Information Sciences Letters
An International Journal

http://dx.doi.org/10.18576/isl/130305

Cost-Efficient Method for Detecting and Mitigating the CrossPath Attack via Shared Links in SDN-Based IoT Network

A. Allakany[1,*] *and S. A. Nooh*[2]

[1]Computer Science Department, Faculty of Computers and Information, Kafrelsheikh University, Kafrelsheikh 33516, Egypt
[2]Information Systems Department, Faculty of Computing and Information Technology, King Abdulaziz University, Jeddah, Saudi Arabia

Received: 2 Jan. 2024, Revised: 17 Jan. 2024 Accepted: 21 Jan. 2024.
Published online: 1 May 2024

Abstract: The security of the Internet of Things (IoT) ecosystem has become a critical challenge due to a tremendous increase in the vulnerable connected IoT devices. Software-Defined Network (SDN) becomes a choice for managing IoT and offers new approaches to solve security problems. In the recent wave of distributed denial-of-service (DDoS) attacks, attackers have shifted their strategy from directly targeting the SDN controller to concentrating on specific links or area, causing disruptions in connectivity. These attacks, known as Linking Flooding Attack (LFA) and CrossPath Attack (CPA), represent a novel form of DDoS attack. Unlike conventional approaches in the literature that solely rely on automatic Machine Learning (ML) model to detect and mitigate DDoS attacks family. In this paper, we introduce a new SDN-based strategy to combat DDoS, LFA, and CPA attacks. This approach includes step-by-step network measurements to detect and pinpoint unusual link behavior, facilitating the prompt identification of potential attacks. Following this, an ML model is applied to verify if these link congestions are indicative of an attack. This method marks a departure from traditional techniques that depend entirely on automated ML models. Instead, it starts with an analysis of each link's congestion before employing the ML model for attack confirmation. This strategy ensures more efficient use of SDN controller resources. Our implementation of this system as an application in the Ryu controller's application layer has shown promising results. Through our evaluations, we found that the approach notably improved link performance assessment and detection of DDoS, LFA, and CPA attacks. This advancement resulted in a more efficient use of SDN controller resources, ultimately enhancing the security of IoT networks.

Keywords: Software-defined networks (SDN); Cross-Path attack (CPA); Link-flooding attack (LFA); DDOS attack; Machine Learning (ML).

1 Introduction

Software-defined networks (SDN) emerging as one of the most promising new era network technologies with its centralized and easily programmable nature. SDN transforms network devices into fundamental packet forwarding elements by decoupling the control plane from the data plane. These devices are subsequently configured by a centrally located unit referred to as the SDN controller, making use of standardized protocols such as OpenFlow [1, 2]. With the introduction of this separation between the control plane and data plane, the SDN architecture empowers network management with advanced control capabilities, ultimately simplifying the creation of intelligent and automated networks [3]. Furthermore, this separation fosters greater flexibility, making it more convenient to implement innovative network applications and services [2].

Although widely adopted, SDN-based networks persistently face vulnerabilities due to diverse network intrusions and are confronted with the ever-changing realm of security threats and attacks [4]. Among these threats are CPA and LFA, representing a novel category of DDoS attacks that can affect the robustness and dependability of SDN networks. In the SDN paradigm, the development of attack detection and mitigation techniques is made more manageable due to SDN's ability to provide a comprehensive network view, effortless collection of flow statistics, and support for dynamic flow entry updates [5]. In traditional networks, accomplishing this task becomes significantly more challenging as devices need to exchange substantial amounts of information with the other parts of the network partially. As a result, this leads to restricted visibility and, consequently, limited detection capabilities.

In DDoS attacks, the typical approach involves utilizing multiple compromised machines to flood the target machine or SDN controller with an enormous amount of traffic. This overwhelming influx of resource demands renders the targeted device incapable of performing its intended functions. LFA attacks can significantly disrupt connectivity for a specific

*Corresponding author e-mail: alaa.ellakani@kfs.edu.eg

link or area without detection since they target specific links rather than a specific target machine or SDN controller [6]. This makes them more difficult to detect than DDoS attacks. Similar to LFA, CPA disrupts SDN control channel links by taking advantage of shared links between control and data traffic paths [7]. Rather than sending an extensive volume of control traffic directly to the controller, it generates precisely designed data traffic on shared links, implicitly interfering with the control traffic delivery. Consequently, real time control messages exchanged between the SDN controller, and the switches experience significant delays or are dropped, as the data traffic fails to reach the controller. Thus, the timely detection and prevention of CPA attacks represents critical concerns for SDN network.

The core motivation behind this research lies in addressing a primary challenge of SDN, which involves the side effects resulting from shared links between control and data traffic paths. This sharing exposes a vulnerability that attackers can exploit to disrupt the control channel, using malicious data traffic to execute CPA. Considering the control channel's responsibility for granting centralized control to the controller over each network switch, it becomes relatively easy for an attacker to compromise all network functions, especially due to the abundance of SDN applications running on the controller. To prevent CPA and ensure SDN network security, it is vital to monitor and measure the delay in shared links. If the delay increases, it could indicate congestion on the link, signaling a potential CPA targeting that specific connection. E2E delay measurement provides an overall perspective of path delay but does not offer details about delays in individual links within the path. While some solutions can measure the E2E delay of the path, they lack the capability to pinpoint the bottleneck link (congested link) along that path. As CPA leads to link congestion and does not target all links in the path, a more effective approach to detect CPA is by measuring the delay of each individual link in the network instead of measuring the E2E delay of the path. Researchers have suggested various methods for detecting DDoS attacks, LFA, and CPA on SDN networks [7-12]. Although these methods have demonstrated efficiency, detecting CPA attacks on SDN networks still poses a significant challenge. Most of these approaches involve adding more functionality to the SDN switch or creating additional overhead on both the controller and the network.

Furthermore, as a common practice in most ML detection methods, flow entries are collected from OpenFlow switches at predetermined time intervals established by the controller. The exact definition of this interval carries significant importance. If data collection occurs infrequently, it will lead to delayed attack detection, and as a result, a longer time for potential mitigation. In contrast, a too short data collection time interval results in heightened overhead for the detection mechanism. For example, in [13], the authors implemented a 5-second interval to acquire a more comprehensive understanding of network traffic.

