Towards a Cyber Defense System in Software-defined Tactical Networks

Sean Kloth*†, Paulo H. L. Rettore†, Philipp Zißner†, Bruno P. Santos‡, and Peter Sevenich†
*Institute of Computer Science 4, University of Bonn, Bonn, Germany
†Department of Communication Systems, Fraunhofer FKIE, Bonn, Germany
‡Department of Computer Science, Federal University of Bahia, Salvador, Brazil
Email: s6seklot@uni-bonn.de, {paulo.lopes.rettore, philipp.zissner, peter.sevenich}@fkie.fraunhofer.de, and bruno.ps@ufba.br

***Abstract*—This study investigates the robustness of a Software-defined Networking (SDN) controller when confronted with a Distributed Denial-of-Service (DDOS) attack in a tactical environment. A proactive defense mechanism is introduced to detect and respond to a flooding of "packet-in" requests, triggering a response once the network features indicate an anomaly. The methodology consists of two components: the Cyber Defense Agent (CDA), consisting of monitoring, feature engineering, detection, and responses, and the Cyber Attack Agent (CAA), including the preparation, execution, and evaluation of the attack. The CDA monitors all the IP flows from the SDN controller and processes four main features such as the average number of "packet-in" requests, the response time to these requests, the entropy of IP addresses and ports for source and destination, and "packet-in" requests per switch to identify compromised switches. All the components were emulated and tested, collecting quantitative evidence to demonstrate the effectiveness of both agents.**

***Index Terms*—Tactical networks, Software-defined Networking, Resilience, Cyber security**

I. Introduction

Software-defined Networking (SDN) separates the network control from data forwarding, allowing for direct programmability through decoupling control and data plane [1]. This introduces versatility across diverse scenarios, like civilian and military, that can be established through the usage of SDN [2], by eliminating dependence on specific transmission technologies and vendors. This flexibility allows networks to incorporate various transmission technologies, including Bluetooth, Wi-Fi, Ethernet, and those relevant to Tactical Networkss (TNs) such as High Frequency (HF), Ultra High Frequency (UHF),Very High Frequency (VHF), and Satellite Communications (SatCom), which have specific requirements with a necessity for tailored SDN solutions [3]–[7].

Numerous factors contribute to fluctuations in TNs, including mobility, radio capabilities, terrain, security risks, and signal interference. Consequently, tactical edge systems must demonstrate resilience against connection disruptions, network changes, cybersecurity risks, and dynamic policy management. Adopting SDN to enhance TNs emerges the Software-defined Tactical Network (SDTN). However, several challenges arise from that symbiosis, such as network vulnerabilities, Single Point of Failure (SPoF), Controller Placement Problem (CPP), ever-changing topologies, and control overhead to network management.

A significant threat to SDTN stability emerges when controllers fall target to malicious attacks. Once in control, attackers can either disrupt or manipulate the communication to their advantage, requiring resilient controllers to mitigate these threats. Such controllers should not only recognize ongoing attacks but also employ predefined countermeasures. Hence, the controller must be capable of detecting and responding to diverse attacks. This paper extends the previous research [6]–[9], emphasizing the need to enhance resilience in SDTNs, as the literature mainly focuses on SDN disregarding tactical aspects. Our work assesses controller vulnerability of cyber-attacks and introduces a defensive strategy to quickly detect abnormal control plane activity and respond by activating a reliable backup controller or deactivating the suspicious switch ports. This is done by introducing two agents: Cyber Defense Agent (CDA), Cyber Attack Agent (CAA). This approach enhances network resilience against controller failures and may reduce the risk of unauthorized network access. In summary, our contributions are as follows:

- Introducing two agents. The CDA is capable of monitoring, featuring engineering, detecting anomalies, and reacting. While then CAA creates successful attacks, challenging the CDA.
- A method to identify ongoing Distributed Denial-of-Service (DDOS) attacks proposing different metrics: Average *packet-in* requests, average *packet-in* requests per switch, average *packet-in* response time, and entropy of IP addresses and ports for source and destination.
- Comparing the effectiveness of a specific DDOS attack creation on the data plane versus the control plane.
- Experimental results quantifying the proposed metrics' effectiveness and impact on a DDOS attack on the data plane.

The paper is organized as follows: Section II discusses the related literature. Section III details the system's methodology and experiment construction. Section IV presents the emulation results, discusses limitations, and outlines future directions in Section V. Finally, Section VI summarizes the study and outlines the next steps.

II. Related Work

Different security and cyber threat approaches exist for SDN, including Kreutz et al. [10], considering seven different threat vectors including three directly targeting the controller. Moreover, other surveys [11], [12] list threats to SDN and controllers, including unauthorized access with consequences for the communication and integrity of messages [11]. Krishnan et al. [12] especially point out the vulnerability of the southbound interface impacting the integrity of the communications.

One of the most common threats for a controller is a Denial-of-Service (DoS) attack. Abdullah et al. [13] evaluate the performance of different controllers, namely Opendaylight, POX, and RYU, against DoS attacks. The performance is measured by the round trip time, latency, bandwidth, and throughput, while the DoS attack is created using *hping3*. The paper measures the effect of an DoS attack through these metrics and concludes which controller application deals best with such an attack, based on the implementation of the controller. However, Abdullah et al. do not introduce any countermeasures.

