telecom

MDPI

Article

Mitigating Timing Side-Channel Attacks in Software-Defined Networks: Detection and Response †

Faizan Shoaib *, Yang-Wai Chow, Elena Vlahu-Gjorgievska and Chau Nguyen

School of Computing and Information Technology, University of Wollongong, Wollongong, NSW 2522, Australia; caseyc@uow.edu.au (Y.-W.C.); elenavg@uow.edu.au (E.V.-G.); chaun@uow.edu.au (C.N.)

* Correspondence: fs984@uowmail.edu.au

† This paper is an extended version of our paper published in 2022 The 6th International Symposium on Mobile Internet Security (MobiSec 2022).

Abstract: Software-defined networking (SDN) is an innovative technology that has the potential to enhance the scalability, flexibility, and security of telecommunications networks. The emergence and development of SDNs have introduced new opportunities and challenges in the telecommunications industry. One of the major challenges encountered by SDNs is the timing side-channel attacks. These attacks exploit timing information to expose sensitive data, including flow tables, routes, controller types, and ports, which pose a significant threat to communication networks. Existing techniques for mitigating timing side-channel attacks primarily focus on limiting them via network architectural changes. This significantly increases the overhead of SDNs and makes it difficult to identify the origin of the attack. To secure resilient integration of SDN in telecommunications networks, it is necessary to conduct comprehensive research that not only identifies the attack activity, but also formulates an adequate response. In this paper, we propose a detection and response solution for timing side-channel attacks in SDN. We used a machine learning-based approach to detect the probing activity and identify the source. To address the identified timing side-channel attack queries, we propose a response mechanism. This entails devising a feedback-oriented response to counter the identified source, such as blocking or diverting it, while minimising any adverse effects on legitimate network traffic. This methodology is characterised by an automated data-driven approach that enables prompt and effective responses. The architecture of this security solution ensures that it has a minimal impact on network traffic and resource usage as it is designed to be used in conjunction with SDN. The overall design findings show that our detection approach is 94% precise in identifying timing side-channel attacks in SDN when compared with traditional mitigation strategies. Additionally, the response mechanism employed by this approach yielded highly customised and precise responses, resulting in an impressive accuracy score of 97.6%.

Keywords: software-defined networking; side-channel attacks; timing attacks; machine learning; intrusion detection; intrusion response

Citation: Shoaib, F.; Chow, Y.-W.; Vlahu-Gjorgievska, E.; Nguyen, C. Mitigating Timing Side-Channel Attacks in Software-Defined Networks: Detection and Response. *Telecom* **2023**, *4*, 877–900. https://doi.org/10.3390/telecom4040038

Academic Editor: Marica Amadeo

Received: 16 August 2023
Revised: 7 October 2023
Accepted: 4 December 2023
Published: 15 December 2023

1. Introduction

Software-defined networking (SDN) is a network-management approach that involves the centralised administration of network resources through programming. SDNs aim to simplify network management, enhance network performance, and reduce operational cost. SDNs have the potential to become critical components of telecommunications networks that enable 5G and 6G technologies. The implementation of SDN can significantly enhance the performance of 5G and 6G networks by enabling a dynamic and adaptable telecommunications network infrastructure that can respond to user demands. In turn, this can enhance user experience and stimulate innovation in the telecommunications industry. SDN can enhance network security by enabling network administrators to monitor and regulate network traffic at a granular level.

SDN has revolutionised how networks are built and managed but has also brought about new security challenges. A significant concern in SDN is the existence of a single controller with access to the entire network, which makes the network vulnerable to compromise. Attackers can also exploit security vulnerabilities in software-defined switches to interfere with traffic and obtain unauthorised network access. Moreover, the dynamic nature of SDN renders real-time identification and response to security attacks a formidable challenge. Therefore, specialised security measures, such as network segmentation, access control, and threat intelligence, are required in SDN networks to address these threats and guarantee the confidentiality, availability, and integrity of network resources.

Several solutions have been proposed to effectively mitigate security challenges in SDN [1]. The majority of existing research on SDN security is primarily concentrated on network segmentation, which involves dividing the network into smaller and more isolated units, thus hindering the spread of an attack. Additionally, Network Access Control (NAC) measures are employed to restrict network access based on predefined policies, thereby ensuring that only authorised devices are permitted to access the network. Furthermore, Intrusion Detection and Prevention Systems (IDPS) are implemented to detect and prevent attacks in SDN. Such systems leverage both signature-based and behaviour-based techniques to detect and impede malicious activity before it can cause damage. Current solutions place considerable emphasis on encryption, which involves converting sensitive data into an unreadable format. To achieve this, transport layer security (TLS) and virtual private network (VPN) technologies have been used.

Within the framework of an SDN environment, an attacker could exploit the centralised nature of the controller to obtain classified data. Through a careful examination of the time patterns of packet transmissions, adversaries can deduce sensitive attributes such as the structure of the network or even the nature of the applications being executed. Furthermore, a comprehensive analysis of the controller's packet response mechanism has the potential to reveal firewall configurations or even identify specific network assets that may be susceptible to future attacks. Owing to the separation of the control and data planes, SDNs are vulnerable to timing side-channel attacks. In conventional networking, control and data planes are implemented on physical infrastructure, such as switches and routers. Therefore, conventional networks are not immune to timing attacks. The architectural differences between SDN and traditional networks have resulted in the disregard for side-channel attacks in SDN deployment. Nonetheless, researchers have identified this as a potential SDN adoption barrier [2]. To date, only a limited number of detection and response techniques have been investigated and implemented [3]. Although these techniques can eliminate time side-channels in an SDN, they cannot determine an adversary's presence or detect the timing probe activity in the network.

In this study, we present a novel intrusion detection and response solution designed to handle timing-probe activities in an SDN environment. The key contributions of this study are as follows:

- To detect the timing side-channel attacks, this solution employs machine learning techniques to accurately predict the network timing probe activity. First, we review, the currently available SDN datasets for the timing side-channel attack parameters. Subsequently, we generated a dataset and utilised our labelling algorithm.
  We evaluated the effectiveness of four commonly used machine learning methods: random forest [4], decision tree [5], XGBoost [6], and bagging [7]. The processed dataset and information from the SDN topology were utilised to detect probe activity and identify the adversary's source machine.
- A response mechanism is implemented to respond to the identified side-channel attack probes. This mechanism involves developing a feedback-oriented response, designed to block or divert the identified source, while minimising any adverse effects on legitimate network traffic. This methodology is characterised by an automated data-driven approach that enables prompt and effective responses.

In contrast to previous studies, both solutions have minimal impact on the SDN environment and are lightweight. Detection is based on machine learning to identify timing attacks and we use data-driven techniques to respond to them, both of which are different from the conventional methods. Both approaches exhibited high accuracy and efficiency. Each element of the solution utilises its own resources, ensuring minimal impact on SDN. The results illustrate that the proposed method is effective in identifying and mitigating timing side-channel attacks in SDN environments.

The remainder of the paper is structured as follows: Section 2 reviews related works. Section 3 describes the architecture of the proposed solution. Section 4 presents the experimental setup and results. Section 5 provides a discussion of the advantages and limitations of this study. Finally, Section 6 concludes the paper.

2. Related Work

SDN has emerged as an attractive option to perform diverse functions in 5G and 6G telecommunications networks. Network programmability offers flexibility in the implementation of telecommunications networks. SDN offers flexibility in the implementation of telecommunications networks, allowing for the creation of customised logical 5G networks with dedicated and shared resources [8]. SDN enables network operators to achieve real-time optimisation of the network performance and dynamic resource allocation. Ahvar et al. [9] conducted a comprehensive analysis of the incorporation of SDN to facilitate the operation of 5G-based IoT devices. The authors of this work highlight the future aspects and challenges in this field, including 'protocol and standardization', 'meeting slicing requirement for 6G', and 'trust and security', as some of the potential challenges faced when combining these technologies. Lin et al.'s work [10] highlights potential challenges in implementing an SDN-based 5G IoT network. They presented an SDN-based 5G IoT network architecture and listed potential challenges, such as the quality of open-source software, availability of open-source SDN for 5G, scarcity of real-world testing, and effective network rollouts. Sarica et al. [11] proposed an SDN intrusion detection method for 5G networks. This study demonstrates the existence of security threats and intrusions in SDN-based 5G telecommunications networks.

In recent years, extensive research has been conducted on the SDN security. The primary focus areas include intrusion detection, controller and data-plane security, access control, and authentication. Various researchers have proposed strategies to address these vulnerabilities and to overcome these challenges. Ahmad et al. [12] provided an elaborate discussion of the security concerns that 5G mobile networks face when integrating SDN. Joberto et al. [13] proposed an intrusion detection system-based security architecture for detection and response in SDNs. Manu et al. [14] employed flow monitoring to design an intrusion-resistant SDN architecture. Nicolas et al. [15] recently developed SYNAPTIC, an automated system designed to verify security chains in SDN. This approach was subsequently employed to develop a security policy validation solution for SDN-based networks [16]. Several researchers have proposed SDN-based security architectures for networks with 5G and 6G connectivities [10].

Several computer systems have successfully targeted side-channel attacks. Paul Kocher [17] introduced a side-channel attack known as a timing attack. His work has since become a significant source of attacks that involve timing and power evaluation. Timing-based side-channel attacks are a significant threat to environments sensitive to timing changes. The fundamental concept underlying these attacks is that the processing time for various operations may vary, depending on the type of data being processed [18]. Studies have indicated that timing-based side-channel attacks can be executed using both hardware and software [19]. Sepulveda et al. [20] investigated side-channel timing attacks within on-chip systems and highlighted how precise timing side-channel attacks can enable adversaries to access processor data. Dunlap et al. [21] employed timing-based side-channel attacks to detect anomalies in the control systems.

Timing-based side-channel attacks are becoming increasingly prevalent in SDNs, owing to the separation of control and data planes [22]. In a previous study [23], we proposed the use of random delay to prevent timing side-channel attacks. Yoon et al. [24] published an in-depth study on the risks posed by SDN. The authors classified data breaches in the data plane as packet round-trip time (RTT) attacks. The study identified that an intruder can determine the control path latency to evaluate the communication patterns. Conti et al. [25] proposed the 'Know Your Enemy' attack, which uses timing information to determine an SDN's network configuration. In this study, the proposed countermeasure redirected all traffic to the proposed solution, thereby increasing RTT for all traffic. In [26], timing probing packets were used to ascertain data plane event parameters and control plane characteristics. The method proposed in this study is dependent on intensive flow insertion operations and uses the switch's computation and queuing resources, resulting in data rate restrictions.