To overcome this challenge, we propose in this paper an approach to detect and mitigate CPA in SDN networks. This approach takes advantage of the fundamental features of SDN to detect and address CPA. This approach has been implemented as an application within the Ryu controller. The implementation comprises three modules, which include a link congestion monitoring module, an ML-based CPA detection module, and a mitigation module. At regular intervals (with a polling frequency of every 5 seconds), the link congestion monitoring module analyzes link delays to pinpoint congested links within the network. When congested links are detected, the CPA detection module is executed to identify the root cause, distinguishing between CPA and legitimate traffic. Implementing link congestion detection provides a notable advantage, allowing the ML detection module to adapt its polling interval behavior or gather flow statistics over a longer time period (in our case, 15 seconds). This interval surpasses that of other solutions, typically set at every 5 seconds. Consequently, the cost of collecting traffic in our approach is reduced. The criteria for making these adjustments depend on recognizing congested links, which frequently indicate an ongoing network attack. This strategic adaptation helps reduce overhead within the detection mechanism. When the attack is detected, the mitigation module is activated to counteract the attack. This paper's contributions are out-lined below:

- The novelty of this paper lies in the introduction of an SDN detection system for online, real-time detection of CPA, as well as LFA, and DDoS attacks. The system utilizes a probing technique for efficient congestion measurement in each link and triggers an ML model when congestion surpasses a predefined threshold.
- An SDN cost-efficient module, based on machine learning, is utilized to automatically detect and mitigate attacks.

The remainder of this paper is organized as follows: The background and related work presented in Section 2, and Section 3 respectively, we give an overview of proposed system architecture in section 4. Sections 5 illustrate the Implementation of the proposed approach for detecting and mitigating CPA attack. Section 6 presents the evaluation. Section 7, present conclusion, and future work.

2 Background

In this section, we present a summary of the SDN architecture, explore the function of SDN controllers and the OpenFlow protocol, and highlight the significance of ML-based approaches in SDN networks. Additionally, we examine various attacks, such as DDoS, LFA, and CPA.

2.1 Software-Defined Networking

SDN introduces a network architecture that fundamentally transforms the conception and management of modern network layouts. In this paradigm, all networking devices are centrally controlled, managed, and configured through a single controller [15]. Figure 1 provides a visual representation of the SDN architecture, commonly illustrated using three layers: forwarding or infrastructure, control, and applications.

The SDN controller, which is logically centralized, undertakes functions like analyzing, managing, and gathering statistics from incoming network traffic packets. Positioned as the central point, the controller bears the responsibility of guiding all network packets and making decisions informed by the accumulated data. Interactions with switches are enabled via southbound application APIs, like OpenFlow, whereas northbound APIs are utilized by SDN applications such as load balancers and firewalls. Several SDN controllers are available in the market, including Trema, Ryu, NVP, Floodlight, and BNC. Open-Flow enables the establishment of a communication protocol between the control and forwarding planes of SDN networks [15].

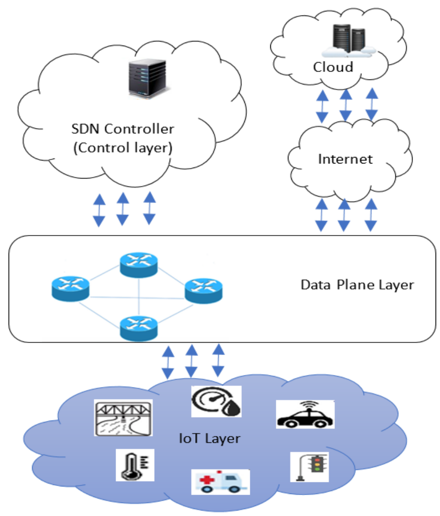

Fig. 1: SDN-Enabled IoT Networks Architecture.

SDN frequently achieves packet control through a flow mechanism centered on OpenFlow switches. This approach involves employing flow tables that house specific flow rules. The task of installing these rules into the flow table is overseen by the controller, which takes into account both network logic and an abstract representation of the network. When a new packet arrives at the OpenFlow switch, it triggers a search for pertinent rules within the flow table. If a matching rule is discovered, the switch proceeds to process the packet in alignment with the corresponding flow rule. However, in the absence of a match, the switch forwards the packet to the controller for further necessary actions. The controller assesses whether the packet should be forwarded or discarded, and then proceeds to include the flow rule into the flow table of the relevant switch.

2.2 DDOS Attack

DDoS attacks are orchestrated by networks of compromised zombie hosts with the aim of rendering target machines or network resources unavailable to their intended users. These attacks have become increasingly complex due to several distinctive characteristics. In contrast to benign traffic, traffic subjected to DDoS attacks exhibits discernible patterns, including higher volume, often surpassing that of normal traffic, brief duration, with abnormal flows being short lived, and increased port variability. In the case of DDoS attacks, the growth rate of various ports exceeds that observed in typical scenarios. Additionally, to compound the damage, multiple attack flows targeting the same target often occur simultaneously. Therefore, it is crucial to anticipate and proactively mitigate these traffic attacks [11,12].

One of the most effective solutions for the early detection of DDoS attacks involves utilizing traffic engineering to identify vulnerable links that, if attacked, would result in similar outcomes to rendering the target machine unavailable.

2.3 Link-flooding attack

The LFA has recently garnered attention as a significant focus, presenting a novel classification of DDoS attacks [6]. This attack methodology revolves around identifying target links and comprises a set of three initial steps: initially, utilizing core links within the network as the chosen target links; subsequently, detecting devices (bots) with the capacity to generate traffic that moves through these target links; and finally, overwhelming the designated target links/area with legitimate traffic to trigger an overload. In contrast to standard DDoS attacks, LFA displays various notable attributes. Firstly, the targeted entity differs. When it comes to traditional DDoS attacks, attackers release substantial and unexpected traffic directly towards the target server. In contrast, in LFA attackers strategically mobilize a multitude of bots to generate low speed traffic directed at decoy servers, all with the intention of overwhelming crucial links that link a target area to the wider expanse of the Internet. Hence, the target remains unaware of the ongoing attack. Secondly, in contrast to conventional DDoS attacks that usually result in the unavailability of a single server, LFA disables all servers within the designated area, thereby blocking their internet access. Thirdly, DDoS defense systems aim to eradicate undesirable network traffic, whereas the bots employed in LFA generate legitimate and intended data streams. As a result, traditional DDoS defense mechanisms face challenges in effectively distinguishing between attack traffic and normal network activity.

