Utilizing Extremely Fast Decision Tree (EFDT) Algorithm to Categorize Conflict Flow on a Software-Defined Network (SDN) Controller

Mutaz H. H. Khairi
Future University, Sudan
Taza1040@gmail.com

Bushra Mohammed Ali Abdalla
Faculty of Computer Science and Information Technology, Ibn Sina University, Sudan
bushra0912115@gmail.com (corresponding author)

Mohamed Khalafalla Hassan
University Technology Malaysia, Malaysia | Future University, Sudan
mohamed.khalafala.hassan@gmail.com

Sharifah H. S. Ariffin
University Technology Malaysia, Malaysia
shafizah@utm.my

Mosab Hamdan
Interdisciplinary Research Center for Intelligent Secure Systems, King Fahd University of Petroleum and Minerals, Saudi Arabia
mosab.mohamed@kfupm.edu.sa

Received: 21 December 2023 | Revised: 11 January 2024 and 27 January 2024 | Accepted: 29 January 2024

Licensed under a CC-BY 4.0 license | Copyright (c) by the authors | DOI: https://doi.org/10.48084/etasr.6793

ABSTRACT

Software-Defined Networks (SDNs) provide a contemporary approach to networking technology, offering a versatile and dynamically efficient network architecture for enhanced surveillance and performance. However, SDN architectures may encounter flow conflicts. These conflicts arise when modifications are made to specific flow properties, such as priority, match field, and action. Despite the existence of recommended solutions, the process of resolving conflicts in SDN continues to encounter difficulties. This study proposes an Extremely Fast Decision Tree (EFDT) classification technique to detect and categorize conflicts inside the flow table. The novelty of this method is based on the development of an accurate and effective machine-learning technique implemented on the Ryu controller plane and validated using the Mininet simulator. The effectiveness and efficiency of the proposed method were evaluated using various indicators, demonstrating superior performance in recognizing and categorizing conflict flow types in all flow sizes ranging from 10,000 to 100,000.

Keywords-software-defined network; conflict flow; machine learning; extremely fast decision tree

I. INTRODUCTION

Software-Defined Networks (SDNs) can improve network performance through dynamic and customizable network design [1]. This architecture enables easy modifications using a centralized control console for network engineers and administrators to meet the evolving business needs. It also enables easy modifications using a centralized control console for network engineers and administrators to meet evolving business needs [2]. SDNs integrate network technologies to enhance flexibility and agility by separating control functions from forwarding planes [3]. This split could allow independent configuration of the network control plane, giving system specialists complete control over its operations [4-5]. In addition to its fundamental advantages, an SDN is dynamic, controllable, adaptive, and cost-effective, making it a perfect solution for the ever-expanding size and high-bandwidth applications of the Internet [6]. OpenFlow is a key advancement in SDN. The controller is a vital element in the architecture, enabling the creation of diverse applications through an Application Programming Interface (API). The controller's activities influence the success of an SDN. An OpenFlow switch has multiple flow tables linked to the controller via the OpenFlow protocol to distribute, classify, and assign packets based on flow entries [7-11].

SDN systems can be optimized using Machine Learning (ML) to handle data more effectively when comprehending and extracting patterns from data is difficult [12]. With the increase in available datasets, ML is becoming more common in fields such as medicine and government that require relevant data [13-14]. ML aims to learn from data, and many studies have been conducted to improve such methods [15]. The Transaction Conflict Detection and Resolution (TCDR) framework was created to eliminate flow policy conflicts, which improves controller layer performance while maintaining low cost [16]. In [17], a new technique was proposed to anticipate conflicts while evaluating the effectiveness of an application. The algorithm predicts network failure by analyzing remaining actions, based on the guidelines created by the developer. Conflict prediction identifies unwanted system behavior, and prediction-related capabilities should be incorporated based on the evaluation of the prediction model to ensure practical control applications. In [18], cross-layering between flow components was investigated using OpenFlow table variables, and an accurate approach was recommended by analyzing the entire input table.

This study examines seven types of conflict, including redundancy, shadowing, overlapping, correlation (A and B), generalization, and imbrication using conflict rules derived from relevant studies as a guide [19-21]. Flow conflicts in an OpenFlow switch can be categorized by type, priority, action, protocol, and IP source address. These conflicts can have a significant impact on SDN operations in a variety of scenarios. This study aims to reduce the negative impact of conflicts in the SDN environment. To achieve this, the Extremely Fast Decision Tree (EFDT) algorithm was used, which depends on the decision tree structure. This study introduces several novel principles, including the EFDT algorithm, which identifies flow conflicts in SDN. The algorithm is used with varying amounts of flows to identify and categorize conflicting flows. To the best of our knowledge, this is the first study that applies ML algorithms to classify seven distinct categories of conflict flows.