Liu et al. [27] have also demonstrated timing-based side-channel attacks. They proposed that the network information can be derived from the delay caused when a packet is sent to the controller. Hou et al. [28] recommended delaying the implementation of the flow at random more recently to conceal the disparity in timing between multiple queries. This method delays flow installation whenever the MAC address of the host changes. The authors did not consider the manipulation of the MAC addresses. Adversaries can be easily fabricated and employ various MAC addresses. Arsalan et al. [29] also presented a timing attack technique for vehicular ad hoc networks. Anyi Liu et al. [30] proposed a technique to detect timing channel intrusions in SDN. Although the results demonstrate that the method can identify timing channels in an SDN cloud, it can be argued that an adversary could potentially exhaust cloud resources. In addition, it reduces the ability of an adversary to identify the virtual machine that is being exploited. [31] addresses timing side-channel attacks on the IoT using machine learning techniques. The findings suggest that this technique can be used to identify the timing side-channel in the IoT.

Machine learning has emerged as an effective approach for identifying the various types of SDN attacks. Moreover, it has been utilised for traffic flow management and route optimisation [32]. Machine learning techniques have been demonstrated to produce accurate results with low false positive rates. Patikiri et al. [33] recently proposed a machine learning algorithm to route applications and improve the performance of software-defined vehicular networks. Tayhan et al. [34] presented a thorough analysis of the integration of machine learning and deep learning techniques for intrusion detection in SDN. This study demonstrates that most research in this field concentrates on high-rate distributed denial of service (DDoS) assaults, whereas only a minority of studies examine low-rate attacks. Alzahrani et al. [35] designed an SDN intrusion detection system based on machine learning. This paper illustrates the application of multiple classification techniques based on machine learning for attack detection. Common types of attacks, such as denial of service (DoS) and network scanning, were detected in this study. Mykhailo et al. [36] employed machine learning techniques to detect DDoS attacks in the SDN. By comparing the average duration of each session with the amount of time required to access the server from a specific IP address, they were able to identify anomalies. Ahnaf et al. [37] evaluated machine-learning algorithms for SDN security. They employed a variety of machine-learning algorithms for a customised dataset and evaluated the performance of each algorithm. Aslam et al. [38] proposed a DDoS detection and mitigation solution for IoT devices based on machine learning and SDN. This method detects attack patterns by using an SVM algorithm. Zakaria et al. [39] presented an SDN attack detection framework based on machine learning. In this study, we evaluate a dataset of multiple attacks. The primary objective of this study was to collect data efficiently from a network. Patikiri et al. [40] presented a comprehensive study of Knowledge-defined Networks (KDN). They elaborated on how KDNs use machine learning models to combine the management and control planes of SDN to form a knowledge plane.

One of the primary challenges during the incorporation of machine learning into SDN security is the absence of high-quality labelled datasets [41]. The compilation and categorisation of network traffic data is a time-intensive and expensive undertaking that presents a challenge to the coverage and scalability of security solutions that rely on machine learning. In the following section, we present a thorough examination of the currently accessible datasets for SDN, along with a critique of their characteristics and features. This section also highlights the need for creating a self-generated dataset.

We reviewed current response methodologies to enhance the effectiveness of the detection solution. We observed that the response mechanisms for SDN security breaches have not received sufficient attention. In this review, we have presented two major observations. First, an IDS implemented for SDN security lacks a concrete response mechanism. The emphasis has been on dropping or redirecting the entire traffic from the attacking IP address, which does not facilitate the development of granular flows owing to the performance overhead [42]. Second, existing solutions focus primarily on the fastest response time and neglect the integrity of the responses. Hence, once a response or flow is sent, it remains until the network administrator performs manual actions to remove it. They lack a verification layer that can validate whether responses should be permanent or transient based on the feedback received from previous responses [43]. Table 1 highlights the gaps in response mechanisms. To date, focus on response mechanisms for SDN security breaches has been inadequate. The inadequacy of concrete response mechanisms is a significant issue and existing solutions fail to preserve the response integrity. Emphasis on rapid response times has led to solutions that disregard the importance of the response integrity. Owing to the performance overhead, an IDS implemented for SDN security fails to develop granular flows.

Table 1. Overview of Recent Response Mechanisms in SDNs

Title	Solution and Features	Issues and Limitations
Timing-based Reconnaissance and Defense in Software-defined Networks [44]	OpenFlow Timeout Proxy: Sends default responses if the controller is taking more time.	Limited responses: An attacker can generate traffic to fill the switch flow table. Limited to less than 10,000 packets/s.
SDN/NFV security framework for fog-to-things computing infrastructure [45]	DDoS/botnets detection and response: Detects attacks and installs Drop rules.	Sends drop rules that can drop legitimate traffic. Also, there is no revision of the drop rule suggested.
A Framework for Real-Time Intrusion Response in Software-defined Networking Using Precomputed Graphical Security Models [46]	Real-time intrusion response in SDN using precomputation: Estimates possible attack paths.	Response drops all outgoing traffic from the affected VM, which impacts all services.
A Security Architecture Proposal for Detection and Response to Threats in SDN Networks [13]	Network architecture for SDN with detection/protection mechanisms, including IDS technology.	The solution has a high response time (6–7 s) and the response drops all traffic from the node.
Counteracting Attacks From Malicious End Hosts in Software-defined Networks [42]	Propose techniques for securing the Controller, switches, and legitimate end hosts from malicious end host attacks in SDNs	The solution drops incoming flow requests as a proactive response. No feedback or monitoring is used on the dropped flow request.

Despite the increased prevalence of these attacks, little effort has been made to mitigate them. The emphasis has predominantly been on modifying the network architecture to prevent such attacks rather than devising remedies to combat them. To detect timing-side

channel attack activity, a machine learning-based approach was proposed as an initial effort [47]. This study represents a continuation of these efforts, providing a detailed comparison of machine learning classification techniques with the algorithm, as well as a comprehensive response solution.

3. Proposed Methodology

In contrast to other forms of attacks, timing side-channel attacks can be performed gradually. Consequently, they stand out and are challenging to detect and avoid. Timing attacks allow an attacker to infiltrate a network using legitimate probing packets to ascertain RTT. Consequently, typical methods based on the traffic volume, packet size, or packet type cannot be used to mitigate such threats. Machine learning is an artificial intelligence (AI) component that enables software programs to become increasingly adept at forecasting without explicitly designing them to do so. Machine learning algorithms use historical data as inputs to anticipate new output values. These features make this technology an excellent option for identifying timed side-channel attempts. The attributes of the timing probe activity are fed into the machine learning model and the prediction model determines whether the incoming packets are attack probes or regular traffic. We used machine learning to develop the proposed solution as it can predict the results accurately and effectively. To determine the most suitable classification technique, we compared the results of our model with those of four classification methods. The response mechanism is distinct because of its strategic use of historical data, which enables it to expedite response techniques against the roots of malicious activities. Compared with some of the more common response mechanisms, which generally value the speed of the reaction above thorough investigation, this approach stands out. Our proposed technique generates a customised response, as opposed to the potentially ineffective strategy of blocking all traffic from a specific IP address. This tailored response is formulated using various parameters such as the IP of the destination and the type of traffic. The resulting customised flow entries ensured the preservation of legitimate traffic.

3.1. Solution Architecture

The proposed solution architecture offers a framework for identifying and responding to timing side-channel attacks in SDN environments using two key phases: detection and response. Collectively, these methods offer a comprehensive framework for identifying and responding to these attacks.

3.1.1. Detection Phase

Figure 1 shows the workflow architecture of the detection phase. SDN offers centrally managed programming for network functionality via software applications that use APIs. The SDN application layer facilitates network integration with proprietary open-source network applications and automation tools. We leverage this layer for the integration of our machine learning-based solution with SDN. The detection phase is also connected to the controller to obtain the visibility of the entire SDN topology. The detection phase is intended to receive incoming packets (Packet-Ins) from any SDN environment, sample them, and check for timing probe activity. It anticipates the timing of the attack, then attempts using an algorithm trained on a historical dataset. Along with each received packet, it extracts information about the source of each packet. This is achieved by obtaining flow tables from all switches connected to the SDN topology. Subsequently, this information is used to locate the adversary's source. The output of this phase depicts the timing of the attack-probing activity and accurately displays its origin. A network administrator can use this output to perform further actions based on a source machine. In the case of the proposed solution, this output was used in the subsequent response phase. Such responses can be denied, permitted, or monitored. The innovative method of integrating SDN and machine learning makes this approach an efficient method for detecting timing probes and determining the source of attacks.

The operational phase of machine learning-based detection occurs in real time and has potential implications for the network performance. To mitigate this impact, the detection phase operates on a dedicated virtual server, thereby ensuring that the SDN environment remains unencumbered. Our design decision enables us to retain the advantages of centralised control while mitigating the performance disadvantages. As a result, the incorporation of SDN and machine learning in our solution proves to be an effective approach for identifying timing probes and determining the origin of attacks, while maintaining an acceptable network performance.

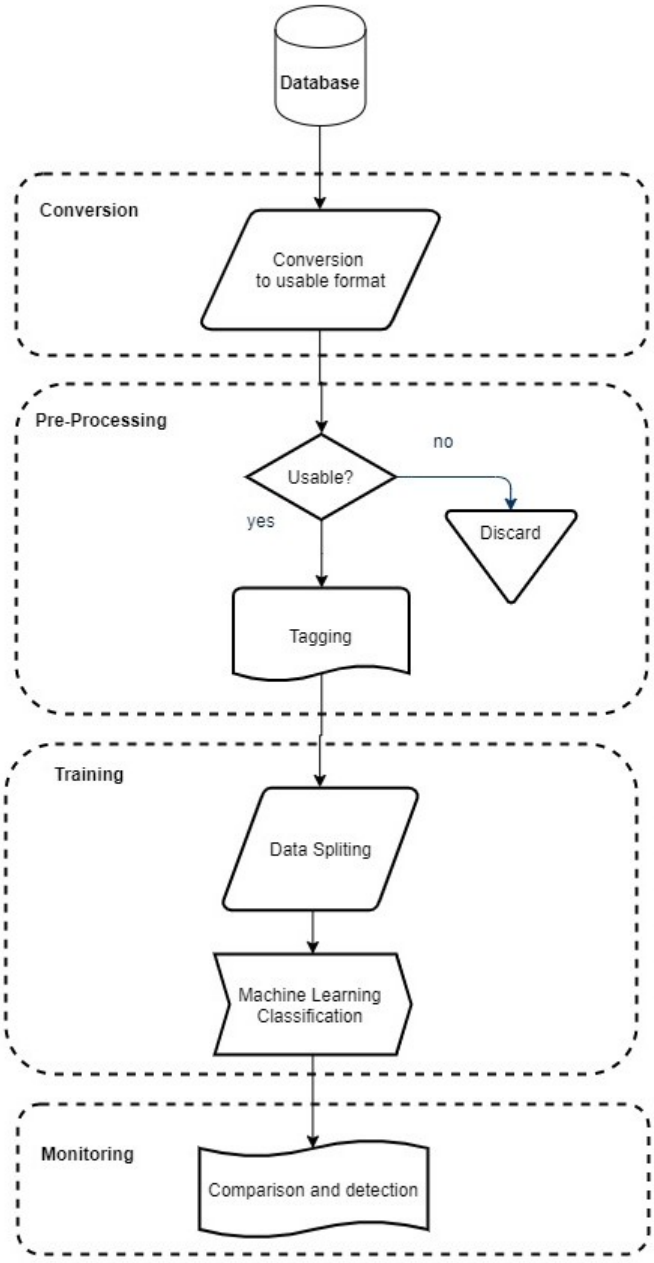

Figure 1. Workflow of Detection Phase.