2.4 Cross-Path attack

This section highlights the impact of CPA on disrupting the SDN control channel. What is CPA? How does it differ from LFA and DDoS attacks? Additionally, a method is introduced to accurately identify a target path containing shared links, i.e., both data traffic and control traffic links.

The objective of the CPA is to disrupt the control channel of SDN by exploiting the shared links that connect paths for both control and data traffic. This is achieved by generating data traffic that traverses these shared links, causing interference with the transmission of control traffic. As a result, there are delays or drops in the delivery of real-time control messages through the control channel. Since the SDN controller exerts centralized control over switches through the control channel, an attacker can potentially dismantle nearly all network functionalities facilitated by SDN through the implementation of this attack. To successfully execute this strategy, the attacker needs to first locate the intersections between data traffic and control traffic paths. Once this information is gathered, the attacker can proceed to dispatch disruptive traffic onto these shared paths, effectively disrupting the normal operation of the control channel.

Figure 2 shows an example to explain CPA. In this network, there are ten switches. Let's assume that an attacker has compromised a host at switch S8 and can send crafted LDoS traffic to two hosts at switch S6. Since the control link between S5 and S8 is shared with data links in the attack path, the attack will impact switches S7, S8, S9, and S10. Control messages exchanged between these switches and the SDN controller may experience significant delays or be dropped, resulting in abnormal network behavior.

In this work, our primary focus is not on how the attacker can discover shared links; rather, our sole concentration is on detecting CPA within the SDN network. Nevertheless, to enhance reader comprehension, we provide an explanation of the method by which the attacker can identify these shared links.

Find attack path: For a successful attack launch, the attacker needs to accurately select a path with shared links as the target. However, identifying such target paths within SDN presents a significant challenge. Within the context of SDN, the allocation of links for control traffic is limited in number. This leads to only a finite set of data paths incorporating the links shared with control routes. In order to discover these data paths, the attacker requires knowledge of the network topology and routing information. However, these critical details are securely stored within the SDN controller, rendering them inaccessible and invisible to the attacker.

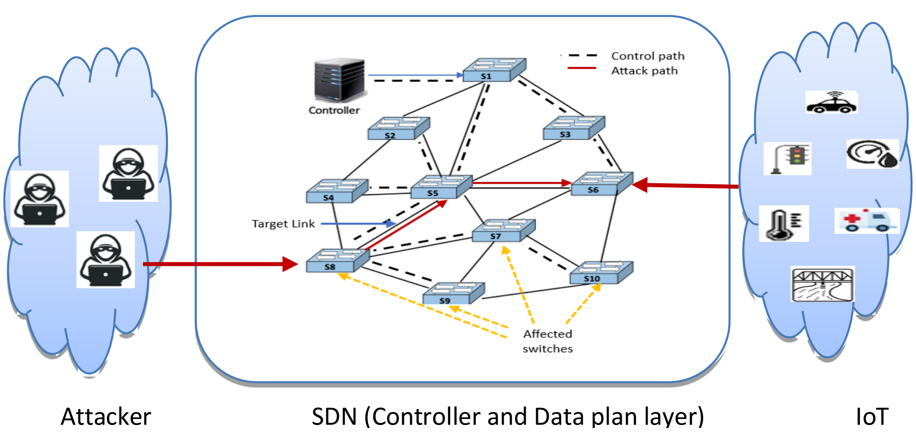

Fig. 2: Cross-Path attack example.

To address the challenges mentioned above, Xie et al. [7] proposed a technique aimed at identifying target data paths that share links with control paths. The technique is inspired by a pivotal observation: the delay of a control path rises when a short-term burst of data traffic passes through the shared links. Consequently, an attacker can exploit a host within the SDN to precisely determine critical data paths. This is accomplished by generating data traffic and monitoring the resulting fluctuations in delay along the control paths. If significant variations in delay are detected, it indicates that the data path being investigated intersects with specific control paths. On the other hand, the absence of noticeable variations provides no evidence of the data path crossing any control paths. The authors of this paper leverage the benefits of SDN, where packets that cannot find a match in a switch follow extended forwarding paths, leading to increased delays. This happens because such packets are directed to the controller to request flow rules. By analyzing the delays experienced by these packets, it becomes feasible to estimate the delays in control paths that have shared links with data paths.

3 Related Works

This section reviews the existing approaches for detecting DDoS, LFA, and CPA attacks in SDN networks.

Khashab et al. [13] introduced a technique for detecting and mitigating the impact of DDoS attacks in SDN through an analysis of network traffic attributes. They applied a range of six algorithms, encompassing Random Forest (RF), Logistic Regression, Naïve Bayes, K-Nearest Neighbors (K-NN), Support Vector Machine (SVM), and Decision Trees (DTs). The outcomes of their approach highlighted RF as the superior performer among these algorithms, rendering it the optimal choice for their methodology. Nevertheless, the execution of this proposed approach within the SDN controller leads to heightened resource utilization during instances of DDoS attacks. Sudar et al. [8] conducted a research study that explored the utilization of SVM and DTs for detecting DDoS attacks in SDN networks. They assessed the effectiveness of their proposed method using the KDD CUP dataset. However, their approach yielded unsatisfactory results, with DTs achieving an accuracy rate of only 78%. In their study, Santos et al. [9] utilized the Multiple Layer Perceptron (MLP), RF, SVM, and DTs to classify different types of DDoS attacks, including point attacks, flow table switch attacks, SDN controller attacks, and bandwidth attacks. The effectiveness of their approach was evaluated using a realistic dataset. Nevertheless, the results indicated suboptimal performance for both MLP and SVM in accurately identifying controller attacks, with an accuracy rate of only 90%. Celesova et al. [10] introduced a technique utilizing a Deep Neural Network (DNN) to enhance the security of data planes and counteract DDoS attacks within SDN networks. However, they employed the UNSW-NB15 dataset, which lacks specific adaptation to the SDN network context, for the purpose of training, testing, and evaluating their proposed system. This led to the method displaying unsatisfactory performance when evaluated in terms of computational metrics. On the contrary, Deepa et al. [11] presented a hybrid algorithm that merges SVM and Self-Organizing Map (SOM) to identify DDoS attacks within SDN networks. This hybrid model accomplished a detection rate of 90.45%, a precise detection rate of 96.77%, and a false alarm rate of 0.032%. Nevertheless, the suggested method exhibited diminished accuracy and detection rate performance.