II. PROBLEM DEFINITION

Multiple conflict types adversely influence efficiency across conventional networks and SDNs [20]. Two primary categories of conflicts are based on their rules and outcomes: Interpretative conflicts (generalization, correlation, imbrication) and intelligible conflicts (redundancy, shadowing, overlapping). This study focuses on discussing the limitations of flow entry in terms of priority, match fields, and action fields. Packet counters and timeout values are not critical in handling flow conflicts. Conflicts in SDN can arise due to changes made to attributes such as action and priority. Conflicts may occur in the controller and flow table depending on changes made to the flow rule policies or entries. It is essential to identify priorities and take action to create SDN rules and flow entries [10, 22]. Traditional networks and SDNs are widely recognized to differ significantly in features, particularly priority and action [21]. Additionally, the flow table can create conflicts in various situations, including inconsistencies in rules caused by the following factors [20]:

- Conflicting behavior may occur when the network management system addresses similar flows in multiple tables.
- VPN programs that modify header information may unintentionally apply flow rules to a particular flow.
- The injection of different subsystems, which used the northbound as primary sources from a controller, may cause the flow rule to exhibit conflicting behavior for identical flows.

III. OBJECTIVES

This study aims to develop a method for recognizing flow conflicts in SDNs. The proposed algorithm will be capable of identifying and categorizing each flow in the OpenFlow switch, which will help reduce conflicts among the numerous flows. The proposed method aims to achieve the following specific goals:

- Creating and integrating the EFDT algorithm into the SDN Ryu controller.
- Utilizing the conflict classification technique to categorize and identify various conflicting flows.

IV. MATERIALS AND METHODS

A. Environment Setup for SDN Platform

The proposed SDN solution was built using Ryu, a network controller, in combination with the Mininet simulator. Mininet can be operated in VirtualBox, which is one of the virtual network simulators. However, it is crucial to consider and prepare for the specific requirements related to the virtual machine, SDN controller, and OpenFlow switch version. Table I provides a list of specifications for the simulation environment.

TABLE I. SYSTEM AND ENVIRONMENT SPECIFICATION

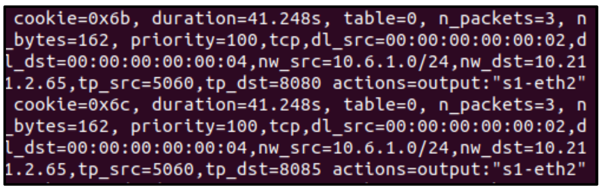
Software and Hardware	Specifications of Machine
Processor	Core i7
RAM	16 GB
Operating system	Ubuntu 18.04
SDN controller	Ryu
Programming language	Python 2.7
OpenFlow Switch	Version 1.3

B. SDN Topology

The fat-tree topology was utilized and integrated into a mininet connected to the Ryu controller. The Topo.py application established the topology of the switches and hosts. Similar topologies have been used in previous studies on SDN conflict flows, including those undertaken to pursue knowledge [18, 20, 22-23].

C. Dataset

The data set used was the one used in [18]. Iperf is a tool that generates and collects OpenFlow data in a Ryu controller topology, including ten performance servers per host. Each server can listen to a different destination port, and new flows are generated based on source and destination IP addresses and source and destination ports and protocols. The controller is responsible for installing a new flow on the switch, updating policy rules, and capturing all flow entries in a CSV file. The analysis proposed a new method to create flows in an SDN controller. This was necessary because there is a lack of SDN datasets that have important flow entry properties such as priority and action features. The algorithm was developed and executed using OpenFlow Switch version 1.3 and the Ryu controller. The dataset used in the study was created by extracting flow entries from the switch OpenFlow table. These flow entries represent the network flows identified and processed by the switch. The study analyzed this dataset to gain insight into network behavior and identify potential performance issues or anomalies. Figure 1 provides a visual representation of the dataset used in the study.



```
 cookie=0x6b, duration=41.248s, table=0, n_packets=3, n
_bytes=162, priority=100,tcp,dl_src=00:00:00:00:00:02,d
l_dst=00:00:00:00:00:04,nw_src=10.6.1.0/24,nw_dst=10.21
1.2.65,tp_src=5060,tp_dst=8080 actions=output:"s1-eth2"
 cookie=0x6c, duration=41.248s, table=0, n_packets=3, n
_bytes=162, priority=100,tcp,dl_src=00:00:00:00:00:02,d
l_dst=00:00:00:00:00:04,nw_src=10.6.1.0/24,nw_dst=10.21
1.2.65,tp_src=5060,tp_dst=8085 actions=output:"s1-eth2"
```

Fig. 1. Flow entries from the OpenFlow table in the switch.