3.1.2. Response Phase

Figure 2 shows the workflow architecture of the response phase. The methodology of this response architecture is based on historical data and includes four essential components: the Data collection module, Data analysis module, Response module, and Feedback module. The data collection module is designed to gather and preserve historical information from an attached database of network traffic and security events. The subsequent phase of the

solution architecture involves a data analysis module that scrutinises the historical data gathered in the preceding phase. The data analysis module determines whether the source has been identified previously by comparing real-time data with historical data collected via the data collection module. The Response module adhered to the established flow of the Data analysis module. It incorporates a temporary rule and supervises the traffic of the rule, thereby significantly contributing to the overall effectiveness of the response mechanism. The final component of the response-phase architecture is the feedback module. The rules and policies were refined by providing feedback to the data collection module based on the performance of the response module. The module determines whether the rule inserted in the preceding section has been triggered. If the attacker repeatedly and frequently violates the rule, the module establishes a customised and unique permanent drop rule. Data analysis and feedback systems are pivotal elements of the response phase. This approach, which is based on the analysis of historical data, offers a more efficient response mechanism for timing side-channel attacks in SDNs by allowing for a tailored response to each attack.

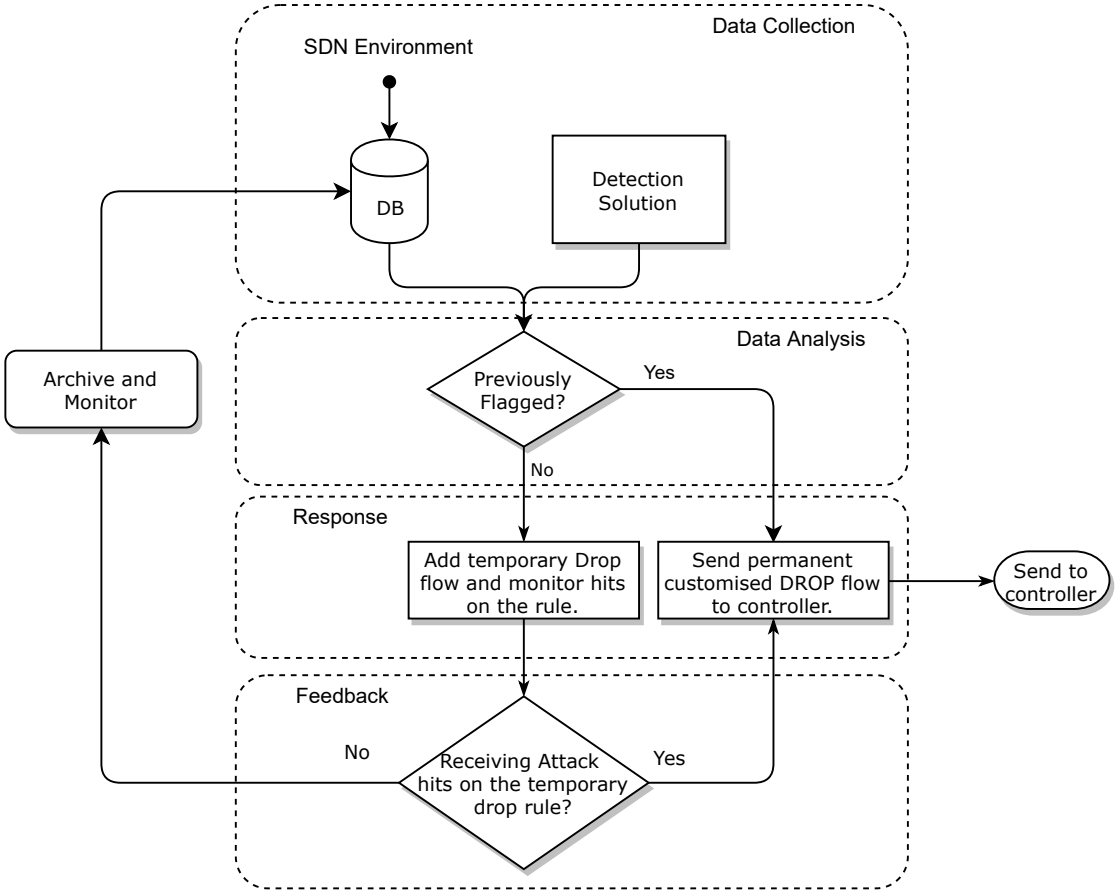

Figure 2. Workflow of Response Phase.

3.2. Dataset

To date, a dataset with timing side-channel attack footprints in an SDN environment is not available. This prompted us to construct the dataset. Some of the commonly available datasets with traffic for SDN environments are reviewed in Table 2. The proposed algorithm for sampling timing probes requires the dataset to consist of the following fields: source MAC address, destination MAC address, source IP address, destination IP address, timestamp, TTL, and protocol. An examination of accessible datasets containing SDN traffic was conducted and the key findings are presented in Table 2. The results indicate that existing datasets pertinent to SDN traffic lack the necessary fields required for the effective execution of the proposed algorithm. To calculate the number of samples with unknown

destination MAC addresses, as well as the number of samples with known destination MAC addresses, we require source and destination MAC addresses. The present datasets for SDN environments are missing these fields, which are essential for the implementation of the timing side-channel algorithm. This motivated us to develop the dataset. For the collection of samples in the dataset, we develop an API, called "Listener API", which is a Python-based RYU application that receives specific network flows and stores them in a database. This application also ensures that the database receives network topology information. In addition, it extracts the source IP and MAC addresses and the destination IP and MAC addresses in conjunction with the timestamps for all 'packet-in' requests. It can determine whether a request is carried via the 'User Datagram Protocol' or the 'Transmission Control Protocol' (TCP). The database is also notified of the switch name and port from which the request originates.

Table 2. Dataset Review.

Dataset Name	Source MAC Address	Destination MAC Address	Source IP Address	Destination IP Address	Time Stamp	TTL	Protocol
SDN Intrusion Detection [48]	x	x	✓	✓	x	✓	✓
DDoS SDN Dataset [49]	x	x	✓	✓	x	x	✓
UNR-IDD Intrusion Detection Dataset [50]	x	x	x	x	x	✓	✓
NSL-KDD [51]	x	x	x	x	x	x	✓
Proposed Dataset for Solution	✓	✓	✓	✓	✓	✓	✓

Dataset features and their respective indications.

3.3. Solution Blueprint

The overall framework of the proposed detection solution comprises a listener API, database, conversion, preprocessing, and training modules, as well as a monitoring application. This is primarily located on the application layer of the SDN. A listener application "Listener API" is used to connect an SDN controller with this solution. This API is designed to receive a copy of selective incoming requests to the controller. These were all requests with unknown destination MAC addresses. It also fetches the network topology, switch port, MAC, and IP addresses, along with the flows. To the East, this API connects to the solution database. It stores information from the SDN environment in a database. We deployed MongoDB [52] as the database for this solution. MongoDB is updated using the listener API, which operates within the SDN topology. The monitoring module compares the actual traffic with the learned model. It identifies the timing side-channel activity by comparing samples and implementing anomaly detection techniques.

The proposed response solution blueprint for responses against timing side-channel attacks is based on historical data. The solution comprises several essential modules including data collection, data analysis, response, and feedback modules. The integration of the detection and response mechanisms is shown in Figure 3. In the following subsections, we delve into the specifics of each module, shedding light on their individual functionality and collective synergy for the detection and response to timing side-channel attacks.

3.3.1. Conversion Module

The Conversion module plays a crucial role in the conversion of raw network traffic data into a JavaScript Object Notation (JSON) format, which is suitable for the preprocessing module. The conversion includes multiple tasks, such as packet reordering, packet fragmentation, and protocol conversion, to ensure that the data are in a suitable format for further processing. This module uses the 'mongoexport' function to facilitate this

transformation. This format not only ensures compliance with the following step of dataset construction but also simplifies data handling in the later phases of our solution. After successfully converting the data samples, the pre-processing phase was initiated to prepare the data for training.

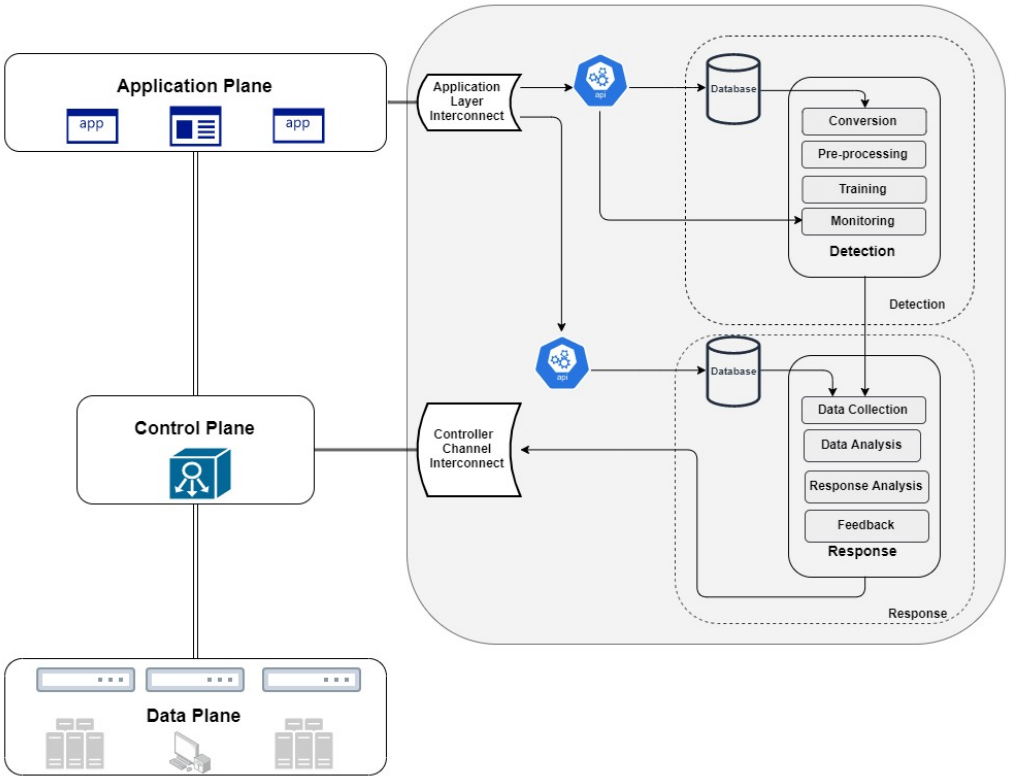

Figure 3. Workflow of Detection and Response Solutions.