Zhang et al. [12] put forward a novel approach known as the spatiotemporal heterogeneous bandwidth allocation (STBA) mechanism, which aims to mitigate DDoS attacks by facilitating domain-level resource management. The study conducted by Wang et al. [6] introduced LFADefender, a system based on Software-Defined Networking (SDN) with a primary objective of effectively countering link flooding attacks. This proposed approach incorporates a diverse range of indispensable technologies, with a particular emphasis placed on the critical components of congestion link detection and rerouting. By integrating these essential elements into its functionality, LFADefender demonstrates its capability to identify and mitigate the impact of link flooding attacks, ensuring the integrity and smooth operation of the network infrastructure. Zheng et al. [16] proposed a system designed to detect anomalous flows by comparing the flow rates of statistical changes with those of aggregated flows traversing vulnerable links. As a subsequent step, the system takes measures to impede the traffic by utilizing a blacklist. In order to prevent the inadvertent exclusion of legitimate traffic, the system employs a technique known as maxmin fairness packet dropping. This approach treats packets that closely resemble characteristics of the attack traffic with suspicion, categorizing them as part of the attack traffic only when their quantity exceeds the throttling threshold established for each OpenFlow port. In their work, Allakany et al. [14] present an innovative IoT network framework built on SDN, specifically designed to address LFA mitigation. The framework's architecture seamlessly integrates an SDN controller with SDN switches, which, in turn, are inter-connected with IoT gateways. Additionally, their study puts forth a proactive approach that employs hop-by-hop network measurement to identify abnormal link patterns associated with LFA. This strategy is further bolstered by the implementation of centralized traffic engineering, aiming to resolve potential link congestion and mitigate the effects of LFA. However, it's worth noting that the paper does not propose a solution to definitively verify whether the identified congested link is indeed attributed to LFA. Rezazad, et al. [17] presents a novel approach to detect a specific type of DDoS attack called the Crossfire attack. The paper discusses the challenges in detecting this attack and presents analytical and emulated results that demonstrate vulnerabilities in the execution of the attack. Furthermore, the paper proposes the use of supervised machine learning approaches, such as SVM and RF, for the classification of network traffic into normal and abnormal traffic categories, i.e., attack traffic. The machine learning models have been trained in various scenarios using link volume as the main feature set. In a recent study, Xie, et al. [7] presented a novel CrossPath attack method for disrupting the control channel of SDN by capitalizing on shared links within paths of both control and data traffic. This innovative attack strategy involves manipulating data traffic to indirectly impede the forwarding of control traffic through shared links. Notably, the stealthy nature of this technique enables it to evade easy detection by SDN controllers, as the manipulated data traffic avoids direct entry into the control channel. To facilitate the identification of susceptible paths for this type of attack, a novel approach called adversarial path reconnaissance was developed. The feasibility and effectiveness of this approach were established through a combination of theoretical analysis and empirical validation. This research contributes to the understanding of potential vulnerabilities in SDN control channels, shedding light on the implications of exploiting shared links in disrupting network operations.

The above-mentioned approaches offer effective solutions for detecting and mitigating DDoS, LFA, and CPA attacks. However, these approaches have different limitations. For instance, some of them require modifications to OpenFlow switches, which add more cost and hinder widespread deployment. Other approaches introduce overhead to both the controller and network devices in order to detect the localization of bottleneck links within the network. Lastly, the majority of research on these topics employs ML to detect and mitigate attacks, without adequately considering link delay as a crucial parameter that indicates link congestion, a potential sign of a link under attack. Several ML-driven detection approaches increase network and controller overhead, resulting in the collection of flow entries from OpenFlow switches at predetermined periodic intervals by the controller. To achieve fast detection, they assign a collection time interval that is too short, which leads to an increase in overhead. The limitations of the aforementioned approaches are addressed in this paper. A probing technique is utilized to measure link congestion with minimal cost. Subsequently, an ML model is employed if the congestion surpasses a specified threshold.

4 System Overview

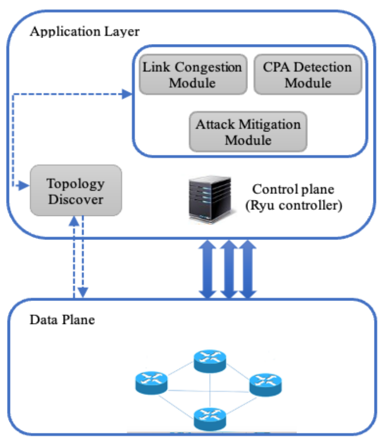

Fig. 3: Overview of SDN framework for detection and mitigation of CPA.

In this section, we introduce a comprehensive SDN framework that effectively leverages the inherent capabilities of SDN. This encompassing framework incorporates several key features, namely a global network view, the ability for dynamic reconfiguration, and advanced flow traceability for enhanced network management, to effectively identify and mitigate the CPA. This approach is implemented as an application within the Ryu controller, ensuring seamless integration into the existing SDN infrastructure. Three modules have been developed to address different aspects of the research: the Link Congestion Module, the Detection of CPA Module, and the mitigation Module. Figure 3 shows the architecture of the proposed framework.

Firstly, the Link Congestion Monitor Module runs periodically every 5 seconds to identify congested links in the network by measuring the delay of each link. Secondly, the CPA Detecting Module runs at intervals of 15 seconds. However, if congested links are detected, this module will be executed even if before the 15-second. This module determines whether the congestion in the links is due to CPA or legitimate traffic by applying an ML algorithm. Finally, the Mitigation Module comes into play upon detecting a CPA, working to mitigate the attack by blocking its source. Our proposed solution involves measuring the delay of each link as an indicator of normal or abnormal link. This enables us to efficiently apply an ML module at longer time intervals for optimizing the collection of flow statistics in the SDN network.