Using a NOX controller, Braga et al. [14] propose a lightweight detection method for DoS attacks. They employ Self Organizing Maps using six features to detect a DoS attack. The main assumption of this study is that all switches keep the statistics of all active flows. For SDTN, a test-bed and a system named Cyber Security Simulation Service (CSSS) were proposed in [15], trying to detect black hole attacks and react by removing the black holes from the network. Black holes are detected by monitoring bidirectional traffic. However, this does not address the issue of attacks against the controller.

Similarly, Alharbi et al. [16] evaluate the impact of DoS attacks in SDN on three different controllers, namely RYU (version 3.22), ONOS (version 1.10.0) and Floodlight (version 1.0). Unlike Abdullah et al. [13], the attack is not created by using *hping3*. Rather, they experimented with two different methods. The first method crafts and sends UDP or TCP packets with random source, destination IP, and MAC addresses using the Scapy library. With this approach, they archived a packet-sending rate of 500 pkts/s. The second method is TCPreplay, where previously captured traffic is injected into the network at a desired rate, achieving a packet rate of 70.000 pkts/s. The authors first evaluated the impact of a DoS attack on the controller measured by the Packet Delivery Ratio (PDR) of 10 ICMP echoes between two hosts. They concluded that all controllers could not respond after the attack rate reached over 7.000 pkts/s. These results are worsened by the network size called the attack amplification effect. This is caused by one packet triggering multiple *packet-in* requests from different switches. The second experiment evaluates the impact of a DoS attack on switches, concluding that the PDR drops to 0% if the attack rate is greater than 65.000 pkts/s. This paper only compares controllers' performance against DoS attacks.

The authors in [17] proposed SDN-Guard, an application to protect the network against DoS attacks. This system is plugged on top of the SDN controller and consists of three components. The flow management component selects routing paths and timeouts for flows. The rule aggregation component aggregates similar flow table entries, while the monitoring component collects statistics about flows, switches, and links. A separate Intrusion Detection System evaluates the *packet-in* requests. The DoS attack is created by using *hping3* in partial-mesh-switch-topology, using the Floodlight controller (version 1.2), and a total of up to 70.000 *packet-in* requests were created. This approach heavily relies on an intrusion detection system to analyze *packet-in* requests and to determine the threat probability.

Different ideas for detecting an ongoing DoS attack have been proposed. However, quite a few focus on the selection of the classifier rather than the definition of features. Meti et al. [18] evaluated three different classifiers for detecting DDOS attacks. The Naive Bayes (NB) classifier, Support Vector Machine (SVM) and Neural Network (NN) classifier were all applied to real-world TCP traffic where the data consists of the number of hosts connected in seconds, host time, which is either the peak time or off-peak-time and a label for each entry. They concluded that SVM is the best classifier for detecting anomalies in an SDN network. In [19], the authors evaluated seven different Machine Learning (ML) techniques without specifying how their feature extractor works. The seven ML techniques were both unsupervised and supervised, including k-nearest Neighbour (kNN), NB, SVM, Random Forest (RF), Artificial Neural Network (ANN), Linear Regression (LR) and Decision Tree (DT). They noticed that LR achieved the highest precision while RF was less precise but faster.

Other papers focus on features to detect a DoS attack. Yue et al. [20] propose DoS detection based on flow table features. These features include entropy of source IP addresses, similarity of flow tables, growth rate of max matched packets and max matched bytes, percentage of flows with a small number of packets, and percentage of flows with short duration. All these features were then used in different ML techniques, evaluating their performance. Mousavi et al. [21] propose calculating entropy within a given window. The entropy is calculated for the destination IP address with a window size. Their proposed method requires the definition of a threshold which, when exceeded, indicates a DoS attack.

In this paper, we are tackling the challenge of tactical networks with constrained links. This limitation it is harder to launch a DDOS attack that overwhelms a controller and forces a shutdown. However, DDOS attacks still impact the network, especially the data plane, causing problems for other users. We aim to develop a system deployed in a controller that can detect ongoing DDOS attacks that impact the data plane only using information that can be extracted from the controller.

III. Methodology

Resilience in SDTNs refers to a controller's ability to withstand and respond to attacks. Our strategy to attend resilience comprises the design of a CDA capable of monitoring, featuring engineering, detecting anomalies, and reacting. Besides, a

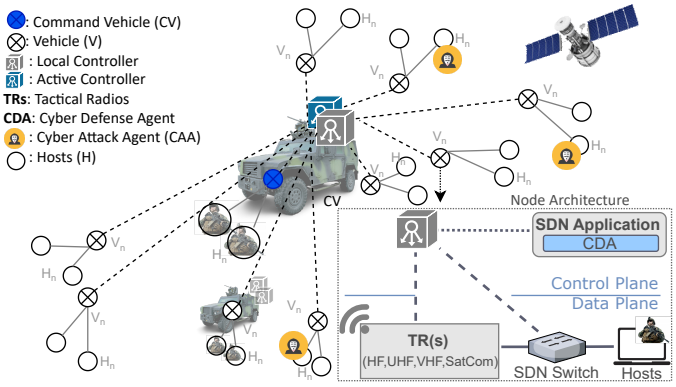

Fig. 1: Network scenario.