3.3.2. Pre-Processing Module

Preprocessing is a component of the detection solution that ensures that the data are ready for training and helps in cleansing, organising, and structuring the data. During the pre-processing stage, excessive fields were removed from the data. In addition, broadcast traffic is filtered at this stage. From the dataset, the preprocessing module identifies existing and non-existing flows in the switch table. This aids the application of tag flows. Furthermore, the request types were determined and segregated at this stage. Preprocessing is performed using a Python-based application that accepts JSON data as input from the preceding stage. The samples were labelled as malicious timing attempts by using the algorithm described in the following section. Figure 4 provides a comprehensive depiction of all functions that occur during the preprocessing stage.

3.3.3. Algorithm (Labelling Scheme)

The process of classifying and tagging timing attack samples based on MAC addresses is outlined in Algorithm 1. A batch of ten samples was chosen and the number of samples with unknown destination MAC addresses was determined. Packets with unknown MAC addresses were sent to the controller and stored in the database for further analysis. It is important to note that these samples may include traffic from discovery protocols such as the Simple Network Management Protocol (SNMP) and the Link Layer Discovery Protocol (LLDP). These discovery protocols are typically used for network management and topology discovery, serving legitimate purposes in network operation. To account for legitimate network discovery requests made via these protocols, we set the threshold for the selected samples to 80 percent. This threshold ensures that only samples significantly deviating from regular traffic patterns are considered as potential threats.

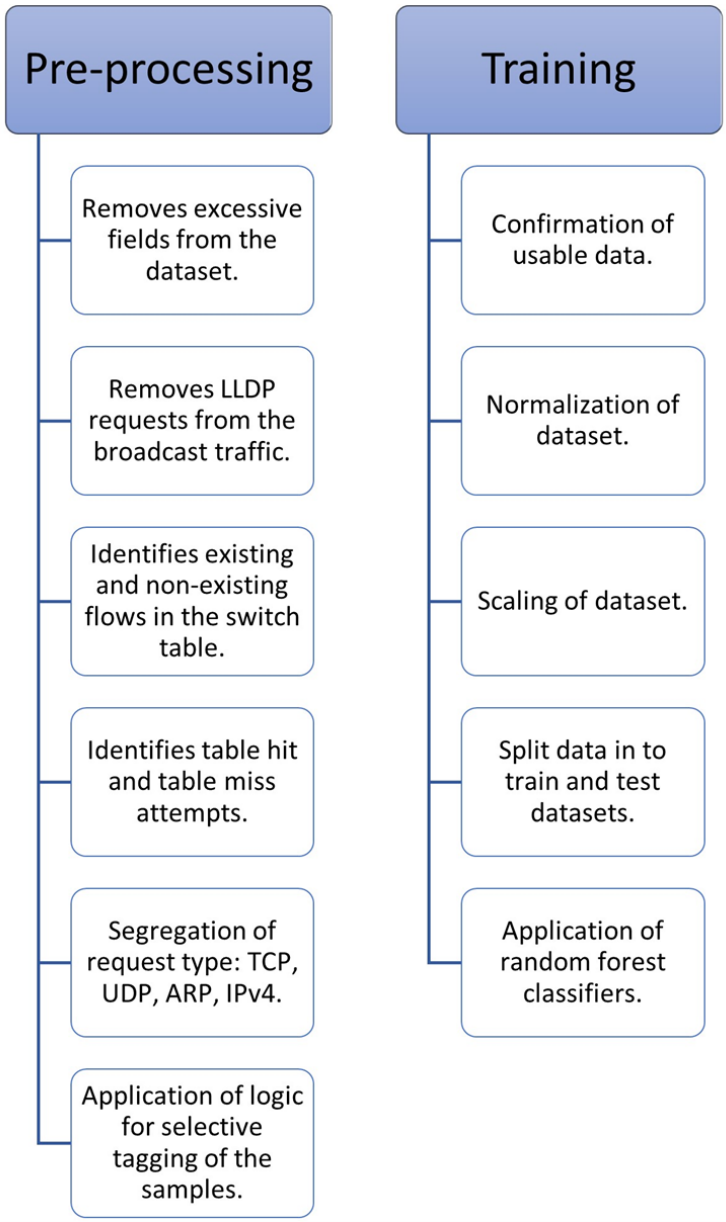

Figure 4. Functions of Pre-processing and Training Phases.

3.3.4. Training Module

The solution trains the machine learning model using pre-processed data. The training module applies the classification algorithm determined in the previous section to form a prediction model. Furthermore, the training component was responsible for scaling, normalising, and validating the dataset. In the experiments, we examined four commonly used machine learning approaches: random forest [4], decision tree [5], XGBoost [6], and bagging [7]. This was achieved by implementing each algorithm using the proposed approach. This step was performed to determine the best machine learning classification algorithm for the training stage. Figure 4 depicts all the functions that occur during the training stage.

3.3.5. Monitoring Module

The monitoring application was designed to collect real-time information regarding the traffic that occurred during the deployment phase. Its main function is to establish a connection with a trained machine learning model and then correlate the activity from the probes. By performing this operation, the monitoring application obtains real-time

information regarding traffic patterns and analyses the timing side-channel attacks. This identified activity is then relayed to the data collection module of the response mechanism to generate an appropriate countermeasure against potential adversaries.

Algorithm 1 Classification algorithm for attack packets [47]

```
N = Number of samples with no destination MAC addresses.
T = Total Number of collected samples.
i = N;
t = T;
n ← 10;
p ← 0.8;
if i > 0 then
    if t ≤ n then
        s = i/t;
        if s ≥ p then
            tag;
        else
            return;
        end
    else
        return;
    end
else
    pass;
end
```

3.3.6. Data Collection Module

The data collection module collects and stores historical data on the network traffic and previously detected adversaries. This module is implemented using various data collection tools such as network traffic analysers, listener API, and sFlow [53]. The data collected via this module includes information about network traffic patterns, security events, and historical responses to security threats. We used the MongoDB platform for this collection. In addition to its primary functions, the module served as the initial stage of the response mechanism. It operates by receiving the output generated via the detection phase and then utilising these data as the input. When the Detection phase successfully identifies the source of the attack, it transmits the crucial information to the data analysis module of the response solution. Specifically, it contains the MAC addresses, IP addresses, and switch ports used by the intruder. This dynamic, real-time process allows for the solution to gather and evaluate key information quickly. Therefore, they form an integral element of the response mechanism by providing actionable information regarding potential threats.

3.3.7. Data Analysis Module

The data analysis module is responsible for analysing the historical data collected via the module to identify patterns and trends in network traffic and security events. Its primary function is to analyse network traffic and security events for unusual behaviours or trends. This task requires a thorough comprehension of the present status of the network, which necessitates the utilisation of both real-time and historical data. The primary objective of this module is to determine whether the source detected during real-time data capture has been identified previously. For this process, real-time data were thoroughly compared with previous data collected via the Data Collection module. Consequently, the output generated via the data analysis module consists of a set of rules and policies that are precisely constructed based on the analysed data. The data analysis module plays a crucial role in developing provisional rules and policies. This enables the response module to generate informed responses.

3.3.8. Response Module

The response module is responsible for reacting to timing side-channel attacks, which occur based on the rules and policies generated via the data analysis module. Real-time responses are enabled by utilising specific response mechanisms, particularly OpenFlow messages. A key characteristic of the response module is its ability to customise the response to each attack attempt, ignoring a generalised one-size-fits-all method in favour of a more tailored and efficient approach. The functionality of the response module is supported by two distinct components. The first component is activated when the data analysis module identifies a previously determined source as the attacker. In this case, the component builds a customised fixed-drop rule, which is then sent to the controller. The creation of this flow is a complex procedure and is the direct result of comprehensive data analysis. It targets specific parameters, such as the source, destination, and protocol, as opposed to hindering all the traffic from the host. When the data analysis module detects an attack launched from a previously undetected source, it triggers the second component of the response module. In response, a temporary drop rule is generated. The transient nature of the rule ensures that it is removed immediately if no further malicious activity is detected from the source. The primary purpose of this rule is to validate and provide feedback on prospective reoccurrences in future network encounters. These two components of the response module represent an advanced and dynamic real-time network threat response strategy for timing side-channel attacks.

3.3.9. Feedback Module

The feedback module serves a twofold purpose: it furnishes evaluative data to the Data Collection Module and collaborates with the Response Module's customised temporary drop rule to modify system behaviour. The hit counter monitor, which is implemented on OpenFlow switches, facilitates the monitoring of the customised flows generated via the response solution. If multiple rule matches are determined, this triggers a data exchange that culminates in a permanent custom drop flow. However, when there are no matches to custom drop rules, the feedback is stored in the database for subsequent monitoring and trend analysis. This archival step assists in refining the flows produced via the data analysis module. This iterative process, integrated with the data from these operations, optimises the flows generated via the Data Analysis Module, augmenting the overall efficacy of the response mechanism.

4. Experiments and Results

The base simulator for the experimental stages was Mininet [54], which was deployed on a Linux-based virtual machine (VM1). As an SDN controller, we used RYU [55], which was deployed as a remote controller. Considering both the processing resources required by Mininet and how to create a realistic SDN scenario, we installed an RYU controller on a separate virtual machine (VM2). In the application layer, we utilised RYU apps integrated with custom Python-based applications. Detection and response solutions have been developed using dedicated Linux-based virtual machines. This ensures that solution processing and predictions do not influence SDN resources. The solutions were connected to the RYU controller using separate APIs in the application layer of the SDN environment.

In the network topology for the detection section, the network architecture comprised four OpenFlow switches. Each switch is linked to a remote RYU controller on the northern side. Each switch in southbound traffic was connected to two hosts. To contemplate traffic congestion on the links, each link between the host and switch in the topology was programmed to have a delay of 2 milliseconds (ms). RYU applications enable the controller to install flows on all the switches. Using the RYU controller, we ran an application to handle spanning trees and Address Resolution Protocol (ARP) requests in the network topology. The APIs forward selective 'Packet Ins' to MongoDB for both the solutions. MongoDB was installed on the same machine as the solution and acted as a database for each model.

Within each solution, separate Python applications are developed for each module. Figure 5 shows the experimental setup used for the detection and response solutions.