5 System Implementation

In this section, we introduce the implementation of the proposed SDN framework for the detection and mitigation of CPA. The details of the threat model, link congestion module, CPA detection module, and attack mitigation module are presented in the following sub-sections. The interactions between these modules are shown in Figure 4, and algorithm details in Figure 5. First, the link congestion detection module measures the delay of each individual link, it sends a probing packet of size 24 byte [18], every 5 seconds and calculates the delay [14]. If link congestion is detected, the ML model will be executed. In the CPA detection module, the flow statistics are collected, then characteristics values extracted and ML module test if there are attacks or not. When no link congestion is detected, the CPA detection module will gather flow statistics every 15 seconds. Based on OpenFlow specifications, the flow statistics collection consumes 122 bytes for the request message, while the reply message consumes 174 bytes for each flow. Finally, if the CPA is detected, then the mitigation module blocks the source of the attack. The cost efficiency of this proposed system will become evident through the packet sizes in both the delay detection module and the CPA detection module, taking into account the intervals at which these packets are sent. This will be demonstrated in the evaluation section.

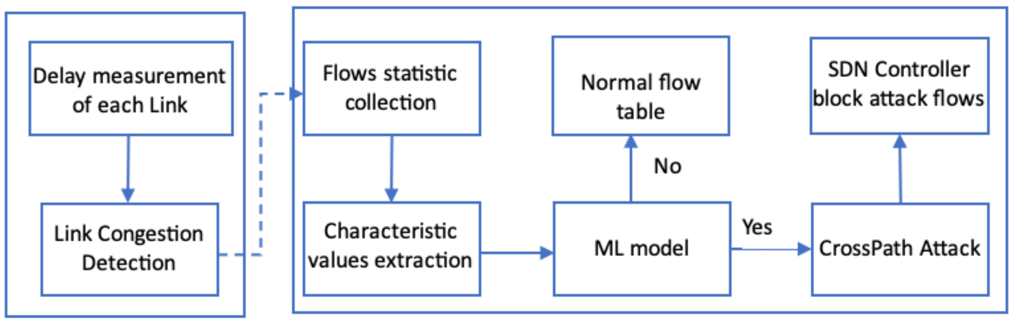

Fig. 4: CPA attack detection and mitigation process.

5.1 Threat Model

We operate under the assumption that an attacker can gain control over at least one network connected host. In the context of the scenario depicted in Figure 2, the attacker has indeed successfully compromised a host connected to switch S8. The specific control channel we are concerned with is the one connecting S8 and S5, while the hosts vulnerable to this targeted attack are linked to switch S6. The primary goal of the attacker is to manipulate data traffic in order to disrupt the SDN control channel responsible for transmitting control traffic. Compromising controllers, applications, or switches is not a prerequisite for the CPA to manipulate control messages. Additionally, we assume that controllers, switches, and applications are adequately safeguarded.

Algorithm 1: CPA Defense Algorithm

Input:
$Net_{Toplogy}$: Network topology
$Link_{Shared\_List}$: List of shared links in $Net_{Toplogy}$,
$Link_{List}$: All individual links in $Net_{Toplogy}$,
T_1 = interval_time for flow collection in the *ML* module
T_2 = interval_time for calculating congestion on network links
every T_1 **do**
collect flows statistics
every T_2 **do**
calculate link congestion /* $T_2 < T_1$ */
If *Link_congestion* detected **do**
Collect flow statistics and run *ML* module.
If *ML* detected the attack **then**

```
Find and block the flows caused that attack.
else
Normal flow table
EndIf
EndIf
```

Fig. 5: Attacks defense algorithm.

5.2 Link congestion detection module

In the scope of SDN, monitoring techniques can be broadly grouped into two categories: active, which involve periodically sending probing packets, and passive, which entail collecting traffic statistics from network forwarding components. Recently, several methods have been proposed that make use of ML algorithms to aggregate traffic statistics obtained from SDN controllers in order to identify different attack types. Nevertheless, there are other traffic indicators, such as link delay, which lose their effectiveness when acquired through passive methods. Since attackers in LFA and CPA have a specific link as their target, employing an active method to measure delay in each individual link proves to be an efficient means of detecting these attacks.

In our framework, we will employ a probing technique, which is an active method used to measure link delay, similar to the approach outlined in [14] for measuring queue delay. The queue delay, a variable component, serves as an indicator of network congestion dynamics within the path. Consequently, we anticipate enhanced precision in identifying congested links within the network. Instead of measuring E2E delay, a process that generates redundancy and adds network overhead due to the monitoring of multiple paths between node pairs, we use a method that assesses each individual link in the network with minimal impact on SDN switches, thus mitigating network overhead. This method operates by crafting a specialized control packet, time-stamped as a 'probe packet,' and introducing it into the data-path. At the path's terminus, the receiving node assesses the path's delay by measuring the time taken for the probe packet to traverse the entire route. Typically, this method employs relatively petite probe packet sizes, frequently employing 24-byte packets within Ethernet frames to ascertain link latency [18].

The details of the link congestion detection algorithm are shown in Figure 6. The algorithm starts by requesting the Topology Discovery Module to discover the network topology and generate the topology graph G. Then, the Link Congestion Module constructs the tree that covers all links in the network and minimizes the total number of links in each path. The proposed algorithm is essentially an adapted form of the Minimum Spanning Tree (MST) algorithm. It prevents the repetition of any node within a given path, although nodes may still be duplicated across distinct paths within the tree. This is due to our utilization of the tree for deriving individual paths. Subsequently, we partitioned the tree into levels. The initial level (Level 1) commences at the root node of the tree. Progressing from the root to the second node (the first link in the path) of each path signifies Level 2. Similarly, moving from the second node to the third node (the second link in the path) designates Level 3, and so on.