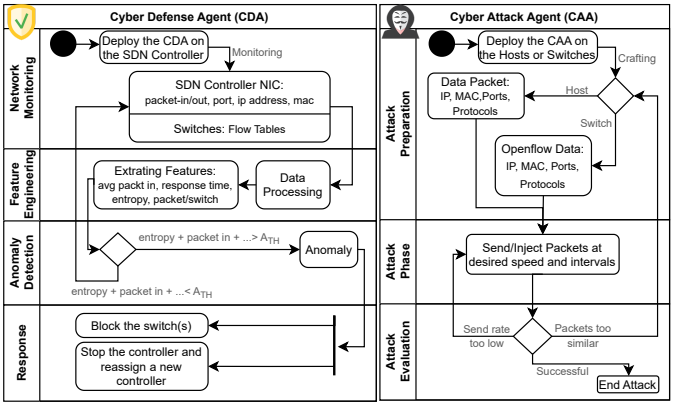

Fig. 2: Cyber Defense and Attack agents design.

CAA is proposed to create a successful attack, challenging the CDA. Fig. 1 presents an illustrative network scenario, including the essential components.

The network scenario represents a cluster of vehicles (V) controlled by a command vehicle (CV) with an active controller via 1Mbit SatCom links. Each vehicle has the minimum node architecture of two local controllers (one is active to control the cluster - e.g., command vehicle), tactical radios (TRs), SDN switches, and hosts (H), where at least two hosts (soldiers) are connected to its respective vehicle. The CDA is an SDN app executed at the active controller aiming to protect the network from cyberattacks. In contrast, the CAAs are deployed randomly across the network topology to simulate cyberattacks on the network to test its defenses.

As part of a common network scenario, user data traffic is simulated to assess the impact on the proposed metrics used by the CDA to identify a DDOS attack on the controller. It uses TCPreplay to inject packets into the data plane. To ensure that *packet-in* requests are generated, all packets have unique IP, MAC addresses, and ports for both source and destination hosts.

A. Cyber Attack Agent

We execute a DDOS attack by inundating the controller with a massive influx of requests, slowing down the controller's response time and causing network communication delays. In extreme cases, a large-scale DDOS attack can potentially overwhelm the controller, leading to possible unresponsiveness. In SDNs, control and user data are distinctly separated. Thus, an attacker cannot directly, without extra access, send packets to the controller and can only communicate with a switch. The switch determines the subsequent actions for incoming packets based on flow table entries installed by the controller. These actions entail forwarding packets to the correct port and recipient or discarding them. If no matching flow entry is found, the switch redirects the packet to the controller, sending a *packet-in* request with the reason *NO MATCH*. The controller then endeavors to determine the routing path for the packet and responds with a *packet-out* message, either containing the appropriate flow entry or providing no specific instruction.

The CAA follows the three steps described in Fig. 2: the preparation, the attack, and the evaluation. We employ two approaches to execute a DDOS attack. In the first, originating from the data plane, we leverage the SDN protocol, specifically OpenFlow. This involves inducing the switch to inundate the controller with a barrage of massive *packet-in* requests. In the second approach, stemming from the control plane, we presume that the attacker has compromised a switch and exploits this access to orchestrate a DDOS attack without altering the software. The minimum setup for a data plane attack involves just one host connected to a switch, while the control plane attack only requires a connected switch. We assume a direct connection between the switch and the controller. Additionally, our work builds upon the research in [16], which previously demonstrated the effectiveness of amplifying the attack through connected switches.

1) Data Plane: To force the switch to inundate the controller with massive *packet-in* requests, the attacker must send packets that cause a flow table miss in the switch. To trigger a flow table miss in OpenFlow 1.3, the sent packet requires an IP, MAC addresses, and ports for both source and destination unknown to the switch. As Alharbi et al. [16] already described, the Scapy Python library provides tools to craft packets with unique IP, MAC addresses, and ports for both source and destination and to send them at varying speeds. TCPreplay is a suite of utilities that enables the injection of previously captured packets into an interface at varying speeds. In this setup, the data for TCPreplay is created by Scapy.

2) Control Plane: A further approach is to take control of the switch directly, which is a more challenging task compared to the data plane approach. However, investigating the impact of such a threat is still interesting, as this could be more destructive than a simple data plane attack. Furthermore, we assume that the attacker does not modify the software, as this would lead to a further investigation of the vulnerabilities of OpenFlow implementation instead of abusing the protocol. To achieve this, we use TCPreplay to inject previously captured packets on the switch interface towards the controller.

B. Cyber Defense Agent

The proposed CDA, shown in Fig. 2, is capable of monitoring the SDN, employing metrics to detect potential anomalies (DDOS attacks) and respond appropriately. We assume that the controller has the required computing capabilities to collect and inspect the network traffic. Notice that the controller cannot access traffic information from the user network. Therefore, we focus on metrics that can be extracted from the controller, primarily based on sent and received packets.