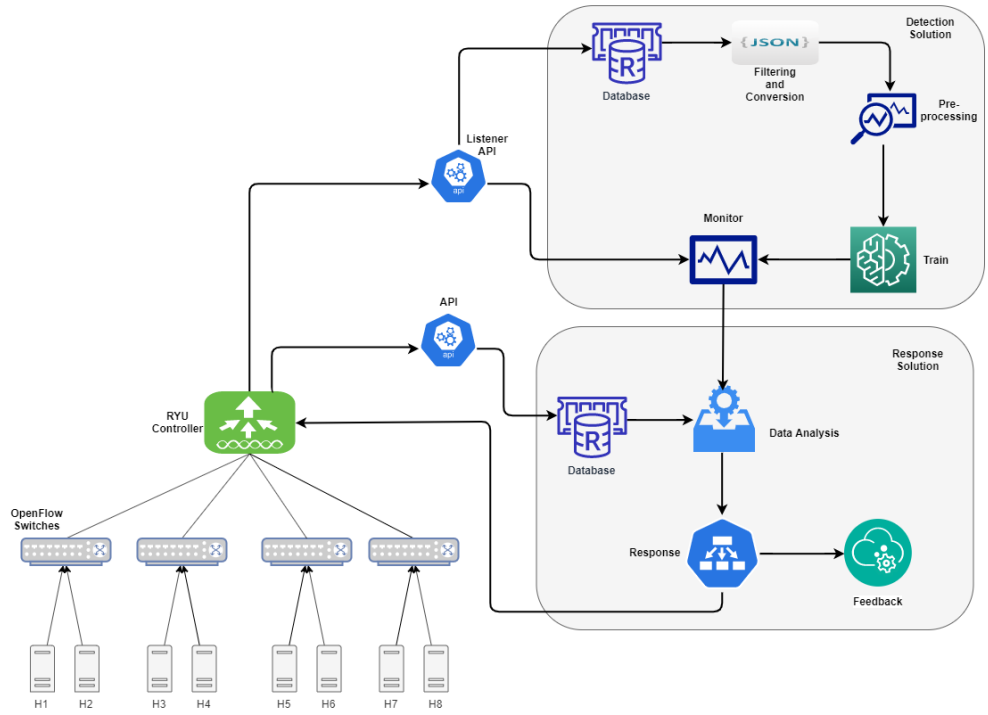

Figure 5. Implementation of Detection and Response Solution.

We evaluate the effectiveness of our proposed detection approach in terms of precision, recall, and F1 score. Each of these evaluation parameters, which collectively provide a comprehensive assessment of the model's performance, is described briefly in the following section.

Precision Score: Precision is the proportion of correctly predicted positive classes relative to the total number of predicted positive classes. The calculation for the Precision Score is articulated in the following equation:

$$\text{Precision} = \frac{\text{True Positives (TP)}}{\text{True Positives (TP)} + \text{False Positives (FP)}} \tag{1}$$

Recall Score: The recall is defined as the proportion of predicted classes that exist. It indicates how many positive instances have been correctly detected via the model. The following formula represents the recall score expression.

$$\text{Recall} = \frac{\text{True Positives (TP)}}{\text{True Positives (TP)} + \text{False Negatives (FN)}} \tag{2}$$

F1 Score: The F1 score is the average of the scores for precision and recall. It combines the precision and recall metrics into a single metric. The following expression represents the formula for the calculation of the F1 score.

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{3}$$

Following the custom topology setup, the initial phase of dataset construction was initiated. Using a script, legitimate and probe traffic is transmitted from multiple nodes within the topology. To construct the dataset, a variety of networking utilities were utilised [56]. Specifically, TCP Netcat plays a role in establishing and maintaining resilient data streams, thereby simulating traffic patterns that closely resemble real-world scenarios. On the other hand, UDP Netcat was utilised for lighter connectionless data transfers. The reachability

of the host was tested using the ICMP protocol via the execution of the 'ping' command. Nmap was utilised to generate network traffic by scanning target IPs and subnets and employing various types of scans to generate distinct packets, such as TCP SYN and UDP packets. This simulation successfully replicated the diverse network conditions for testing. Finally, Arping [57] was used to dispatch ARP requests to multiple IP addresses within the local network. In most cases, these requests prompted hosts to respond with ARP replies, thereby providing a straightforward method for generating local network traffic and facilitating tasks, such as MAC address resolution. Each of these utilities plays a crucial role in the creation of a comprehensive and accurate dataset tailored specifically to the SDN environment. For probe traffic, we utilised attack ping scanning, broadcast traffic, and map scan traffic. We ran these scripts for 12 h to obtain a comprehensive dataset.

The features of concern in formulating a dataset are the source and destination MAC addresses, timestamps, time-to-live (TTL), and protocol. To train the machine learning model, we extracted flows with unknown destination MAC addresses. The Listener API selects 'packet-in' and sends samples to MongoDB. A total of 60,649 samples were collected from this dataset. We converted these samples to JSON to implement them in our machine learning model. This phase employed the 'Mongo-export function'.

After data collection, a Python-based preprocessing application was used to clear, organise, and label the samples. This application transforms the unprocessed dataset into a usable format, which is an essential step in any machine learning method that prepares raw data for the subsequent machine learning model. This application, which is intended to accept JSON data as input from the preceding preprocessing phase, separates and labels the data samples. Based on the unique algorithm detailed in the following section, these samples were identified as malicious timing sensors.

In the training phase, we implemented a detection algorithm (detailed in Section 3.3.3) and selected a classification technique for our machine learning model. To implement the detection Algorithm 1 and sample timing probes effectively, the following fields are extracted from the dataset: 'source MAC address', 'destination MAC address', 'source IP address', 'destination IP address', 'time stamp', 'TTL', and 'protocol'. These features were used for the data classification. Source and destination MAC addresses are crucial for determining the number of samples with known and unknown destination MAC addresses. We select ten samples at a time and counted the number of samples whose destination MAC addresses were unknown. We used a Python-based training application to train the machine learning model. This pre-processing gives us 42,489 samples. The training module is also responsible for machine learning classification. We compared four tree-based machine learning classification techniques to determine the best classification method for training: random forest [4], decision trees [5], XGBoost [6], and bagging [7]. We implemented each algorithm using the proposed approach. This step is performed to determine the best classification algorithm for the detection stage. The results of this section are analysed and the results from this step are detailed in further sections. The outcome of this step validates that the random forest algorithm is the best-suited classification method for the training phase.

The network topology was reset during the monitoring phase of the experiment. Following this, we initiated a monitoring application that was programmed to precisely identify the IP and MAC addresses of the malicious host. Additionally, the application identifies the switch and port to which the malicious host is connected. Throughout the twelve-hour experiment, timing probes were generated at multiple intervals with varying attack frequencies per deployment. We recorded the number of attack samples in each phase. Multiple experiments were conducted to ensure the validity of our solution and account for potential variations in background traffic. The results were derived from the data obtained by processing the data in MongoDB using the characteristics of the monitoring app.

To verify the performance of detection Algorithm 1, we must first verify that probing traffic generates samples with unknown destination MAC addresses. To achieve this, we gathered the network statistics during the monitoring process. Figure 6 shows the network

statistics for flows with known destination MAC addresses, flows with unknown destination MAC addresses, and the total number of flow samples. These parameters were monitored for ten attack attempts during the experiment and are denoted on the x axis. These are the most significant elements of the labelling procedure. During the experimentation phase, it was discovered that the number of timing probes (attack traffic) has a direct relationship with the number of flows with unknown destination MAC addresses. This validated the proposed classification algorithm and methodology.

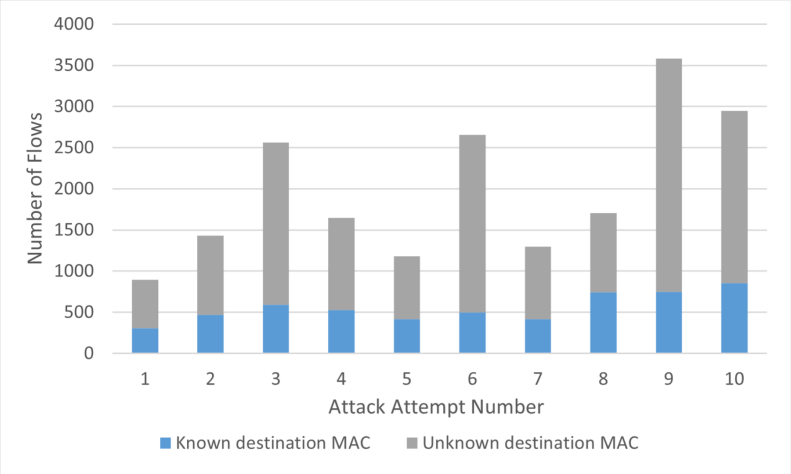

Figure 6. Analysis of Network Flows via Destination MAC Address Recognition.

False positives and negatives for each attack attempt are summarised in Figure 7 [47]. The *x* axis represents the number of attacks in a sequence. Clearly, the overall procedure produced a small number of false negatives. False negatives manifest in successive attack attempts. This also suggests that the solution involves learning more about the network and adapting to varying background traffic.

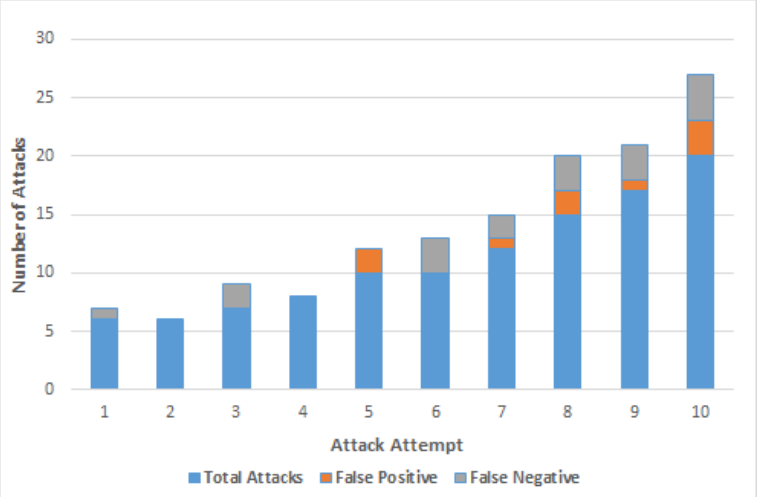

Figure 7. Summary of False Positives and False Negatives per Attack Attempt.

We refer to equations for evaluating our model's precision score, recall score, and F1 score as Equations (1)–(3), respectively. The precision value was calculated as 94%, the recall value as 85%, and the F1 score as 89%. The F1 score was closer to 90%, indicating that our model's learning technique for detecting timing side-channel attacks in SDNs was accurate.

We compared the random forest classification technique with the tree-based methods. The same dataset and algorithm are used to label the data. The F1 score, precision, and recall were obtained from three evaluation metrics. Figure 8 demonstrates that random

forest achieved the highest evaluation score. It has an F1 score of 89%, whereas the XGBoost, decision tree (DT), and bagging classifiers (BC) have scores of 87%, 84%, and 82%, respectively. In terms of precision, random forest performed better than the others by 94%. The XGBoost, DT, and BC scored 90%, 87%, and 85%, respectively. With a score of 85% for recall, random forest was the most stable classification method for our solution, whereas for XGBoost, DT, and BC, the scores were 84%, 81%, and 80%, respectively.

The results indicate that the proposed machine learning-based technique is adequate for detecting time side-channel attacks in SDNs. Additionally, as shown in the results in Figure 8, the random forest method is the optimal classification approach for this solution. The attack probes were launched at predetermined intervals during the experiments. This guaranteed that the background activity was unique each time an attack was initiated. Advanced machine learning techniques and listener applications certify that the solution is efficient and undetectable.

