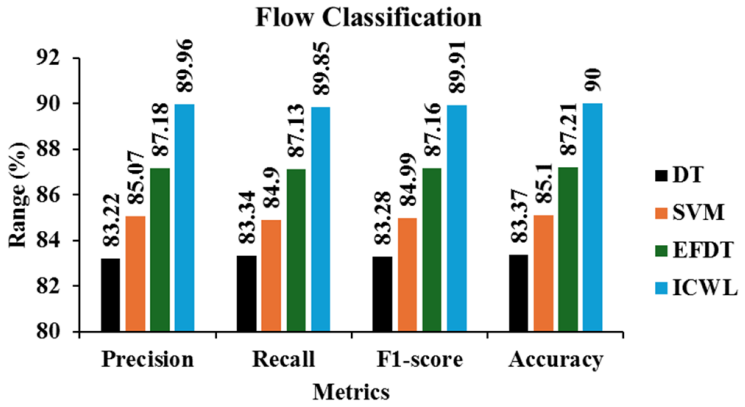

Figure 5. Performance Analysis of Different Algorithms for Flow Classification

Table 3 presents the flow classification results for DT, SVM, EFDT, and the proposed ICWL. The ICWL outperformed the other algorithms due to its enhanced performance. Therefore, ICWL was selected for classifying conflict flow types. Flows within the range of 10,000 to 100,000 were chosen with a multiplier of 10,000 flows. The proposed ICWL achieved the best performance in categorizing conflict flow types across different flow numbers. The performance metric for each interval is measured and tabulated for clear understanding of both the proposed and existing algorithms. For all intervals, proposed work always outperform then all other methods.

Table 3. Flow Classification Results of Different Algorithms

Dataset	Accuracy (%)	Precision (%)	Recall (%)	F1-score (%)
DT				
10000	83.37	83.32	83.34	83.28
20000	82.91	82.82	82.90	82.86
30000	82.05	81.97	82.04	82.00
40000	81.53	81.48	81.52	81.50
50000	81.01	80.90	80.99	80.94
60000	80.62	80.55	80.60	80.57
70000	80.10	80.02	80.09	80.05
80000	79.35	79.27	79.33	79.30
90000	78.80	78.69	78.78	78.73
100000	78.09	77.95	78.05	78.00
SVM				
10000	85.10	85.07	84.90	84.99
20000	84.61	84.56	84.60	84.58
30000	84.03	83.95	84.02	83.98
40000	83.52	83.41	83.50	83.45
50000	82.94	82.87	82.93	82.90
60000	82.30	82.24	82.30	82.27
70000	81.86	81.78	81.85	81.81
80000	81.11	81.05	81.10	81.07
90000	80.63	80.59	80.64	80.61
100000	80.27	80.20	80.27	80.23
EFDT				
10000	87.21	87.18	87.13	87.16
20000	86.60	86.51	86.59	86.55
30000	85.98	85.86	85.97	85.91

Dataset	Accuracy (%)	Precision (%)	Recall (%)	F1-score (%)
EFDT				
40000	85.17	85.09	85.17	85.13
50000	84.55	84.49	84.56	84.52
60000	84.00	83.92	83.99	83.95
70000	83.41	83.35	83.40	83.37
80000	82.84	82.73	82.82	82.77
90000	82.33	82.27	82.34	82.30
100000	81.85	81.78	81.85	81.81
ICWL				
10000	90.00	89.96	89.85	89.91
20000	89.12	89.04	89.11	89.07
30000	88.55	88.38	88.52	88.45
40000	87.89	87.71	87.85	87.78
50000	87.01	86.89	87.00	86.94
60000	86.54	86.43	86.51	86.47
70000	85.97	85.90	85.94	85.92
80000	85.23	85.12	85.21	85.26
90000	84.70	84.59	84.66	84.62
100000	84.06	83.94	84.03	83.98

4.3. Performance Analysis of Flow Rerouting

To evaluate the efficiency of flow rerouting for congestion avoidance, the OAMLCFAPRA is compared to the MLPRS [7] and default SDN in terms of network metrics. Here, default SDN stands for SDN in its standard configuration, which does not include advanced traffic handling or ML optimization.