1) Network Monitoring: After deploying the CDA on the controller, the monitoring phase collects all relevant information from the IP traffic using a packet sniffer that is designed to instantly respond to the received packets. The packet sniffer uses a Python extension module, Pcapy, that allows to access the routines from the Pcap packet capture library within a Python environment. We use Pcapy over other well-known libraries for packet sniffing, such as Pyshark or Scapy, as it provides the fastest processing of incoming packets. The biggest drawback of Pyshark is that it buffers packets in such a way that packets are still being processed even after the communication is finished. This introduces a delay in processing the data and responding to an attack, making this tool unfeasible. Similarly, Scapy does not provide fast processing of the captured packets, as stated in the documentation of the library.

2) Feature Engineering: In sequence, the feature engineering processes and temporarily stores the logs to extract the metrics described below. All proposed features can be used either in a threshold-based detection system or in a machine learning-based system. With this, we decouple the features from the detection system, allowing us to test the effectiveness of each component individually.

a) Entropy of packet-in requests: A well-known and studied indicator for detecting DDOS attacks is calculating the entropy of selected features. As features, we propose calculating the entropy for IP addresses (IP_{src} and IP_{dst}) and ports ($Port_{src}$ and $Port_{dst}$) for source and destination on the data plane level. Packets that cause a flow table miss result in a *packet-in* request. This miss is most likely due to unknown IP addresses and ports for source and destination, and these addresses vary heavily in an effective DDOS attack. The entropy $H_{feature}$ is calculated using Shannon's equation defined in (1) where p(x) = {IP_{src}, IP_{dst}, $Port_{src}$, $Port_{dst}$,...} denotes the relative occurrence of one of the specific features within a specific time window.

$$H_{feature} = -\sum p(x) log(p(x)) \tag{1}$$

b) Average number of packet-in requests: Pin_{avg} is determined by calculating the mean of requests over a specified time interval, irrespective of their source. As defined in Equation (2), this calculation relies on the chosen monitoring time interval. The interval must be long enough to provide a reliable basis for monitoring the traffic (avoiding false positives) but not too long, as the attack should be detected as early as possible.

Algorithm 1 Anomaly detection mechanism

while *True* **do**
  $Rep_{\text{avg}} \leftarrow$ calculate average response time;
  $Pin_{\text{avg}} \leftarrow$ calculate average number of packets;
  $Pkt_{\text{switch}} \leftarrow$ average number of packets per switch;
  $H_{feature} \leftarrow$ calculate entropies;
  if $Rep_{\text{avg}} + Pin_{\text{avg}} + H_{feature} > A_{TH}$ **then**
    Identify_compromised_switches(Pkt_{switch});
    Deploy counter measurements;
repeat process;

$$Pin_{\text{avg}} = \frac{\sum_{i=1}^{N_{\text{ports}}} \text{Number of packet-in on port } i}{\text{Time interval}} \tag{2}$$

c) Average response time for packet-in requests: Rep_{avg} is determined by examining each request and measuring the time it takes to receive a response based on source, destination port, and packet type, as shown in Equation (3). This calculation is performed for all *packet-in* requests, assuming that an attacked controller would exhibit delayed responses. The to-be-defined parameter is the window size in which the average response time is to be calculated.

$$Rep_{\text{avg}} = \frac{\sum_{i=1}^{N_{\text{requests}}} (\text{request time i}) - (\text{response time i})}{\text{Time interval}} \tag{3}$$

d) Identification of compromised switches: Identifying compromised switches is crucial in addressing a DDOS attack within such a network. This identification is necessary either when the switch itself is compromised or when compromised hosts are connected to the switch, thereby indirectly compromising it. Therefore, the system monitors all the packets per switch in order to capture abnormal behaviors and be able to trace the sources of attacks. We assume that a DDOS attack will cause an increase of monitored packets of the compromised switch as well as the neighboring switches. With this, we can identify the area in the topology graph where the attack is created. For equation (4), a time window needs to be defined in which the packets per switch are to be calculated.

$$Pkt_{\text{switch}} = \sum_{i=1}^{\text{Time interval}} \text{i, i: Pkts to Controller} \tag{4}$$

3) Anomaly detection mechanism: The above-defined features are part of the anomaly detection mechanism. These features indicate an ongoing DDOS attack by either reaching a pre-defined threshold (A_{TH}) or being used in a machine learning system. If a DDOS attack is detected, the compromised switches need to be identified by evaluating the monitoring results of the switches. The final step is an appropriate response to the attack, such as blocking the compromised ports, isolating the switches, or reassigning uncompromised switches to a backup controller. This process is constantly executed to ensure resilience, as shown in Algorithm 1.

4) The response mechanism: Once the features are established, a response must be triggered and indicate an ongoing attack. To do this, compromised switches must be identified by evaluating the *packet-in* requests per switch. With this separation, different responses are possible. The most naive one is to reassign the switches to a backup controller. However, this can be very costly, and an alternative can be to block/timeout the compromised switches.