D. Conflict Classification Model

The proposed method involves classifying the conflict categories present in conflict flows across all flow data sizes using the Conflict Classification Model (CCM). The EFDT, combined with the Hoeffding Anytime Tree SEA Generator, is an innovative learning technique that consistently exceeds the traditional decision tree approach. The EFDT has been shown to outperform the Hoeffding Tree version of the Very Fast Decision Tree (VFDT) on multiple standard evaluation workloads, specifically with respect to prequential reliability. This impressive performance highlights the remarkable potential of the EFDT for application in various business and academic settings. The EFDT's ability to produce more reliable outcomes than traditional decision tree methods makes it a compelling option, especially in time-sensitive situations where high accuracy is critical. As such, the EFDT is a valuable addition to the range of data-driven techniques available for decision-making. The CCM comprises four primary activities: compiling conflict flow data, setting up the Ryu controller to use the implemented classification algorithm, learning the EFDT algorithm with various conflicts, and evaluating the algorithm. Figure 2 illustrates the design and implementation of the CCM. Gathering and classifying conflict flows is the first stage in the CCM process. Next, two distinct categories are created from the flows gathered using preprocessing: 70% for training and 30% for testing the EFDT classification algorithm. Then, each of the conflict types is classified using the classification method. The following steps are used to organize conflict flows and serve as a guide for designing the classification algorithm.

- Step 1: Implement and execute the EFDT classification algorithm.
- Step 2: Start by analyzing the detection flows.
- Step 3: Check the priority features of conflicting flows.
- Step 4: Check the conflicting flows' Internet Protocol (IP) addresses.
- Step 5: Based on the results of Steps 4 and 5, categorize the different types of conflict.

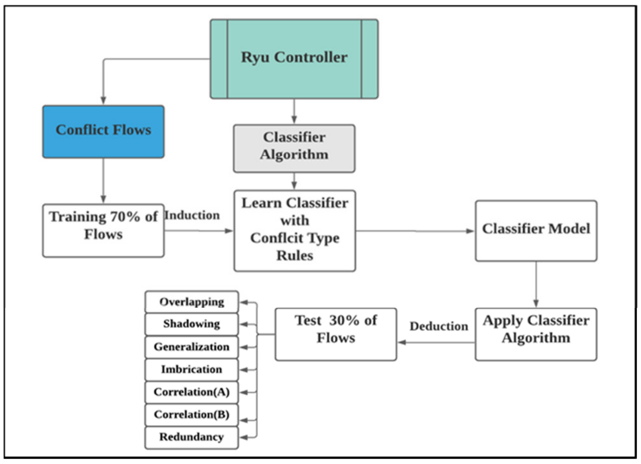


Fig. 2. Conflict classification model.

E. Implementation of the Classification Algorithm

The EFDT algorithm was integrated into the Ryu controller to identify conflicting flows. The algorithm creates a conditional filter that is used to identify flows that may conflict with each other. Once these flows are identified, the Hoeffding tree is used to categorize them into one of the seven conflict types. The criteria used to categorize flows into these conflict types are based on several factors, such as type of traffic, priority of the flows, and current state of the network. The implementation steps of the EFDT algorithm in the Ryu controller can be summarized as follows:

- Create the SEA Generator with decision tree elements.
- Integrate the classifier with the Hoeffding tree.
- Set up the Hoeffding Tree estimator to verify the actions and IP address rules for the produced flows and implement any required adjustments.
- Develop new parameters to supervise the cycle to verify actions and IP address rules.
- Retrieve and include all streams produced in the OpenFlow switch for various stream sizes and make it ready for deployment.
- Utilize 70% of the generated flows to train the EFDT algorithm and 30% to test it.

F. Classification Process

Figure 3 presents the process used to identify seven different types of conflict. The first step is to collect all data flows. Then, the EFDT classification method is applied to the data on conflict flow to classify the different types of conflict at the topology controller level. The conflict flow data are divided into two groups and the algorithm is trained to learn all conflict rules for each type of conflict. This is done to allow the prediction algorithm to be prepared and evaluated.