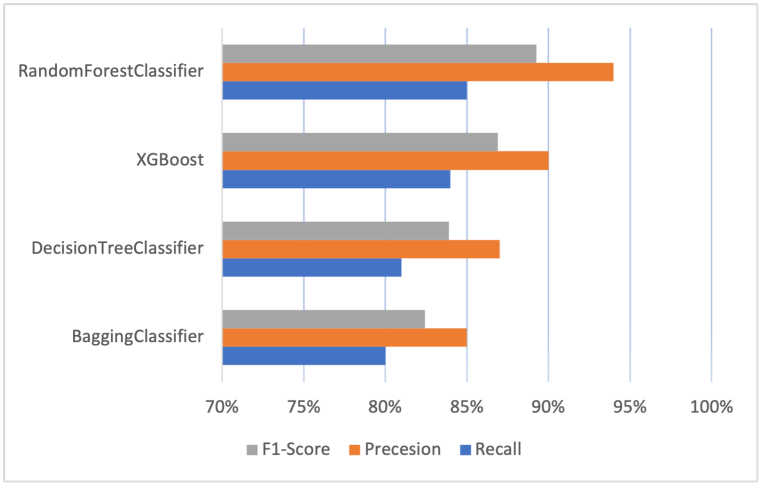

Figure 8. Performance Metrics of Machine Learning Algorithms: F1 Score, Precision, and Recall.

Next, we conducted an experiment for the response solution in two phases: phases A and B. In Phase A, we evaluated a scenario in which the data analysis module determined that the source host had not been detected previously. As this was the first probe attempt by the adversary in this experiment, the source was marked as not having been detected previously. This is sent to the response module, which generates a temporary customised rule. The response module uses the identity and activity characteristics of the source to formulate customised rules. These include the source MAC and IP addresses, destination addresses, and protocols. This customised flow is temporary, as it is set with an 'idle_timeout' value, which is a feature of the OpenFlow protocol. This means that if there is no matching attempt by the adversary for a certain amount of time, it is removed. The response module is directly connected to the controller to implement these flows.

The temporary rule of the response module is observed using the feedback module of the solution. We do this by developing a Python-based application that leverages 'OFPFlowStatsRequest', which is a feature of the OpenFlow protocol. We generated probe traffic from the same host. In this case, the traffic is blocked. This is due to the temporary customised rule. We tested the traffic to other networks and protocols to ensure that the flow applied only dropped the probe attempts. The results in Figure 9 illustrate that only the customised rule is accurate. Probe traffic using the ICMP protocol was dropped from specific destinations. Other hosts connected to the same OpenFlow switch can also reach this destination. The results demonstrate that all other traffic from that host is not impacted by this temporary customised rule.

Next, the feedback module monitors the statistics of temporary rules. The statistics show that temporary rule registers were hit. This implies that the adversary remains malicious and requires a permanent solution. In this case, the module provides feedback

to the response module to add a permanent customised flow to permanently block the adversary's probe at the switch level. We repeated these steps to simulate a scenario in which we validated the update of the database using the feedback module. In this step, we observed the OpenFlow switch entries to verify that the switch has a temporary rule from the response module. We let the idle_timeout expire and did not generate another probe from the same host. The feedback module sends the source and other parameters to the database and we observe that this process creates new entries in our database.

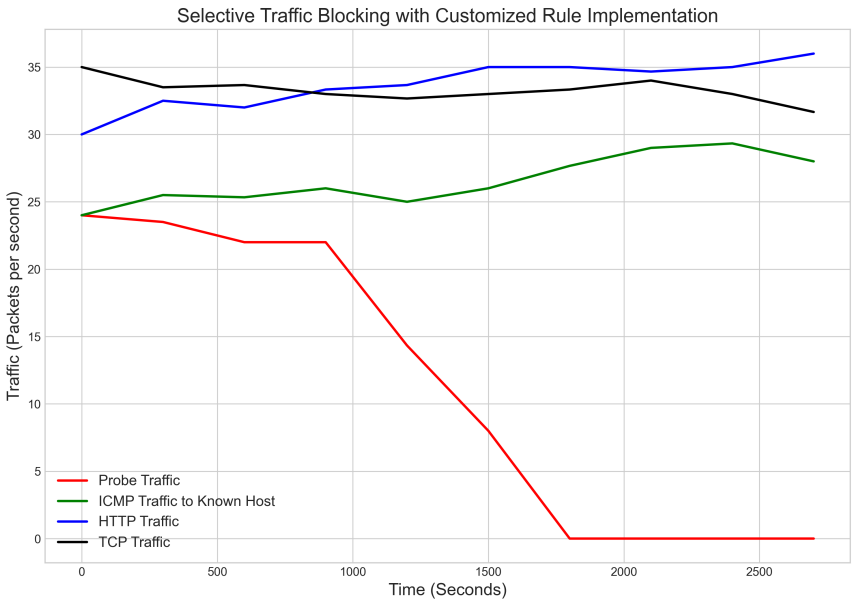

Figure 9. Selective Traffic Blocking with Customized Rule Implementation.

In Phase B, the probe was launched again from the second host. As the source has already been detected in the previous stage, the data analysis module marks it as having been detected previously. This is sent to the response module to form a permanently customised rule. The response solution forms a customised flow based on the detected source, destination, and protocol. It instructs the controller to send this flow to its switch. This rule is now a permanent rule which blocks only the specific probing traffic from the controller, instead of adding it.

The data depicted in Figure 10 suggest that the response time of the previously identified malicious machines remains relatively stable and does not change significantly. The efficiency of these requests can be attributed to their streamlined process, which includes the data analysis and response phases of the response mechanism. By reducing the processing of sources blacklisted in the past, the data analysis component enables prompt response measures. The results in Figure 10 show that, on average, only an additional 10.41 ms of processing time was added to the response time. This value is negligible because the process ensures that the flow sent to the switch is granular and does not block the entire traffic from the detected source.

To test the effectiveness of the permanent customised drop rule, we generated probe traffic from hosts *1*, *3*, *5* and *7* connected to switches *1*, *2*, *3* and *4*, respectively. Timing probes are generated from these hosts with variable destination addresses until a temporary customised drop rule is obtained. We note the destination addresses and address ranges of these probes. We stop the probes and let the 'idle_timeout' expire. This implies that the response solution has learned about the probing packets; however, there is no rule in the switches to drop these probe packets. After the 'idle_timeout' for the temporary flow rule expires, we regenerate the probes from the same sources to the same destinations which were used previously. The response solution immediately responds using a customised permanent drop rule. We evaluated the accuracy of these flows by generating ICMP, TCP,

and UDP traffic from all the hosts in the network. This traffic includes legitimate traffic from all hosts on multiple TCP and UDP ports. We integrated sFlow [53] with our solution to monitor and evaluate the traffic passing through all four switches. This allowed us to obtain a comprehensive and statistically accurate view of the network traffic, monitor the hit count for each flow in the switch, and identify potential traffic interruptions.

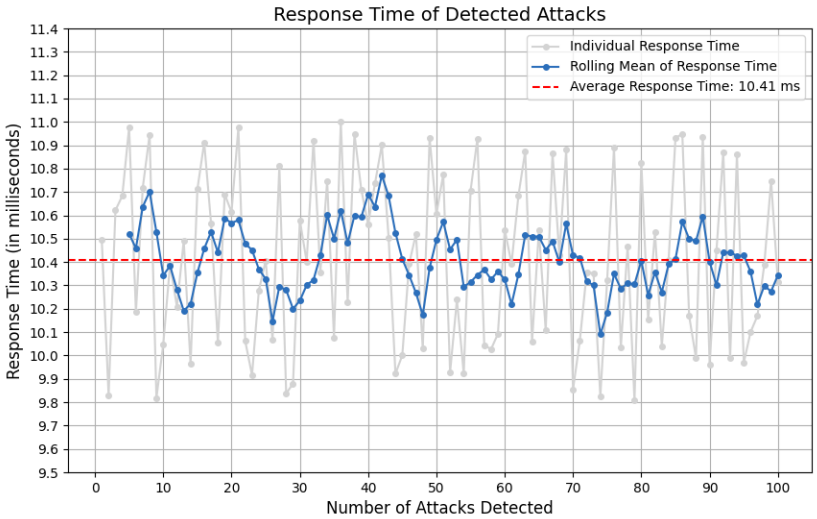

Figure 10. Response Time Trend for Previously Processed Attack Sources.

The results in Figure 11 depict the accuracy of the permanent customised rule and its impact on other host machines connected to the network. For each host that generated malicious traffic, we evaluated the accuracy by comparing the malicious probe traffic generated with the malicious probe traffic dropped. The overall accuracy of the permanent responses generated via the proposed solution is 97.68%. This shows that the permanent rule generated via the response solution does not affect the legitimate traffic from the affected hosts. This also shows that the network traffic for other hosts connected to the same SDN switch is not tempered after the addition of this flow. These statistics demonstrate that our method accurately responds to the timing side-channel attacks.

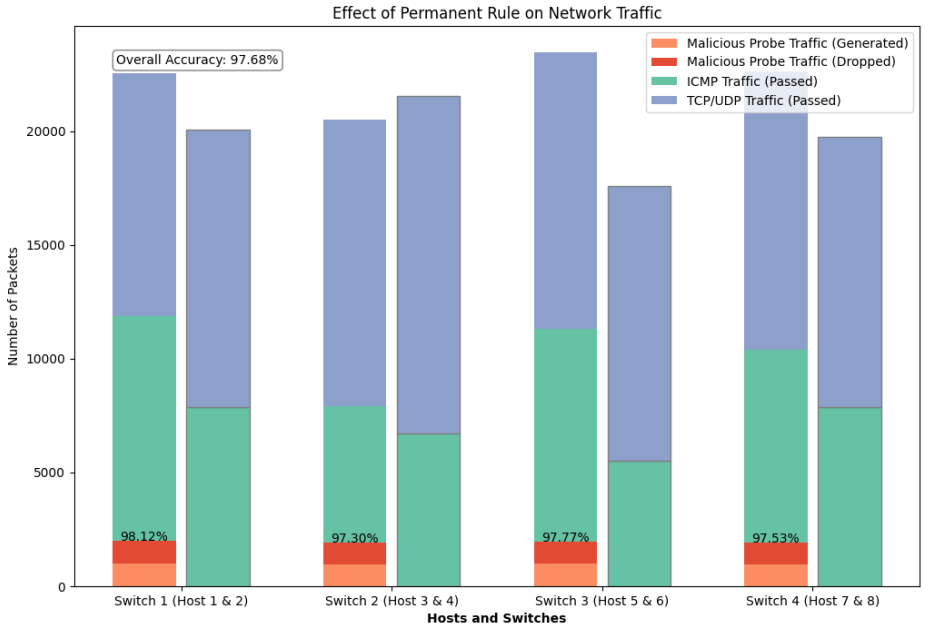

Figure 11. Accuracy of Permanent Customized Rule and Its Impact on Host Machines.

Our experimental results show that our response mechanism is effective in responding to timing side-channel attacks in SDNs. Specifically, this solution is distinguished via the successful implementation of customised flow responses designed to counter these attacks. The results demonstrated that the response time to the redetected sources was notably fast. Consequently, the implementation of a customised permanent rule in such instances has minimal effects. In this experiment, attack probes were launched at random intervals. This ensured that the background traffic was distinct each time an attack was launched.