IV. Evaluation

A. General settings of the experiment

The experiment is conducted in a Mininet environment. The network scenario is configured with ten switches (Vehicles), each linked to 2 hosts (Soldiers) as shown in Fig. 1. All vehicles are connected to the same controller (Command Vehicle - CV) in a linear topology. As discussed earlier, it is assumed that all vehicles are equipped with two local controllers, but only the CV has an active controller managing all vehicles. The controller is provided through a Ryu rest API application. The connections between switches and hosts are set to 2 Mbit/s with a delay of 2 seconds to simulate a constraint link such as SatCom.

The traffic is generated using TCPreplay, injecting previously crafted packets into the network. The idea here is to simulate normal traffic with some variation as expected in a tactical network with nodes using different user applications (Command and Control (C2)). The packets consist of TCP and UDP packets with the following specifications for the user traffic: 1) The IP addresses cover a range of 78 different IPs, while the ports cover a range of 500 different numbers; 2) The MAC addresses are chosen arbitrarily as they do not yield any useful information. This is due to the sheer range of possibilities as well as the changing scenario, as there exists no guarantee of a whitelist of MAC addresses; 3) Other protocols can also be used, but the extracted features do not consider different protocols apart from TCP and UDP.

Similarly, the attack traffic is created using TCP and UDP as protocols, with the major difference being the bigger range of IP addresses and ports used. In numbers, the range is doubled compared to the user traffic, and it needs to be bigger, as an attack that covers the same range would be ineffective. The assumption is that the network should be built to manage the planned traffic. Given this assumption, all switches will install a correct flow table for all routes, thus reducing the frequency of *packet-in* requests generated.

B. Comparison of data plane and control plane attack

To evaluate the efficacy of attacks from different planes in SDNs, we conducted experiments creating DDOS attacks from the data and control planes. In the former, we used one attacker host and varied the number of switches in the network. To evaluate the effectiveness of such an attack as well as confirm the amplification effect, we measure the number of *packet-in* requests. In the last, we assumed the attacker gains access to the switch and injects packets through it, varying the number of compromised switches in the network. A total of 1,000,000 packets were injected, and the switches ranged from 2 to 16, increasing by 2 in both experiments.

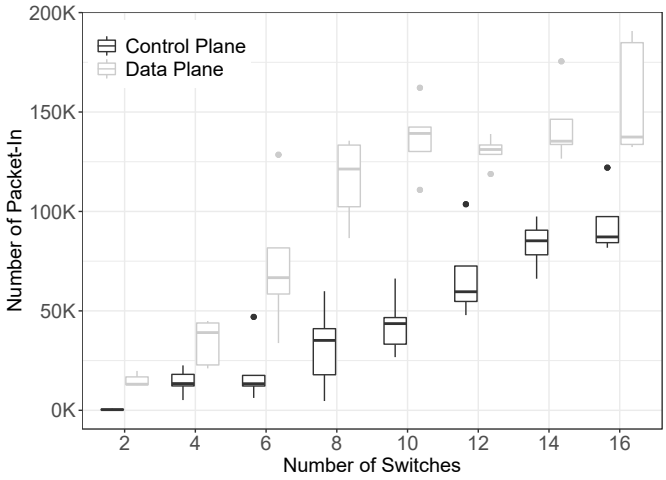

Fig. 3: Attack on the Control vs Data Plane.

Fig. 3 compares the two methods within the same topology based on eight runs to mitigate the variation capture in a single execution. It is possible to observe the amplification effect of the switches. As more switches are added, more *packet-in* requests are generated as a switch forwards unknown packets to its neighbor switches. This is explained by the fact that those switches also do not have a corresponding flow table entry; thereby, they all send a *packet-in* request. The second observation is that a data plane attack is far more effective than a control plane attack. For all experiments (2 to 16 switches), we also noticed the number of successful packets (*packet-in*) is considerably less than the total of packets injected. The increase of *packet-in* requests in the data plane attack stagnates after ten switches. This is due to the stress on the system. As more hosts try to inject packets, the switches are overloaded, and the total number of requests stagnates.

The reason for the worse performance on the control plane is the encapsulation of the injected packet. As packets are injected, the switch encapsulates them into an OpenFlow packet, which is then sent to the controller even putting multiple packets into one OpenFlow packet. To ensure a fair comparison the total number of *packet-in*-requests is considered as the controller should process every *packet-in* request similarly fast. This makes the switch the bottleneck of the attack, and as the switch is not a powerful device, the attack is not very effective. To make this type of attack more effective, the attacker would have to circumvent the encapsulation by changing the switch to be able to craft custom packets and send them on a lower level. This comparison shows that an DDOS attack from the data plane is far more effective than an attack from the control plane. With less effort, the attacker can cause more harm, as hosts are not as protected as switches. Additionally, in a large network, the attacker profits from the amplification effect of the other switches, while an attack from the control plane requires more effort to generate more