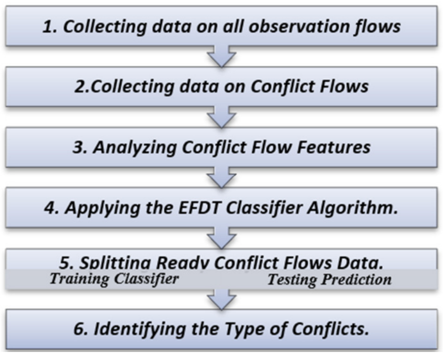

1. Collecting data on all observation flows
2.Collecting data on Conflict Flows
3. Analyzing Conflict Flow Features
4. Applying the EFDT Classifier Algorithm.
5. Splitting Ready Conflict Flows Data.
Training Classifier
Testing Prediction
6. Identifying the Type of Conflicts.

Fig. 3. Classification process.

V. RESULTS AND DISCUSSION

Figure 4 displays the results of the EFDT classifier's production for different flow conflict sizes. The results show that all seven conflict categories were identified correctly, and no conflicts were missed. The classifier accurately identified all conflict flows of various sizes. The two-line sets represent conflicts related to shadowing, overlapping, generalization, imbrication, redundancy, correlation (A), and correlation (B), including shadowing and overlapping conflicts. All conflicts were classified linearly.

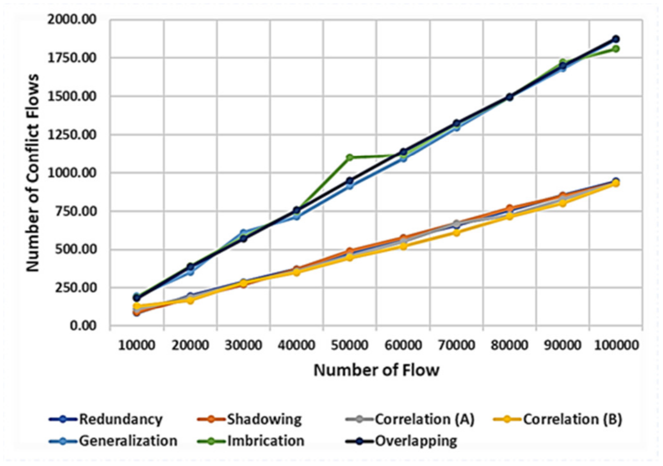


Fig. 4. Types of conflict classified by EFDT.

Figure 4 shows that different types of imbrication conflicts arise in flows containing 50,000 entries. This is a critical observation because the number of conflicts has been increasing during the implementation phase. As a result, the number of conflicts has also increased to ensure that each conflict flow is handled appropriately. During the OpenFlow transition, the flow entries are transferred to a new set of tables, and the size of the flow tables is significant in this case of 50,000 flows. One of the primary criteria used to identify the connection between two sets of flow tables is the MAC address of the flow entries involved. The MAC address is also used to implement several imbrication conflicts, which are classified into seven conflict types by the EFDT algorithm.

Tables II and III extensively investigate the seven conflict groups identified using the EFDT classification approach. These conflict groups were derived using conflict flow data of different sizes. The initial row of the data table displays the aggregate count of flows managed by the OpenFlow switch. In contrast, the remaining rows show the distribution of the discovered conflicts among the seven distinct conflicts categorized by the EFDT and its classification method. As the volume of flow data increases for each conflict type, the number of conflict types also increases. Moreover, the results indicate that the classification method accurately categorizes all flow data into specific conflict types. This is remarkably accurate when considering the rising number of conflict kinds. Tables II and III present the flows classified by the EFDT algorithm for flow ranges of 10,000-50,000 and 60,000-100,000, respectively. A comprehensive analysis was performed to compare the effectiveness of two algorithms in categorizing various types of conflict: the proposed EFDT and the Brew security [14] algorithms. The performance of these algorithms was evaluated using a dataset ranging from 10,000 to 100,000 flows for learning and classification purposes. The results showed that the EFDT algorithm outperformed the Brew security algorithm in classification. Moreover, the EFDT algorithm could identify and categorize seven distinct forms of conflict, while the prior method could only detect six of them.

TABLE II. NUMBER OF FLOWS CLASSIFIED BY THE EFDT ALGORITHM FOR (10000–50000) FLOWS.