Taking advantage of OpenFlow capabilities, we aim to inject packets into the network to assess path delay. For every path extracted from the tree, we systematically inject packets at the root node. This guarantees that the probe packet precisely traverses the same path, and upon reaching the final node, it is sent back to the controller. To calculate the departure time at the initial node and the arrival time at the last node in the path, our approach involves calculating the delay from the controller to the first node, $RTT_{Controller \rightarrow Node}$ (RRT is Round-Trip Time) from the controller to node, also from the last node to controller we calculate $RTT_{Node \rightarrow Controller}$. Then we use Equation 1 and Equation 2 to calculate departure time at first node and arrival time at last node, respectively.

$$Time_{First-Node} = Time_{Send} + (RTT_{Controller \rightarrow Node})/2 \quad (1)$$

$$Time_{Last-Node} = Time_{Arrival} + (RTT_{Node \rightarrow Controller})/2 \quad (2)$$

Here $Time_{Send}$ represents the time of sending probe packet form the controller, and $Time_{Arrival}$ represents the arrival time of the packet to the controller. The output of both Tree Construction and Path Latency procedures will be used to calculate link delay. The details of these two procedures are in Algorithm 2 and shown in Figure 6. In the end, the link delay of each link will be calculated, and if any of those links exceed the threshold, an alarm will be sent to the CPA detection module.

5.3 CPA detection module

This research aims not to enhance the accuracy of or alter existing machine learning techniques but rather to present a novel, cost-effective approach for detecting and mitigating CPA. This module possesses the capability to identify attacks through the application of ML techniques. If link congestion is detected, this module will activate. Otherwise, it will periodically collect traffic flow data from all switches within the preceding 15 seconds. To accomplish this, the controller sends out flow-stats requests. As a result, each switch provides a flow-stats reply message, including all flow entries in its individual flow table. In accordance with OpenFlow specification 1.3 [19,20], the polling request message's length for a single flow poll is 122 bytes, with a reply to length of 174 bytes. Consequently, a reduction in the interval time for collecting traffic statistics will lead to higher network costs.

These features are subsequently employed by a detection module for the purpose of categorizing each flow as either normal or anomalous, utilizing a binary classification ML algorithm. The initial step is to create a model using a classification ML algorithm, followed by using this model to classify network flows. It's important to acknowledge that there are multiple classification ML algorithms available for building the detection model, such as LR, NB, KNN, SVM, DT, and RF [13]. In our specific implementation, we employed the DT algorithm. In the event of attack detection, the module promptly initiates blocking measures against the attack source.

An essential component is a flow-based labeled dataset that represents network traffic. The selection of features for model training plays a pivotal role, as it exerts a substantial influence on the performance of ML algorithms. We have opted for a set of five specific features for constructing our model: pkt_count, byte_count, pkt_size, same_host, and same_host_port. Using these features, the module will assess specific flow IDs for anomalies, leveraging the model we have developed. Flows classified as normal require no further action. In this research, we employed a dataset similar to the one used for detecting DDoS attacks [13]. It is noteworthy that both CPA and LFA fall under the same category of DDoS attacks, differing solely in their attack targets: LFA and CPA target network links, while traditional DDoS attacks focus on end devices or servers.

Algorithm 2: Link Congestion Detection

Input:
$Net_{Toplogy}$: Network topology
$Link_{Shared\_List}$: List of shared links in $Net_{Toplogy}$,
$Link_{List}$: All individual links in $Net_{Toplogy}$,
$Delay_{Threshold}$: The delay Threshold
$Root_{Tree}$: Node represent the root of the constructed tree
Nod_{eList}: List of nodes in $Net_{Toplogy}$
$Edge_{List}$: List of Edge in $Net_{Toplogy}$
Procedure *Tree_Constraction* ($Root_{Tree}$, $Net_{Toplogy}$, $Link_{Shared}$, or $Link_{List}$) Drive a tree covering $Link_{Shared\_List}$ or $Link_{List}$
$Tree = Root_{Tree}$
$d[Root_{Tree}] \leftarrow 0$; $d[u] \leftarrow \infty$; $pred[i] \leftarrow nil$: For each $u \neq Root_{Tree}$; $u\ \varepsilon\ Node_{List}$
while $Q \neq \varphi$ **do**
for *each* $u\ \varepsilon\ Node_{List}$ **do**
insert u with key $d[u]$ into the priority queue Q
$j \leftarrow Extract - Min(Q)$
for every node i, $i\ \varepsilon\ Tree$ and i is adjacent to j **do**
$alt = d[j] + Edge_{weight}$ (j, i) /* $Edge_{weight}$ (j, i) is edge weight =1 */
for all edge in $Edge_{List}$ **do**
$d[i] \leftarrow alt$
$pred[i] \leftarrow j$ /* set i as a child node of j */
if *edge* (i, j)ε *Tree* **then**
add the *edge* into *Tree*

Endif

Procedure *Measure_Link_Delay* (*Tree, Delay$_{Threshold}$*)

Partitioned tree into levels (1, N) /*N, number of level form Root$_{Tree}$ at level 0 */

for $m = 1$ to N **do**

for $every\ path\ in\ level\ m$ **do**

inject Probe Packet from tree$_{root}$ travels path to m

$Calculate\ RTT$ $_{Controller \rightarrow Node}$ /* Round-Trip Time*/

$Calculate\ RTT$ $_{Node \rightarrow Controller}$

from $Equation$ 1&2 $claculate$: $Link\_Delay = \text{Time}_{\text{Last\_Node}} - \text{Time}_{\text{First\_node}}$

if $Link\_Dely > Delay_{Threshold}$ **then**

Sent alarm to *CPA_Detection_Module*

Endif

Fig. 6: Link congestion detection algorithm.

5.4 Attack mitigation module

The primary purpose of the Anomaly Mitigation module is to initiate mitigation measures once the detection module confirms the presence of a malicious flow. This is essential to prevent disruptions in the network and potential performance degradation. Within our framework, we actively block the source of the detected attack. As a part of future research, we plan to implement a traffic engineering module that will enable us to configure network flows with the assistance of the controller, aiming to mitigate such attacks.

6 Simulation and results

The evaluation of the system includes the evaluation of the costs required in our proposed method with other methods, in addition to efficiency of using link congestion as indicator to CPA. For that we first established an SDN topology using Mininet [21] emulator and Ryu controller. We built a custom network topology as shown in Figure 2. After the SDN network topology was set up, we used a python script to generate normal and attack traffic. In our results, we calculate the average of five experimental runs. The following sections will discuss the evaluation of our method.