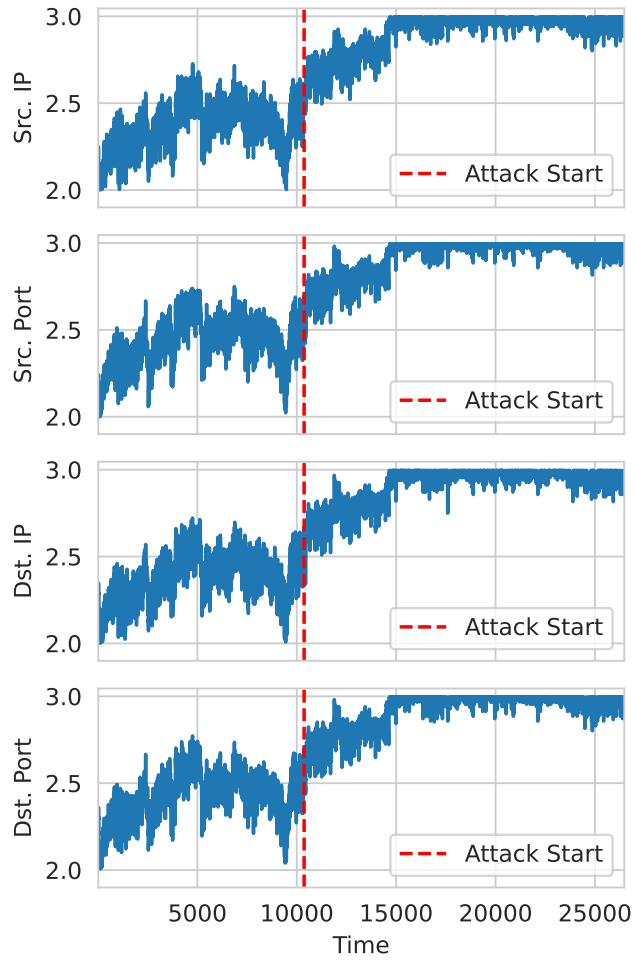

Fig. 4: Entropy of src./dst. IP addresses and ports.

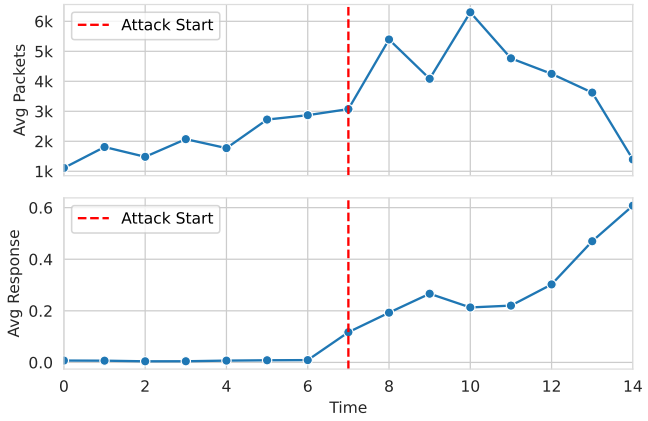

Fig. 5: Avg. response time and number of *packet-in* requests.

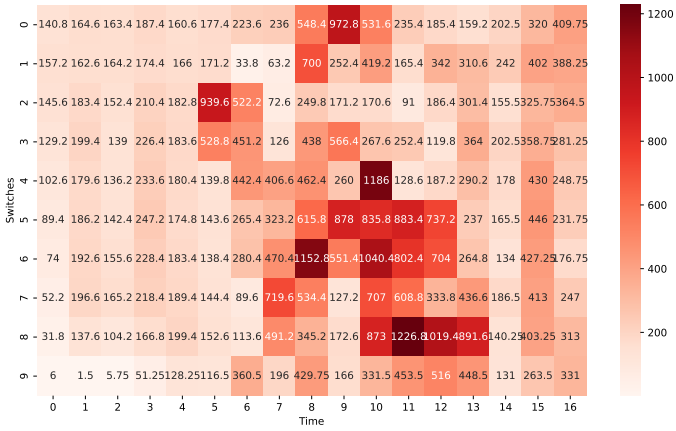

Fig. 6: Packet-in per switch.

packets.

C. Evaluation of the features

For these evaluations, we compare the processed features during normal user traffic and the attack traffic over time. We start the experiment with an idle phase where only the user data traffic is active, and after some time has passed, the attack begins. For all features, we set the time window to 20 seconds to observe the historical traffic and capture the network behavior. After a wide exploration, this time interval proved to be the most suitable as it is not too short to consider isolated peaks, such as new additions of single nodes, and not too large to miss abnormal traffic behavior.

1) Entropy: The entropy of the IP addresses and ports for source and destination are depicted in Fig. 4. As noticed, the results are similar for all four features, meaning that if an attacker manipulates any of these features, the entropy can still capture the anomaly. During the idle phase, the entropy varies most frequently between 2 and 2.7, never reaching the maximum of 3. On the other hand, during the attack phase, the entropy varies between 2.5 and 3, with a tendency to 3.5.

2) Average packet-in requests and response time: The result of the *avg number of packet-in requests* depicted in Fig. 5 (top) shows a clear difference between the idle and the attack phase. Initially, the number of requests varies between 1000 and 3000 packets, followed by an increase to roughly 6000 requests every 20 seconds. The *average response time for packet-in requests*, Fig. 5 (bottom), depicts a similar behavior. Initially, the response time is low, and it considerably increases once the attack starts. For the first part, the response is immediate, with a response time below 0.1 seconds. During the second part, where the attack is running, the time increases to 0.6 seconds, averaging around 0.4 seconds. Furthermore, we can see the correspondence of peaks occurred shortly after the peaks of the average number of packet-in requests, as those cause delays for the controller to respond.