Number of flows	**10000**	**20000**	**30000**	**40000**	**50000**
Redundancy	85	198	287	373	477
Shadowing	89	185	272	372	491
Correlation (A)	110	190	282	360	458
Correlation (B)	131	169	283	351	446
Generalization	195	353	611	714	914
Imbrication	190	392	580	751	1011
Overlapping	182	387	569	756	949

TABLE III. NUMBER OF FLOWS CLASSIFIED BY THE EFDT ALGORITHM FOR (60000–100000) FLOWS.

Number of flows	**60000**	**70000**	**80000**	**90000**	**100000**
Redundancy	537	656	755	853	943
Shadowing	577	669	770	849	933
Correlation (A)	550	670	722	826	934
Correlation (B)	521	610	714	801	930
Generalization	1093	1297	1493	1683	1873
Imbrication	1117	1319	1496	1721	1811
Overlapping	1141	1325	1498	1700	1877

VI. CONCLUSION

This study introduces a new ML approach, called EFDT, that efficiently classifies conflict flows in the SDN infrastructure. The algorithm identifies different types of conflict based on flow rule priority, action, protocol, and IP source address. The algorithm was developed using the Hoeffding tree function to enhance its effectiveness and efficiency. A Mininet emulation and the Ryu controller were used to connect a fat tree topology and carry out the experiments. The experiment involved selecting flows from 10,000 to 100,000, increasing 10,000 flows per dataset. According to the results obtained, the EFDT algorithm demonstrated superior classification performance compared to the Brew security algorithm. This means that the EFDT algorithm was better at identifying and categorizing the various conflict types in the dataset. The EFDT algorithm identified and categorized seven distinct forms of conflict, while the Brew security algorithm could only detect six types. This indicates that the EFDT algorithm is more comprehensive and effective in analyzing complex conflict scenarios, which could be helpful in various fields such as social sciences, conflict resolution, and international relations. This study is the first attempt to classify multiple conflict flows in SDN using such methods. Further studies will explore other ML techniques and evaluate them on the same dataset.

REFERENCES

[1] S. Bera, S. Misra, and A. V. Vasilakos, "Software-Defined Networking for Internet of Things: A Survey," *IEEE Internet of Things Journal*, vol. 4, no. 6, pp. 1994–2008, Sep. 2017, https://doi.org/10.1109/JIOT.2017.2746186.

[2] M. H. H. Khairi, S. H. S. Ariffin, N. M. A. Latiff, A. S. Abdullah, and M. K. Hassan, "A Review of Anomaly Detection Techniques and Distributed Denial of Service (DDoS) on Software Defined Network (SDN)," *Engineering, Technology & Applied Science Research*, vol. 8, no. 2, pp. 2724–2730, Apr. 2018, https://doi.org/10.48084/etasr.1840.

[3] M. Karakus and A. Durresi, "A survey: Control plane scalability issues and approaches in Software-Defined Networking (SDN)," *Computer Networks*, vol. 112, pp. 279–293, Jan. 2017, https://doi.org/10.1016/j.comnet.2016.11.017.

[4] E. T. B. Hong and C. Y. Wey, "An optimized flow management mechanism in OpenFlow network," in *2017 International Conference on Information Networking (ICOIN)*, Da Nang, Vietnam, Jan. 2017, pp. 143–147, https://doi.org/10.1109/ICOIN.2017.7899493.

[5] M. H. H. Khairi, P. I. D. S. H. S. Ariffin, P. M. D. N. M. A. Latiff, D. K. M. Yusof, and M. K. Hassan, "A Review of Flow Conflicts and Solutions in Software Defined Networks (SDN)," *IIUM Engineering Journal*, vol. 22, no. 2, pp. 178–187, Jul. 2021, https://doi.org/10.31436/iiumej.v22i2.1613.

[6] P. P. Ray and N. Kumar, "SDN/NFV architectures for edge-cloud oriented IoT: A systematic review," *Computer Communications*, vol. 169, pp. 129–153, Mar. 2021, https://doi.org/10.1016/j.comcom.2021.01.018.

[7] W. Hao, Y. Jiang, and J. Gao, "Detection mechanisms of rule conflicts in SDN based on a path-tree model," in *2017 8th IEEE International Conference on Software Engineering and Service Science (ICSESS)*, Aug. 2017, pp. 336–339, https://doi.org/10.1109/ICSESS.2017.8342927.

[8] M. S. Tok and M. Demirci, "Security analysis of SDN controller-based DHCP services and attack mitigation with DHCPguard," *Computers & Security*, vol. 109, Oct. 2021, Art. no. 102394, https://doi.org/10.1016/j.cose.2021.102394.