The communication cost of collecting traffic statistics for detecting CPA in our system is shown Table 1, and 2. In Table 1, we analyze the communication cost required for gathering flow statistics and the communication cost needed to calculate link congestion. We assume a fixed number of switches (10 switches) while increasing the number of flows. Initially, the network is initialized with 6 flows chosen randomly, and for each subsequent step, we add 2 flows with random sources and destinations. As depicted in the table, the communication cost required to calculate flow statistics is significantly higher than the cost required to calculate the delay of each link in the network. This indicates that our proposed system, capable of adeptly balancing these two approaches, leads to a reduction in communication costs associated with CPA detection.

In Table 2, we examined networks of varying sizes (5, 10, and 15 switches) while keeping the number of flows constant (6 flows in each network), with the flow generation being random. It becomes apparent that a more addition of 5 switches to the network size leads to a noteworthy escalation in communication costs during the collection of flow statistics. On the other hand, when measuring link delay through probing techniques, the cost increase is comparatively modest.

Table 1: Number of flows with fixed number of switches.

Number of flows	6	8	10	12
flow approach	5624	7992	9768	11544
Proposed Probing approach	912	1296	1584	1872
	Communication cost (Bytes)			

Table 2: Number of switches with a fixed number of flows.

Number of switches	5	10	15
flow approach	7104	9768	12728
Proposed Probing approach	1152	1584	2064
	Communication cost (Bytes)		

In Table 3 and 4, we analyze the communication cost required in our proposed system and compare it with approaches that depend solely on flow statistics, as proposed in [13, 22]. The network consists of 10 switches and 10 flows generated randomly.

Table 3 highlights an initial observation: the cost associated with our proposed solution is slightly higher than that of the flow-based alternative [13,22]. This discrepancy arises because our method simultaneously calculates the costs related to flow statistic collection and link delay measurement at 5-second intervals, whereas other methods solely rely on flow statistics. However, over time, we observe a significant improvement in our method's performance. This enhancement can be attributed to our approach, which collects flow statistics every 10 seconds and measures link delay every 5 seconds, as opposed to other methods that collect flow statistics every 5 seconds.

Table 3: The communication cost: in other schemes collecting flow statistics every 5 seconds, while in our proposed scheme, flow statistics collected every 10 seconds and delay measurements every 5 seconds.

Times (S)	5	10	15	20	25	30	35	40	45
[13], [22]	9768	19536	29304	39072	48840	58608	68367	78144	87912
Proposed	11352	12936	24228	25872	37224	38808	50160	51744	63096
	Communication cost (Bytes)								

Table 4: The communication cost: in other schemes involves collecting flow statistics every 5 seconds, while in our proposed scheme, flow statistics collected every 15 seconds and perform delay measurements every 5 seconds.

Times (S)	5	10	15	20	25	30	35	40	45	50	55	60	65
[13], [22]	9768	19536	29304	39072	48840	58608	68367	78144	87912	97680	107448	117216	126984
Proposed	11352	12936	14520	25872	27456	29040	40392	41976	43560	54912	56496	58080	69432
	Communication cost (Bytes)												

In Table 4, our method extends the interval for collecting flow statistics from 10 seconds to 15 seconds, while maintaining a consistent 5-second interval for measuring link delay costs. Conversely, the alternative solution retains a fixed 5-second interval for gathering flow statistics. As depicted in the table, we can observe a substantial improvement in the communication cost associated with our proposed solution as we increase the interval for collecting flow statistics. To provide an example, in Table 3, at the 45-second mark, our method incurs a communication cost of approximately 63,000 bytes, whereas the other solutions result in 88,000 bytes. However, at the same 45-second mark in Table 4, our costs are reduced to 43,500 bytes, while the other solution remains at 88,000 bytes.

The results in Table 5, represent the communication cost, as in Table 4, However, in this experiment, we perform attacks several times against the network at times 30, 35, and 60. As depicted in the figure, at the time of the attacks, the communication cost increases compared to the communication cost in Table 3. This occurs because when link congestion is detected, an alarm is sent to the CPA detection module, which then collects flow statistics even if the interval time (15 seconds) has not yet elapsed. However, our proposed method still outperforms other methods in terms of the total communication cost.

Table 5: The communication cost while the network under attacks at different interval times (at time: 30, 45, and 60): in pair flow approaches collecting flow statistics every 5 seconds, while in our proposed scheme, flow statistics collected every 15 seconds and perform delay measurements every 5 seconds.

Times (S)	5	10	15	20	25	30	35	40	45	50	55	60	65
[13], [22]	9768	19536	29304	39072	48840	58608	68367	78144	87912	97680	107448	117216	126984
Proposed	11352	12936	14520	25872	27456	38808	50160	51744	63096	74448	76032	87384	98736
	Communication cost (Bytes)												

7 Conclusions

In this research, we addressed the security challenges in SDN and focused on combating CPA, which are novel forms of DDoS attacks targeting SDN control channels. Our proposed SDN-based security solution utilizes hop-by-hop network measurement and ML to detect CPA effectively. We demonstrated that our approach optimizes the utilization of SDN controller resources by activating the ML model at longer time intervals or only when abnormalities in links are detected, thereby reducing communication costs associated with CPA detection. Furthermore, we extended our investigation to evaluate the impact of various network parameters on communication costs and the effectiveness of our approach during network attacks. Our results show that our proposed solution performs favorably in terms of communication cost, especially when increasing the interval for collecting flow statistics. During network attacks, the system exhibits an increase in communication cost due to the activation of CPA detection mechanisms but continues to outperform other methods. Moreover, we observed that latency, particularly link latency, serves as a valuable parameter for detecting link congestion and abnormal network behavior during attacks. This observation highlights the effectiveness of our proposed approach in addressing security challenges in SDN environments.

In conclusion, our research presents a promising security solution for SDNs, offering effective CPA detection and optimized resource utilization. This work contributes to enhancing the security and reliability of SDNs in the face of evolving threats. In our future work, we plan to use a traffic engineering module to reconfigure the network, mitigate the attack, and compare its performance with the mitigation module used in this paper.