All the features showed similar results with different measurements. Thus, we can conclude that different metrics/features processed can be used as indicators of an attack.

3) Identification of compromised switches: The heat map in Fig. 6 shows the calculated *packet-in* requests per switch. The compromised hosts were selected arbitrarily as from the controller's perspective all hosts are the same. Noticeably, packets

are spread more evenly over the different switches during the idle phase. No switch sends considerably more packets than others. During the attack phase, some switches send many more packets than others. Additionally, for neighboring switches an increase in *packet-in* requests can be seen. With this, we can identify the area in which the CAA is deployed. Depending on the difference between the peak of *packet-in* requests of a switch and the number of those requests for the neighboring switches, we can conclude if the attacker is connected to multiple switches or whether the amplification effect occurs. The difference between peak and neighboring switches is smaller if the CAA is connected to multiple switches as *packet-in* requests are more frequently generated by direct connections to hosts than through other switches. With this approach, we can identify compromised switches and use this information for the appropriate response.

V. Limitations and the way forward

In this section, we discuss the limitations faced in this study and the ways to move forward.

A. Virtual machine and Mininet environment

The first aspect that requires further discussion is the virtual machine and the Mininet environment. Currently, we consider the numbers to be distorted due to the hardware limitations of the virtual machine. However, as all experiments were run on the same machine, this distortion can be disregarded. Furthermore, we need to consider the limitations of the Mininet environment, which can be seen in the different values of the experiments. This means that the same experiment can be run twice and can have two different results concerning the concrete numbers. In certain cases, there is a considerably high differentiation between the results. This especially holds for switches, as those easily prove to be bottlenecks. However, the general picture stays the same. This can be seen, for example, with the average number of *packet-in* requests for a given time interval. The average in the different phases can differ up to 1000 packets. Nevertheless, a clear difference between the phases in the same experiment can be observed.

1) Way forward: Solving these limitations proves to be hard, as an improvement of the virtual machine means an upgrade of hardware, and Mininet itself is a working system and, therefore, hard to improve. The most sensible approach is to ensure that the applications used and the experiment's settings are not too costly for the system.

B. Defensive mechanism

Another limitation is the defensive mechanism itself. At the moment, static thresholds dictate whether or not the response will be triggered. This makes deploying the mechanism in different scenarios problematic, as the thresholds heavily depend on the attacker's size and the normal network traffic. Furthermore, the defensive mechanism only uses two metrics, which can cause trouble if the scenario is dynamic with new hosts added, resulting in more *packet-in* requests. Moreover, the mechanism only provides a single response switching to a backup controller. This has been made without any drawbacks, not necessarily reflecting real-world scenarios. Drawbacks include a slower or more error-prone connection and a backup controller with less computing power. Finally, the defensive mechanism is only tailored to one kind of attack. However, other attacks are also possible.

1) Way forward: An improvement for the defensive system is to deploy a dynamic system. This includes the use of machine learning to distinguish normal traffic from an attack. To use a more dynamic approach, an appropriate model must be introduced to detect an ongoing DDOS attack utilizing the detection features and the identification of compromised switches. Different responses need to be explored for an effective system before triggering a backup controller. Other options include finding the CAA or blocking compromised switches.

VI. Conclusion

In this study, we proposed a CAA and a CDA, which act in a SDTN. The CAA simulates a DDOS attack on the controller, where the controller is flooded with *packet-in* requests. The CAA consists of three steps: the preparation by crafting packets, injecting these packets, and finally evaluating the attack's success. The CDA consists of four steps: monitoring, feature engineering, anomaly detection, and response. The mechanism to detect an ongoing attack uses four processed features: the average number of *packet-in* requests, the average response time to *packet-in* requests, the entropy of IP addresses and port for source and destination, as well as monitoring the *packet-in* requests per switch. If the first three features indicate an ongoing DDOS attack, the compromised switches are identified by the heatmap of the number of sent packets. The final step is an appropriate response, including blocking the compromised switch ports or switching to a backup controller.

We evaluated two types of attack approaches, either a control plane or a data plane attack, concluding that a data plane attack is far more effective as it profits from large networks through the amplification effect and provides easier access to the network. All the features proved to be good indicators of an ongoing DDOS attack. Furthermore, the monitoring of packets per switch helps to identify compromised switches. Future work includes implementing a machine learning system that uses the defined features to decide which responses are more appropriate.

Acknowledgement

This paper was originally presented at the NATO Science and Technology Organization Symposium (ICMCIS) organized by the Information Systems Technology (IST) Panel, IST205-RSY – the ICMCIS, held in Koblenz, Germany, 23-24 April 2024. The authors would like to thank CAPES, CNPq, PROPCI-PROPG/UFBA 007/2022 - JOVEMPESQ for funding support.

REFERENCES

[1] W. Xia, Y. Wen, C. H. Foh, D. Niyato, and H. Xie, "A survey on software-defined networking," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 1, pp. 27–51, 2014.

[2] P. H. L. Rettore, P. Zissner, M. Alkhowaiter, C. Zou, and P. Sevenich, "Military data space: Challenges, opportunities, and use cases," *IEEE Communications Magazine*, pp. 1–7, 2023.