5. Discussion

The experimental results presented in the preceding section provide convincing evidence that the proposed solution for addressing timing side-channel attacks in SDNs is reliable. The proposed solutions are designed to operate in a covert manner to avoid detection by potential adversaries. The techniques employed in this study for detecting and responding to timing side-channel attacks are innovative and provide a resilient defense against such attacks in SDNs.

5.1. Comparison with Other Work

Most current research in the field of SDN and machine learning focuses on detecting DDoS attacks with a high attack rate, as discussed in [34]. Research on low-frequency attacks is limited. Although different labelling techniques may be used, most modern machine learning techniques are primarily focused on detecting intrusions or DDoS attacks. Although most studies have focused on detecting attacks based on the volume or type of traffic [32], determining the true origin of an attacker is often challenging [37]. Most studies have focused solely on identifying IP addresses, without considering IP spoofing. Furthermore, the controller is an essential component of most existing defense mechanisms [26,28,29], leading to a significant increase in the network overhead.

The current approach [44–46] to respond to attacks prioritises speed over thoroughness and disregards historical responses or attack patterns before acting. Current response mechanisms are based on applications that use a control plane to send a drop-all flow to the switch when an attack is detected [13,45,46]. Consequently, all traffic from the identified malicious IP addresses was blocked. This affects the host's services and may allow the adversary to bring the host down. Furthermore, this drop flow response is not reviewed until a network administrator intervenes, owing to the lack of automated rule monitoring. This results in prolonged outages and turnaround times.

Our detection mechanism focuses on identifying low-rate timing side-channel attacks. This methodology is distinct as it does not require a minimum amount of traffic. Our employment of a machine learning model and labelling mechanism relies on previously unknown destination MAC addresses rather than IP addresses. This approach allows for the visualisation of layer two, thereby simplifying the process of identifying the device responsible for generating the timing probe. In contrast to alternative approaches, our detection method does not impose a requirement on the controller or OpenFlow switches to execute intensive operations. The response methodology utilised in this study incorporates customised flow entries based on destination IP and traffic type, rather than completely blocking traffic from a specific source IP or MAC address. With these customised flow entries, legitimate traffic is preserved and an effective response is delivered to counteract malicious activity. In contrast to alternative methods, our approach prioritises verification of the response prior to its permanent implementation. This was accomplished by constantly monitoring the response flow hits of the switch. Thus, the response flow is retained in the flow table only if it is invoked persistently. However, if the response flow remained inactive for a predetermined duration, it was automatically removed and the participant was expired.

5.2. Advantages

5.2.1. Compatibility with SDN Environments

This solution is compatible with all the SDN controllers and switches. This compatibility is ensured via the peripheral design, which consists of Application Layer Interconnect and Control Channel Interconnect modules. The flow channel between the controller and switch remained unaffected. Our methodology can benefit SDN environments that use hardware or software switches.

5.2.2. Validation of Attack Responses

Incorporating a validation phase for attack responses distinguishes our SDN security methodology from the conventional approaches. Before transmitting the response to the controller, the appropriateness of the analysis module is evaluated and verified. This unique approach ensures that every response to the security solution is subjected to significant scrutiny, thereby enhancing the overall credibility of the framework.

5.2.3. Reduced Impact of Attack and Downtime

Customised rule creation is a key component of the suggested response solution, which reduces the impact of an attack and the ensuing downtime. It operates by identifying and eliminating specific attack traffic, thereby ensuring that normal network traffic remains uninterrupted. This refined technique prevents a single attack from disrupting the host traffic. In contrast, conventional methods without such tailored flows tend to respond by blocking all the traffic from the observed source. This results in a complete service outage for the accused host, demonstrating the distinct advantage of our approach over the other methods.

5.2.4. Reduced Human Interference

The incorporation of temporary customised rules and a feedback mechanism into the response solution significantly decreases the need for human intervention. Instead of transmitting responses directly to the controller, the proposed solution mandates a thorough inspection to enhance the stability and accuracy of the solution.

5.3. Limitations

It can be argued that the dataset created in this experiment was unchanged and fixed. The process of collecting and storing data is fast; however, making the data usable requires time-consuming pre-processing and training stages that require substantial computing resources. As this dataset is produced once the network topology has been set up, frequent updates are not required. Nevertheless, modifying the network topology may decrease its accuracy and increase the number of false positives. To ensure optimal performance, we suggest updating the dataset when a switch is added or removed from the network, or at the conclusion of a set time interval.

The lack of a comprehensive real-attack dataset for timing attacks in SDN restricts our research. Despite our best efforts, we were unable to obtain real-world attack data for timing attacks in SDN for experimental reasons; therefore, we relied on simulated attacks instead. It is possible that these simulations did not accurately reflect real-world conditions, thereby limiting the generalisability and applicability of the results. However, the primary objective of this study is to construct and evaluate our proposed method within the constraints of existing knowledge and resources. Furthermore, the validation process of the response mechanism has the potential to introduce delays in real-time response mechanisms. In some instances, this may result in a delayed response time. The first reaction to a detected attack attempt was somewhat delayed because of the temporary rule generation. Although essential for ensuring the resilience of the solution, this delay can potentially affect the overall response time of the solution, which can lead to slower reaction times in specific situations. Another limitation of our proposed response mechanism is the granularity of the customised response rules. An adversary can potentially generate requests with different destination subnets and

protocols, which generates many customised flow entries on the switch. This could potentially overwhelm the switch flow table and deteriorate the network performance.

6. Conclusions and Future Work

The proposed solution demonstrates the ability to detect and respond to side-channel attacks in SDNs. Moreover, it possesses the capability to pinpoint the root cause of the timing probe activity. Its implementation exhibits minimal latency in an SDN environment. This study incorporated various novel aspects, such as the application of machine learning techniques, probe activity detection, and a data-driven automation approach for response. The results of this study demonstrate that the proposed techniques effectively minimise the occurrence of false positives and false negatives, thus attesting to the consistency and feasibility of these methods for detecting and responding to such attacks.

Future research could concentrate on adapting the proposed techniques to larger and more complex network architectures and evaluating their effectiveness. Furthermore, augmenting the proposed response mechanism by testing different attack detection techniques can extend the scope of this study. This mechanism was created to respond to timing side-channel attacks; however, it can also be adapted to counter other types of attacks. For instance, the mechanism may be combined with intrusion detection systems to generate customised responses for detecting network intrusion attacks. Augmentation of the response mechanism can potentially increase the adaptability and practicality of SDN security solutions, thereby supporting the development of more robust and effective techniques.

Author Contributions: Conceptualisation, F.S.; experiments, F.S.; result analysis, F.S.; paper write-up, F.S.; supervision, Y.-W.C., E.V.-G. and C.N.; critical revision Y.-W.C., E.V.-G. and C.N. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data are contained within the article.

Acknowledgments: The authors thank the School of Computing and Information Technology, University of Wollongong, Australia for technical support.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

SDN	Software-defined networking
DDoS	Distributed Denial-of-Service
TTL	Time-to-live

References

1. Maleh, Y.; Qasmaoui, Y.; El Gholami, K.; Sadqi, Y.; Mounir, S. A comprehensive survey on SDN security: Threats, mitigations, and future directions. *J. Reliab. Intell. Environ.* **2022**, *9*, 201–239. [CrossRef]
2. Scott-Hayward, S.; Natarajan, S.; Sezer, S. A survey of security in software defined networks. *IEEE Commun. Surv. Tutor.* **2015**, *18*, 623–654. [CrossRef]
3. Chica, J.C.C.; Imbachi, J.C.; Vega, J.F.B. Security in SDN: A comprehensive survey. *J. Netw. Comput. Appl.* **2020**, *159*, 102595. [CrossRef]
4. Liu, Y.; Wang, Y.; Zhang, J. New machine learning algorithm: Random forest. In Proceedings of the Information Computing and Applications: Third International Conference, ICICA 2012, Chengde, China, 14–16 September 2012; Proceedings 3; Springer: Berlin/Heidelberg, Germany, 2012; pp. 246–252.
5. Jijo, B.T.; Abdulazeez, A.M. Classification based on decision tree algorithm for machine learning. *Evaluation* **2021**, *6*, 7.
6. Chen, T.; Guestrin, C. Xgboost: A scalable tree boosting system. In Proceedings of the 22nd Acm Sigkdd International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, 13–17 August 2016; pp. 785–794.