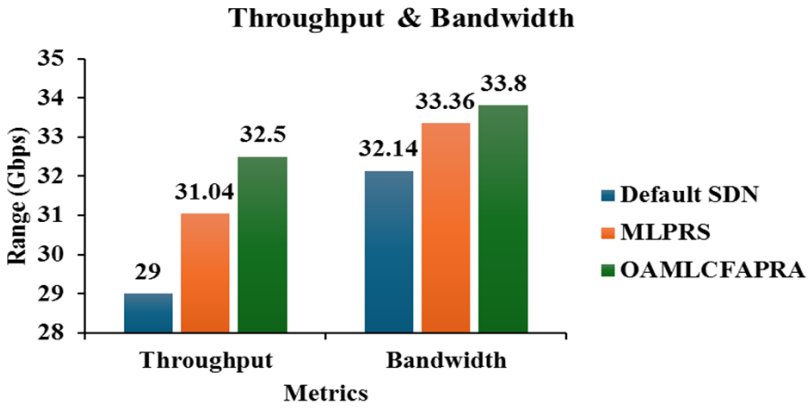

Figure 6. Throughput and Bandwidth Analysis of OAMLCFAPRA against MLPRS and Default SDN

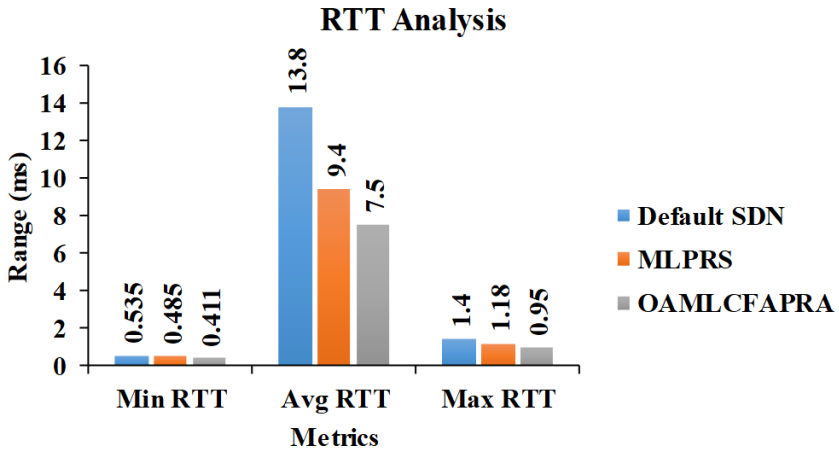

Figure 7. RTT Analysis of OAMLCFAPRA against MLPRS and Default SDN

In this setup, a centralized controller manages network flows using static or traditional routing methods without proactive re-routing based on dynamic traffic patterns. Figure 6 compares the throughput and bandwidth between OAMLCFAPRA, MLPRS, and default SDN. OAMLCFAPRA achieves a throughput of 32.5 Gbps, which is 12.07% and 4.7% higher than the default method and MLPRS, respectively. Additionally, the bandwidth of OAMLCFAPRA is 33.8 Gbps, representing a 5.16% and 1.32% improvement over the default SDN and MLPRS. This is due to prioritizing links with decreasing betweenness values for packet re-routing rather than packet routing in random manner.