[3] M. von Rechenberg, P. H. L. Rettore, R. R. F. Lopes, and P. Sevenich, "Software-Defined Networking Applied in Tactical Networks: Problems, Solutions and Open Issues," in *2021 International Conference on Military Communication and Information Systems (ICMCIS)*, 2021.

[4] S. M. Eswarappa, P. H. L. Rettore, J. Loevenich, P. Sevenich, and R. R. F. Lopes, "Towards Adaptive QoS in SDN-enabled Heterogeneous Tactical Networks," in *2021 International Conference on Military Communication and Information Systems (ICMCIS)*, 2021.

[5] P. H. L. Rettore, M. von Rechenberg, J. F. Loevenich, R. R. F. Lopes, and P. Sevenich, "A handover mechanism for centralized/decentralized networks over disruptive scenarios," in *MILCOM 2021 - 2021 IEEE Military Communications Conference (MILCOM)*, 2021, pp. 836–842.

[6] P. H. L. Rettore, M. Djurica, R. R. F. Lopes, V. F. S. Mota, E. Cramer, F. Drijver, and J. F. Loevenich, "Towards Software-Defined Tactical Networks: Experiments and Challenges for Control Overhead," in *MILCOM 2022 - IEEE Military Communications Conference (MILCOM)*, 2022.

[7] P. Zißner, P. H. L. Rettore, B. P. Santos, R. R. F. Lopes, J. F. Loevenich, and P. Sevenich, "Improving robustness and reducing control overhead via dynamic clustering in tactical sdn," in *MILCOM 2023 - 2023 IEEE Military Communications Conference (MILCOM)*, 2023, pp. 491–496.

[8] P. H. L. Rettore, J. Loevenich, and R. R. F. Lopes, "TNT: A Tactical Network Test platform to evaluate military systems over ever-changing scenarios," *IEEE Access*, vol. 10, pp. 100 939–100 954, 2022.

[9] A. Velazquez, R. R. F. Lopes, A. Bécue, J. F. Loevenich, P. H. L. Rettore, and K. Wrona, "Autonomous cyber defense agents for nato: Threat analysis, design, and experimentation," in *MILCOM 2023 - 2023 IEEE Military Communications Conference (MILCOM)*, 2023, pp. 207–212.

[10] D. Kreutz, F. M. Ramos, and P. Verissimo, "Towards secure and dependable software-defined networks," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, 2013, pp. 55–60.

[11] T. Eom, J. B. Hong, S. An, J. S. Park, and D. S. Kim, "A systematic approach to threat modeling and security analysis for software defined networking," *Ieee Access*, vol. 7, pp. 137 432–137 445, 2019.

[12] P. Krishnan and J. S. Najeem, "A review of security, threats and mitigation approaches for sdn architecture," *Int. J. Innov. Technol. Explor. Eng*, vol. 8, no. 5, pp. 389–393, 2019.

[13] A. F. Abdullah, F. M. Salem, A. Tammam, and M. H. A. Azeem, "Performance analysis and evaluation of software defined networking controllers against denial of service attacks," in *Journal of Physics: Conference Series*, vol. 1447, no. 1. IOP Publishing, 2020, p. 012007.

[14] R. Braga, E. Mota, and A. Passito, "Lightweight ddos flooding attack detection using nox/openflow," in *IEEE Local Computer Network Conference*. IEEE, 2010, pp. 408–415.

[15] F. Battiati, G. Catania, L. Ganga, G. Morabito, A. Mursia, and A. Viola, "Csss: Cyber security simulation service for software defined tactical networks," in *2018 International Conference on Information Networking (ICOIN)*. IEEE, 2018, pp. 531–533.

[16] T. Alharbi, S. Layeghy, and M. Portmann, "Experimental evaluation of the impact of dos attacks in sdn," in *2017 27th International Telecommunication Networks and Applications Conference (ITNAC)*. IEEE, 2017, pp. 1–6.

[17] L. Dridi and M. F. Zhani, "Sdn-guard: Dos attacks mitigation in sdn networks," in *2016 5th IEEE International Conference on Cloud Networking (Cloudnet)*. IEEE, 2016, pp. 212–217.

[18] N. Meti, D. Narayan, and V. Baligar, "Detection of distributed denial of service attacks using machine learning algorithms in software defined networks," in *2017 international conference on advances in computing, communications and informatics (ICACCI)*. IEEE, 2017, pp. 1366–1371.

[19] K. S. Sahoo, A. Iqbal, P. Maiti, and B. Sahoo, "A machine learning approach for predicting ddos traffic in software defined networks," in *2018 International Conference on Information Technology (ICIT)*. IEEE, 2018, pp. 199–203.

[20] M. Yue, H. Wang, L. Liu, and Z. Wu, "Detecting dos attacks based on multi-features in sdn," *IEEE Access*, vol. 8, pp. 104 688–104 700, 2020.

[21] S. M. Mousavi and M. St-Hilaire, "Early detection of ddos attacks against sdn controllers," in *2015 international conference on computing, networking and communications (ICNC)*. IEEE, 2015, pp. 77–81.