Conflicts of Interest Statement

The authors certify that they have NO affiliations with or involvement in any organization or entity with any financial interest (such as honoraria; educational grants; participation in speakers' bureaus; membership, employment, consultancies, stock ownership, or other equity interest; and expert testimony or patent-licensing arrangements), or non-financial interest (such as personal or professional relationships, affiliations, knowledge or beliefs) in the subject matter or materials discussed in this manuscript.

References

[1] B. Wolfgang, and M. Menth. 2014. "Software-Defined Networking Using OpenFlow: Protocols, Applications and Architectural Design Choices" Future Internet 6, no. 2: 302-336. https://doi.org/10.3390/fi6020302.

[2] F. M. Shoaib, S. Riaz, and A. Alvi. 2023. "Security and Privacy Issues in Software-Defined Networking (SDN): A Systematic Literature Review" Electronics 12, no. 14: 3077. https://doi.org/10.3390/electronics12143077.

[3] W. Raniyah, R. Ahmad, and S. Alhiyari. 2021. "SDN-OpenFlow Topology Discovery: An Overview of Performance Issues" Applied Sciences 11, no. 15: 6999. https://doi.org/10.3390/app11156999.

[4] T. Dargahi, A. Caponi, M. Ambrosin, G. Bianchi and M. Conti, "A Survey on the Security of Stateful SDN Data Planes," in IEEE Communications Surveys & Tutorials, vol. 19, no. 3, pp. 1701-1725, thirdquarter 2017, doi: 10.1109/COMST.2017.2689819.

[5] W. Song, K. Gomez, K. Sithamparanathan, M. Rizwan Asghar, Giovanni Russello, and Paul Zanna. 2021. "Mitigating DDoS Attacks in SDN-Based IoT Networks Leveraging Secure Control and Data Plane Algorithm" Applied Sciences 11, no. 3: 929. https://doi.org/10.3390/app11030929.

[6] J. Wang, et al.,"Detecting and Mitigating Target Link-Flooding Attacks Using SDN" in IEEE Transactions on Dependable and Secure Computing, vol. 16, no. 06, pp. 944-956,2019. doi: 10.1109/TDSC.2018.2822275.

[7] R. Xie et al., "Disrupting the SDN Control Channel via Shared Links: Attacks and Countermeasures," in IEEE/ACM Transactions on Networking, vol. 30, no. 5, pp. 2158-2172, Oct. 2022, doi: 10.1109/TNET.2022.3169136.

[8] K. M. Sudar, M. Beulah, P. Deepalakshmi, P. Nagaraj and P. Chinnasamy, "Detection of Distributed Denial of Service Attacks in SDN using Machine learning techniques," 2021 International Conference on Computer Communication and Informatics (ICCCI), Coimbatore, India, 2021, pp. 1-5, doi: 10.1109/ICCCI50826.2021.9402517.

[9] R. Santos, D. Souza, W. Santo, A. Ribeiro, E. Moreno. Machine learning algorithms to detect DDoS attacks in SDN. Concurr. Comput. Pract. Exp. 2020, 32, e5402.

[10] B. Celesova, J. Val'ko, R. Grezo and P. Helebrandt, "Enhancing security of SDN focusing on control plane and data plane," 2019 7th International Symposium on Digital Forensics and Security (ISDFS), Barcelos, Portugal, 2019, pp. 1-6, doi: 10.1109/ISDFS.2019.8757542.

[11] V. Deepa, K. M. Sudar and P. Deepalakshmi, "Detection of DDoS Attack on SDN Control plane using Hybrid Machine Learning Techniques," 2018 International Conference on Smart Systems and Inventive Technology (ICSSIT), Tirunelveli, India, 2018, pp. 299-303, doi: 10.1109/ICSSIT.2018.8748836.

[12] X. Zhang, L. Xie, W. Yao, Spatio-temporal heterogeneous bandwidth allocation mechanism against DDoS attack.

J. Netw. Comput. Appl. 2020, 162, 102658.

[13] O. Rahman, M. A. G. Quraishi and C. -H. Lung, "DDoS Attacks Detection and Mitigation in SDN Using Machine Learning," 2019 IEEE World Congress on Services (SERVICES), Milan, Italy, 2019,pp.184-189, doi: 10.1109/SERVICES.2019.00051.

[14] A. Allakany. and K. Okamura. 2017. Latency Monitoring in Software-Defined Networks. In Proceedings of the 12th International Conference on Future Internet Technologies (CFI'17). Association for Computing Machinery, New York, NY, USA, Article 5, 1–4.

[15] P. Goransson, C. Black, T. Culver, Software Defined Networks: A Comprehensive Approach; Morgan Kaufmann: Burlington, MA, USA, 2015.

[16] J. Zheng, Q. Li, G. Gu, J. Cao, D. K. Y. Yau and J. Wu, "Realtime DDoS Defense Using COTS SDN Switches via Adaptive Correlation Analysis," in IEEE Transactions on Information Forensics and Security, vol. 13, no. 7, pp. 1838-1853, July 2018, doi: 10.1109/TIFS.2018.2805600.

[17] M. Rezazad, M. R. Brust, M. Akbari, P. Bouvry, N. Cheung, Detecting Target-Area Link-Flooding DDoS Attacks Using Traffic Analysis and Supervised Learning. In: Advances in Information and Communication Networks. FICC. Advances in Intelligent Systems and Computing, 2019. 887, 180–202

[18] K. Phemius and M. Bouet, "Monitoring latency with OpenFlow," Proceedings of the 9th International Conference on Network and Service Management (CNSM 2013), Zurich, Switzerland, 2013, pp. 122-125, doi: 10.1109/CNSM.2013.6727820.

[19] Z. Su, T. Wang, Y. Xia, M. Hamdi. CeMon: A cost-effective flow monitoring system in software defined networks. Comput. Netw. 2015, 92, 101–115.

[20] B. Pfaff, B. Lantz, B. Heller,. Openflow switch specification, version 1.3. 0. Open Networking Foundation; 2012.

[21] M. Team, "Mininet: An Instant Virtual Network on your Laptop (or other PC) - Mininet", Mininet.org, 2020. [Online]. Available: http://mininet.org/.

[22] H. Polat, O. v, A. Cetin, Detecting DDoS Attacks in Software-Defined Networks Through Feature Selection Methods and Machine Learning Models. Sustainability 2020, 12, 1035.