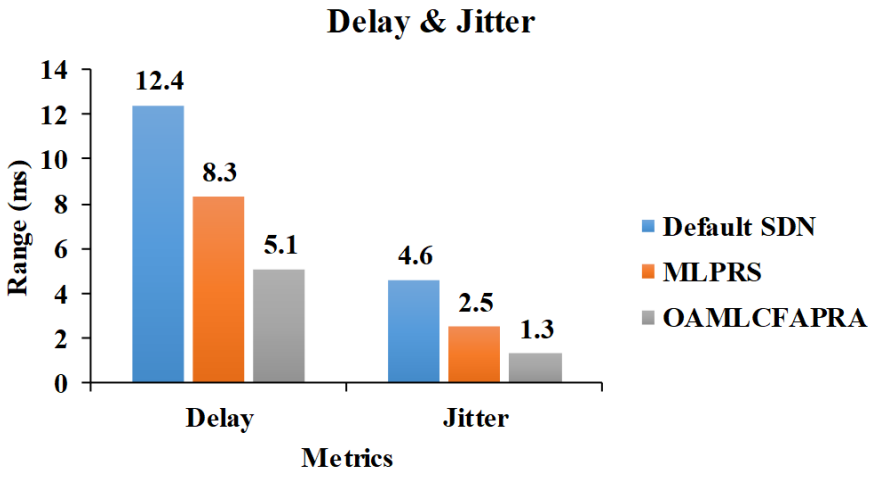

Figure 8. Delay and Jitter Analysis of OAMLCFAPRA against MLPRS and Default SDN

Figure 7 compares the RTT of OAMLCFAPRA, MLPRS, and default SDN while packets are rerouted in descending betweenness score. The minimum RTT of OAMLCFAPRA is 23.18% and 15.26% lower than default SDN and MLPRS algorithm, respectively. The average RTT of OAMLCFAPRA is 45.65% and 20.21% lower than the same algorithms. The maximum RTT of OAMLCFAPRA is 32.14% and 19.49% lower than the same algorithms. This shows that OAMLCFAPRA significantly reduces minimum, maximum, and average RTT compared to efault SDN and MLPRS.

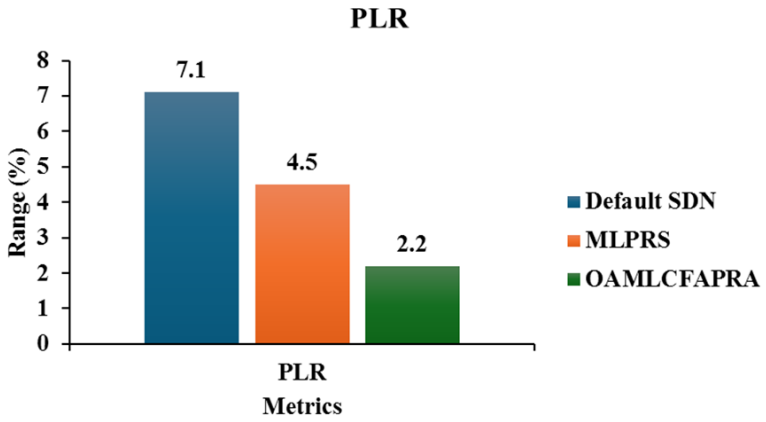

Figure 9. PLR Analysis of OAMLCFAPRA against MLPRS and Default SDN

Figure 8 compares the delay and jitter between OAMLCFAPRA, MLPRS, and default SDN. OAMLCFAPRA achieves a delay of 5.1 ms, which is 58.87% and 38.55% lower than the default SDN and MLPRS, respectively. The jitter of OAMLCFAPRA is 1.3 ms, representing a 71.74% and 48% reduction over the default SDN and MLPRS. Additionally, Figure 9 illustrates the PLR for OAMLCFAPRA, MLPRS, and default SDN methods. The OAMLCFAPRA reduces the PLR by 69.01% and 51.11% compared to the MLPRS and default SDN methods. This is achieved by prioritizing links for packet re-routing rather than packet routing in weight updating method.

5. CONCLUSIONS

This paper introduced the OAMLCFAPRA, a model for classifying, prioritizing, and rerouting traffic flows in SDN based on their behavior characteristics. It involved generating and pre-processing traffic flows to extract features, classifying them as normal or conflicting using ICWL with a customized weight updating scheme, and prioritizing them into the highest and lowest categories. Conflicting flows were further categorized into seven types based on priority and IP address features, whereas normal flows were passed to OpenFlow. Furthermore, it determined the significance of each path using betweenness centrality and monitored load capacity. In case of overload, the highest priority flow was rerouted immediately, while lower priority flows were stored for later rerouting. This approach helps prevent congestion, leading to improved throughput and bandwidth. Experimental results proved that the ICWL achieved 90% accuracy, 89.96% precision, 89.85% recall, and 89.91% F1-score for flow classification compared to the DT, SVM, and EFDT. Additionally, the OAMLCFAPRA achieved 32.5 Gbps throughput, 33.8 Gbps bandwidth, 0.411ms minimum RTT, 7.5ms average RTT, and 0.95ms maximum RTT compared to the default SDN and MLPRS algorithm.