[9] C. N. Tran and V. Danciu, "A General Approach to Conflict Detection in Software-Defined Networks," *SN Computer Science*, vol. 1, no. 1,Jul. 2019, Art. no. 9, https://doi.org/10.1007/s42979-019-0009-9.

[10] M. K. Hassan, S. H. S. Ariffin, S. K. Syed-Yusof, N. E. Ghazali, and K. A. Obeng, "A Short Review on the Dynamic Slice Management in Software-Defined Network Virtualization," *Engineering, Technology & Applied Science Research*, vol. 13, no. 6, pp. 12074–12079, Dec. 2023, https://doi.org/10.48084/etasr.6394.

[11] M. H. H. Khairi *et al.*, "The Impact of conflict flows on TCP And UDP Transfer Rate in Software Defined Network," *Innovative Networking Technologies Series 1*, no. 978, 2022.

[12] M. K. Hassan, A. Babiker, M. Baker, and M. Hamad, "SLA Management For Virtual Machine Live Migration Using Machine Learning with Modified Kernel and Statistical Approach," *Engineering, Technology & Applied Science Research*, vol. 8, no. 1, pp. 2459–2463, Feb. 2018, https://doi.org/10.48084/etasr.1692.

[13] M. K. Hassan *et al.*, "DLVisor: Dynamic Learning Hypervisor for Software Defined Network," *IEEE Access*, vol. 11, pp. 84144–84167, 2023, https://doi.org/10.1109/ACCESS.2023.3302266.

[14] B. Mahesh, "Machine Learning Algorithms - A Review," *International Journal of Science and Research*, vol. 9, no. 1, pp. 381–386, 2018.

[15] N. Mehrabi, F. Morstatter, N. Saxena, K. Lerman, and A. Galstyan, "A Survey on Bias and Fairness in Machine Learning," *ACM Computing Surveys*, vol. 54, no. 6, Apr. 2021, Art. no. 115, https://doi.org/10.1145/3457607.

[16] J. Cui, S. Zhou, H. Zhong, Y. Xu, and K. Sha, "Transaction-Based Flow Rule Conflict Detection and Resolution in SDN," in *2018 27th International Conference on Computer Communication and Networks (ICCCN)*, Hangzhou, China, Jul. 2018, pp. 1–9, https://doi.org/10.1109/ICCCN.2018.8487415.

[17] V. Danciu and C. N. Tran, "Side-Effects Causing Hidden Conflicts in Software-Defined Networks," *SN Computer Science*, vol. 1, no. 5, Aug. 2020, Art. no. 278, https://doi.org/10.1007/s42979-020-00282-0.

[18] M. H. H. Khairi, S. H. S. Ariffin, N. M. A. Latiff, and K. M. Yusof, "Generation and collection of data for normal and conflicting flows in software defined network flow table," *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 22, no. 1, pp. 307–314, Apr. 2021, https://doi.org/10.11591/ijeecs.v22.i1.pp307-314.

[19] R. Aryan, A. Yazidi, P. E. Engelstad, and Ø. Kure, "A General Formalism for Defining and Detecting OpenFlow Rule Anomalies," in *2017 IEEE 42nd Conference on Local Computer Networks (LCN)*, Singapore, Jul. 2017, pp. 426–434, https://doi.org/10.1109/LCN.2017.94.

[20] S. Pisharody, J. Natarajan, A. Chowdhary, A. Alshalan, and D. Huang, "Brew: A Security Policy Analysis Framework for Distributed SDN-Based Cloud Environments," *IEEE Transactions on Dependable and Secure Computing*, vol. 16, no. 6, pp. 1011–1025, Aug. 2019, https://doi.org/10.1109/TDSC.2017.2726066.

[21] Mutaz Hamed Hussien Khairi, "Flow Conflict Eliminations through Machine Learning for Software Defined Network," Ph.D. dissertation, Universiti Teknologi Malaysia, 2021.

[22] M. H. H. Khairi *et al.*, "Detection and Classification of Conflict Flows in SDN Using Machine Learning Algorithms," *IEEE Access*, vol. 9, pp. 76024–76037, 2021, https://doi.org/10.1109/ACCESS.2021.3081629.

[23] M. Hamdan *et al.*, "Flow-Aware Elephant Flow Detection for Software-Defined Networks," *IEEE Access*, vol. 8, pp. 72585–72597, 2020, https://doi.org/10.1109/ACCESS.2020.2987977.