7. Dietterich, T.G. Ensemble methods in machine learning. In Proceedings of the Multiple Classifier Systems: First International Workshop, MCS 2000, Cagliari, Italy, 21–23 June 2000; Proceedings 1; Springer: Berlin/Heidelberg, Germany, 2000; pp. 1–15.
8. Su, R.; Zhang, D.; Venkatesan, R.; Gong, Z.; Li, C.; Ding, F.; Jiang, F.; Zhu, Z. Resource allocation for network slicing in 5G telecommunication networks: A survey of principles and models. *IEEE Netw.* **2019**, *33*, 172–179. [CrossRef]
9. Ahvar, E.; Ahvar, S.; Raza, S.M.; Manuel Sanchez Vilchez, J.; Lee, G.M. Next generation of SDN in cloud-fog for 5G and beyond-enabled applications: Opportunities and challenges. *Network* **2021**, *1*, 28–49. [CrossRef]
10. Lin, B.S.P. Toward an AI-enabled SDN-based 5G & IoT network. *Netw. Commun. Technol.* **2021**, *5*, 1–7.
11. Sarica, A.K.; Angin, P. Explainable security in SDN-based IoT networks. *Sensors* **2020**, *20*, 7326. [CrossRef]
12. Ahmad, I.; Kumar, T.; Liyanage, M.; Okwuibe, J.; Ylianttila, M.; Gurtov, A. Overview of 5G security challenges and solutions. *IEEE Commun. Stand. Mag.* **2018**, *2*, 36–43. [CrossRef]
13. Martins, J.S.; Campos, M.B. A security architecture proposal for detection and response to threats in SDN networks. In Proceedings of the 2016 IEEE ANDESCON, Arequipa, Peru, 19–21 October 2016; pp. 1–4.
14. Manu, B.; Koundinya, A.K. Intrusion Tolerant Architecture for SDN Networks Through Flow Monitoring. In Proceedings of the 2017 2nd International Conference on Computational Systems and Information Technology for Sustainable Solution (CSITSS), Bengaluru, India, 21–23 December 2017; pp. 1–5.
15. Schnepf, N.; Badonnel, R.; Lahmadi, A.; Merz, S. Automated verification of security chains in software-defined networks with synaptic. In Proceedings of the 2017 IEEE Conference on Network Softwarization (NetSoft), Bologna, Italy, 3–7 July 2017; pp. 1–9.
16. Schnepf, N.; Badonnel, R.; Lahmadi, A.; Merz, S. Synaptic: A formal checker for SDN-based security policies. In Proceedings of the NOMS 2018-2018 IEEE/IFIP Network Operations and Management Symposium, Taipei, Taiwan, 23–27 April 2018; pp. 1–2.
17. Kocher, P.C. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In Proceedings of the Advances in Cryptology—CRYPTO'96: 16th Annual International Cryptology Conference, Santa Barbara, CA, USA, 18–22 August 1996; Proceedings 16; Springer: Berlin/Heidelberg, Germany, 1996; pp. 104–113.
18. Cui, H.; Karame, G.O.; Klaedtke, F.; Bifulco, R. On the fingerprinting of software-defined networks. *IEEE Trans. Inf. Forensics Secur.* **2016**, *11*, 2160–2173. [CrossRef]
19. Karimi, E.; Fei, Y.; Kaeli, D. Hardware/software obfuscation against timing side-channel attack on a GPU. In Proceedings of the 2020 IEEE International Symposium on Hardware Oriented Security and Trust (HOST), San Jose, CA, USA, 7–11 December 2020; pp. 122–131.
20. Sepulveda, M.J.; Diguet, J.P.; Strum, M.; Gogniat, G. NoC-based protection for SoC time-driven attacks. *IEEE Embed. Syst. Lett.* **2014**, *7*, 7–10. [CrossRef]
21. Dunlap, S.; Butts, J.; Lopez, J.; Rice, M.; Mullins, B. Using timing-based side channels for anomaly detection in industrial control systems. *Int. J. Crit. Infrastruct. Prot.* **2016**, *15*, 12–26. [CrossRef]
22. Shaghaghi, A.; Kaafar, M.A.; Buyya, R.; Jha, S. Software-defined network (SDN) data plane security: Issues, solutions, and future directions. In *Handbook of Computer Networks and Cyber Security: Principles and Paradigms*; Springer: Berlin/Heidelberg, Germany, 2020; pp. 341–387.
23. Shoaib, F.; Chow, Y.W.; Vlahu-Gjorgievska, E. Preventing Timing Side-Channel Attacks in Software-Defined Networks. In Proceedings of the 2021 IEEE Asia-Pacific Conference on Computer Science and Data Engineering (CSDE), Brisbane, Australia, 8–10 December 2021; pp. 1–6.
24. Yoon, C.; Lee, S.; Kang, H.; Park, T.; Shin, S.; Yegneswaran, V.; Porras, P.; Gu, G. Flow wars: Systemizing the attack surface and defenses in software-defined networks. *IEEE/ACM Trans. Netw.* **2017**, *25*, 3514–3530. [CrossRef]
25. Conti, M.; De Gaspari, F.; Mancini, L.V. A novel stealthy attack to gather SDN configuration-information. *IEEE Trans. Emerg. Top. Comput.* **2018**, *8*, 328–340. [CrossRef]
26. Zhang, M.; Li, G.; Xu, L.; Bai, J.; Xu, M.; Gu, G.; Wu, J. Control plane reflection attacks and defenses in software-defined networks. *IEEE/ACM Trans. Netw.* **2020**, *29*, 623–636. [CrossRef]
27. Liu, S.; Reiter, M.K.; Sekar, V. Flow reconnaissance via timing attacks on SDN switches. In Proceedings of the 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS), Atlanta, GA, USA, 5–8 June 2017; pp. 196–206.
28. Hou, J.; Zhang, M.; Zhang, Z.; Shi, W.; Qin, B.; Liang, B. On the fine-grained fingerprinting threat to software-defined networks. *Future Gener. Comput. Syst.* **2020**, *107*, 485–497. [CrossRef]
29. Arsalan, A.; Rehman, R.A. Prevention of timing attack in software defined named data network with VANETs. In Proceedings of the 2018 International Conference on Frontiers of Information Technology (FIT), Islamabad, Pakistan, 17–19 December 2018; pp. 247–252.
30. Liu, A.; Chen, J.X.; Wechsler, H. Real-time timing channel detection in an software-defined networking virtual environment. *Intell. Inf. Manag.* **2015**, *7*, 283. [CrossRef]
31. Sahu, K.; Kshirsagar, R.; Vasudeva, S.; Alzahrani, T.; Karimian, N. Leveraging Timing Side-Channel Information and Machine Learning for IoT Security. In Proceedings of the 2021 IEEE International Conference on Consumer Electronics (ICCE), Las Vegas, NV, USA, 10–12 January 2021; pp. 1–6.
32. Amin, R.; Rojas, E.; Aqdus, A.; Ramzan, S.; Casillas-Perez, D.; Arco, J.M. A survey on machine learning techniques for routing optimization in SDN. *IEEE Access* **2021**, *9*, 104582–104611. [CrossRef]
33. Wijesekara, P.A.D.S.N.; Gunawardena, S. A Machine Learning-Aided Network Contention-Aware Link Lifetime-and Delay-Based Hybrid Routing Framework for Software-Defined Vehicular Networks. *Telecom* **2023**, *4*, 393–458. [CrossRef]

34. Ahmed, M.; Islam, A.; Shatabda, S.; Islam, A.K.M.M.; Robin, T.I. Intrusion detection system in software-defined networks using machine learning and deep learning techniques—A comprehensive survey. *TechRxiv Prepr.* **2021**. [CrossRef]
35. Alzahrani, A.O.; Alenazi, M.J. Designing a network intrusion detection system based on machine learning for software defined networks. *Future Internet* **2021**, *13*, 111. [CrossRef]
36. Klymash, M.; Shpur, O.; Peleh, N.; Maksysko, O. Concept of Intelligent Detection of DDoS Attacks in SDN Networks Using Machine Learning. In Proceedings of the 2020 IEEE International Conference on Problems of Infocommunications. Science and Technology (PIC S&T), Kharkiv, Ukraine, 6–9 October 2020; pp. 609–612.
37. Ahmad, A.; Harjula, E.; Ylianttila, M.; Ahmad, I. Evaluation of machine learning techniques for security in SDN. In Proceedings of the 2020 IEEE Globecom Workshops (GC Wkshps), Taipei, Taiwan, 7–11 December 2020; pp. 1–6.
38. Aslam, M.; Ye, D.; Hanif, M.; Asad, M. Machine learning based SDN-enabled distributed denial-of-services attacks detection and mitigation system for Internet of Things. In Proceedings of the Machine Learning for Cyber Security: Third International Conference, ML4CS 2020, Guangzhou, China, 8–10 October 2020; Proceedings, Part I 3; Springer: Berlin/Heidelberg, Germany, 2020; pp. 180–194.
39. Abou El Houda, Z.; Hafid, A.S.; Khoukhi, L. A novel machine learning framework for advanced attack detection using sdn. In Proceedings of the 2021 IEEE Global Communications Conference (GLOBECOM), Madrid, Spain, 7–11 December 2021; pp. 1–6.
40. Wijesekara, P.A.D.S.N.; Gunawardena, S. A Comprehensive Survey on Knowledge-Defined Networking. *Telecom* **2023**, *4*, 477–596. [CrossRef]
41. Banton, M.D. *A Deep Learning-Based Approach to Identifying and Mitigating Network Attacks within SDN Environments Using Non-Standard Data Sources*; Liverpool John Moores University: Liverpool, UK, 2021.
42. Varadharajan, V.; Tupakula, U. Counteracting attacks from malicious end hosts in software defined networks. *IEEE Trans. Netw. Serv. Manag.* **2019**, *17*, 160–174. [CrossRef]
43. Aladaileh, M.A.; Anbar, M.; Hasbullah, I.H.; Sanjalawe, Y.K. Information theory-based approaches to detect DDoS attacks on software-defined networking controller a review. *Int. J. Educ. Inf. Technol* **2021**, *15*, 83–94. [CrossRef]
44. Sonchack, J.; Dubey, A.; Aviv, A.J.; Smith, J.M.; Keller, E. Timing-based reconnaissance and defense in software-defined networks. In Proceedings of the 32nd Annual Conference on Computer Security Applications, Los Angeles, CA, USA, 5–9 December 2016; pp. 89–100.
45. Krishnan, P.; Duttagupta, S.; Achuthan, K. SDN/NFV security framework for fog-to-things computing infrastructure. *Softw. Pract. Exp.* **2020**, *50*, 757–800. [CrossRef]
46. Eom, T.; Hong, J.B.; An, S.; Park, J.S.; Kim, D.S. A framework for real-time intrusion response in software defined networking using precomputed graphical security models. *Secur. Commun. Netw.* **2020**, *2020*, 7235043. [CrossRef]
47. Shoaib, F.; Chow, Y.W.; Vlahu-Gjorgievska, E.; Nguyen, C. Using Machine Learning for Detecting Timing Side-Channel Attacks in SDN. In Proceedings of the International Symposium on Mobile Internet Security, Jeju, Republic of Korea, 15–17 December 2022; Springer: Berlin/Heidelberg, Germany, 2022; pp. 180–194.
48. Subhajournal. SDN Intrusion Detection. 2022. Available online: https://www.kaggle.com/datasets/subhajournal/sdn-intrusion-detection (accessed on 14 August 2023).
49. Market Research Future. Software-Defined Networking (SDN) Market Size, Share | 2030—marketresearchfuture.com. Available online: https://www.marketresearchfuture.com/reports/software-defined-networking-market-1607 (accessed on 8 July 2023).
50. Das, T.; Hamdan, O.A.; Shukla, R.M.; Sengupta, S.; Arslan, E. UNR-IDD: Intrusion Detection Dataset using Network Port Statistics. In Proceedings of the 2023 IEEE 20th Consumer Communications & Networking Conference (CCNC), Las Vegas, NV, USA, 8–11 January 2023; pp. 497–500.
51. Dhanabal, L.; Shantharajah, S. A study on NSL-KDD dataset for intrusion detection system based on classification algorithms. *Int. J. Adv. Res. Comput. Commun. Eng.* **2015**, *4*, 446–452.
52. Banker, K.; Garrett, D.; Bakkum, P.; Verch, S. *MongoDB in Action: Covers MongoDB Version 3.0*; Simon and Schuster: New York, NY, USA, 2016.
53. Ujjan, R.M.A.; Pervez, Z.; Dahal, K.; Bashir, A.K.; Mumtaz, R.; González, J. Towards sFlow and adaptive polling sampling for deep learning based DDoS detection in SDN. *Future Gener. Comput. Syst.* **2020**, *111*, 763–779. [CrossRef]
54. Kaur, K.; Singh, J.; Ghumman, N.S. Mininet as software defined networking testing platform. In Proceedings of the International Conference on Communication, Computing & Systems (ICCCS), Pubjab, India, 8–9 August 2014; pp. 139–142.
55. Bhardwaj, S.; Panda, S.N. Performance evaluation using ryu sdn controller in software-defined networking environment. *Wirel. Pers. Commun.* **2022**, *122*, 701–723. [CrossRef]
56. Adeleke, O.A.; Bastin, N.; Gurkan, D. Network traffic generation: A survey and methodology. *ACM Comput. Surv. (CSUR)* **2022**, *55*, 1–23. [CrossRef]
57. Ibrahim, H.Y.; Ismael, P.M.; Albabawat, A.A.; Al-Khalil, A.B. A secure mechanism to prevent ARP spoofing and ARP broadcasting in SDN. In Proceedings of the 2020 International Conference on Computer Science and Software Engineering (CSASE), Duhok, Iraq, 16–18 April 2020; pp. 13–19.