CONFLICTS OF INTEREST

The authors declare no conflict of interest

REFERENCES

[1] M.A. Al-Shareeda, A.A. Alsadhan, H.H. Qasim, and S. Manickam, “Software defined networking for internet of things: review, techniques, challenges, and future directions,” *Bulletin of Electrical Engineering and Informatics*, Vol.13, No.1, pp.638-647, 2024, https://doi.org/10.11591/eei.v13i1.6386

[2] S. Maulana, S.A. Anjani, Y.P.A. Sanjaya, and P. Sithole, "Software-defined networking: Revolutionizing network management and optimization," *Journal of Computer Science and Technology Application*, Vol.1, No.2, pp.164-171, 2024, https://doi.org/10.33050/g1as4162

[3] R. Chaudhary, G.S. Aujla, N. Kumar, and P.K. Chouhan, "A comprehensive survey on software-defined networking for smart communities," *International Journal of Communication Systems*, Vol.38, No.1, p. e5296, 2025, https://doi.org/10.1002/dac.5296

[4] A. Mwangi, R. Sahay, E. Fumagalli, M. Gryning, and M. Gibescu, "Towards a software-defined industrial IoT-edge network for next-generation offshore wind farms: State of the art, resilience, and self-X network and service management," *Energies*, Vol.17, No.12, pp.2897, 2024, https://doi.org/10.3390/en17122897

[5] U.A. Bukar, and M. Othman, "Architectural design, improvement, and challenges of distributed software-defined wireless sensor networks," *Wireless Personal Communications*, Vol.122, No.3, pp.2395-2439, 2022, https://doi.org/10.1007/s11277-021-09000-2

[6] N. Gupta, M.S. Maashi, S. Tanwar, S.Badotra, M. Aljebreen, and S. Bharany, "A comparative study of software defined networking controllers using mininet," *Electronics*, Vol.11, No.17, p.2715, 2022, https://doi.org/10.3390/electronics11172715

[7] D. Kafetzis, S. Vassilaras, G. Vardoulias, and I. Koutsopoulos, "Software-defined networking meets software-defined radio in mobile ad hoc networks: state of the art and future directions," *IEEE Access*, Viol.10, pp. 9989-10014, 2022, https://doi.org/10.1109/ACCESS.2022.3144072

[8] D. Carrascal, E. Rojas, J.M. Arco, D. Lopez-Pajares, J. Alvarez-Horcajo, and J.A. Carral, "A comprehensive survey of in-band control in sdn: Challenges and opportunities," *Electronics*, Vol.12, No.6, p.1265, 2023, https://doi.org/10.3390/electronics12061265

[9] C. Ezechi, M.O. Akinsolu, A.O. Sangodoyin, F.T. Akinsolu, and W. Sakpere, "Software-defined networking in cyber-physical systems: benefits, challenges, and opportunities," *Cyber Physical System 2.0*, 44-69, 2025.

[10] P.A. Baziana, "Optical Data Center Networking: A Comprehensive Review on Traffic, Switching, Bandwidth Allocation, and Challenges," *IEEE Access*, Vol.12, p.186413-186444, 2024, https://doi.org/10.1109/ACCESS.2024.3513214

[11] J. Cunha, P. Ferreira, E.M. Castro, P.C. Oliveira, M.J. Nicolau, I. Núñez, and C. Serôdio, "Enhancing network slicing security: machine learning, software-defined networking, and network functions virtualization-driven strategies," *Future Internet*, Vol.16, No.7, p.226, 2024, https://doi.org/10.3390/fi16070226

[12] G. Kirubasri, S. Sankar, D. Pandey, B.K. Pandey, V.K. Nassa, and P. Dadheech, "Software-defined networking-based ad hoc networks routing protocols," *In Software defined networking for Ad Hoc networks*, pp. 95-123, 2022, https://doi.org/10.1007/978-3-030-91149-2_5

[13] A.M. Abdulghani, A. Abdullah, A.R. Rahiman, N. A. W. A.Hamid, B.O. Akram, H. Raissouli, "Navigating the Complexities of Controller Placement in SD-WANs: A Multi-Objective Perspective on Current Trends and Future Challenges," *Computer Systems Science & Engineering*, Vol.49, No.1,pp. 123-157, 2025, https://doi.org/10.32604/csse.2024.058314

[14] J.A. Rahim, R. Nordin, and O.A. Amodu, "Open-Source Software Defined Networking Controllers: State-of-the-Art, Challenges and Solutions for Future Network Providers," *Computers, Materials & Continua*, Vol.80, No.1, ,pp.1-10, 2024, https://doi.org/10.32604/cmc.2024.047009

[15] Fogli, C. Giannelli, and C. Stefanelli, "Software-Defined Networking in wireless ad hoc scenarios: Objectives and control architectures," *Journal of Network and Computer Applications*, Vol.203, p.103387, 2022, https://doi.org/10.1016/j.jnca.2022.103387.

[16] M.D. Tache, O. Păscuțoiu, and E. Borcoci, "Optimization algorithms in SDN: Routing, load balancing, and delay optimization," *Applied Sciences*, Vol.14, No.14, p.5967, 2024, https://doi.org/10.3390/app14145967

[17] M.A. Gunavathie, and S. Umamaheswari, "MLPRS: a machine learning-based proactive re-routing scheme for flow classification and priority assignment," *Journal of Engineering Research*, Vol.11, No.3, pp.114-122, 2023, https://doi.org/10.1016/j.jer.2023.100075

[18] A.O. Salau, and M.M. Beyene, "Software defined networking based network traffic classification using machine learning techniques," *Scientific Reports*, Vol.14, No.1, p.20060, 2024, https://doi.org/10.1038/s41598-024-70983-6

[19] F.A. Yaseen, N.A. Alkhalidi, and H.S. Al-Raweshidy, "She networks: Security, health, and emergency networks traffic priority management based on ml and sdnM," *IEEE Access*, Vol.10, pp.92249-92258, 2022, https://doi.org/10.1109/ACCESS.2022.3203070

[20] S. Liang, and J. Su, "Detection of SDN flow rule conflicts based on knowledge graph," *In International Conference on Emerging Networking Architecture and Technologies*, pp. 93-104, 2022, https://doi.org/10.1007/978-981-19-9697-9_8

[21] A. Aqdus, R. Amin, S. Ramzan, S.S. Alshamrani, A. Alshehri, and E.S.M. El-kenawy, "Detection collision flows in SDN based 5G using machine learning algorithms," *Computers, Materials & Continua*, Vol.75, No.1, pp.1413-1435, 2023, https://doi.org/10.32604/cmc.2023.031719

[22] R. Mohammadi, S. Akleylek, A. Ghaffari, and A. Shirmarz, "Automatic delay-sensitive applications quality of service improvement with deep flows discrimination in software defined networks," *Cluster Computing*, Vol.26, No.1, pp.437-459, 2023, https://doi.org/10.1007/s10586-022-03729-6

[23] B. Ananth, "Hybrid support vector machine for predicting accuracy of conflict flows in software defined networks," *Salud, Ciencia y Tecnología*, Vol.4, pp.797-797, 2024, https://doi.org/10.56294/saludcyt2024797

[24] M.H. Khairi, B.M.A. Abdalla, M.K. Hassan, S.H. Ariffin, and M. Hamdan, "Utilizing extremely fast decision tree (EFDT) algorithm to categorize conflict flow on a software-defined network (SDN) controller. Engineering," *Technology & Applied Science Research*, Vol.14, No.2, pp.13261-13265, 2024, https://doi.org/10.48084/etasr.6793

[25] B. Han, Y. Liu, Y.Zhou, and Y. Gao, "An efficient flow rule conflict comprehensive detection scheme for SDN networks," *In IEEE International Symposium on Parallel and Distributed Processing with Applications*, pp. 1895-1902, 2024, https://doi.org/10.1109/ISPA63168.2024.00258

[26] R.H. Serag, M.S. Abdalzaher, H.A.E.A. Elsayed, M. Sobh, M. Krichen, and M.M. Salim, "Machine-learning-based traffic classification in software-defined networks," *Electronics*, Vol.13, No.6, p.1108, 2024, doi: https://doi.org/10.3390/electronics13061108

[27] I.H. Abdulqadder and I.T. Aziz, "Load balanced attack defense system with lightweight authentication and modified blockchain in SDN for B5G," *International Journal of Computer Networks & Communications*,Vol.13, No.6, p.1108, 2024, https://doi.org/10.5121/ijcnc.2025.17106

AUTHORS

Kalaivani S is an Assistant Professor at Dr. GR Damodaran College of Science and a Ph.D. research scholar at KPR Institute of Engineering and Technology. She holds an MCA and an M.Phil. in Computer Science. Her research focuses on networking and network security. She has published papers in journals and presented at conferences. She is passionate about teaching and mentoring students.

Dr. A. Sumathi serves as the Associate Professor and Head of the Department of Information Technology at KPR College of Arts Science and Research in Coimbatore. She holds M.Sc., M.Phil., and Ph.D. degrees, all completed at Bharathiar University. With over 21 years of teaching experience, Dr. Sumathi has contributed significantly to academia since 2004. Her leadership continues to foster growth and innovation within the department.