ANTIDOSTE: detection and mitigation of network-based Denial-of-Service attacks for a location certification system

Pedro André Ferreira Teixeira

Abstract—**Denial-of-Service (DoS) attacks have a long history and there are many of them, ranging from exploits that crash individual devices to attacks that overwhelm the capacity of servers by having many clients issue a *barrage* of requests. These attacks target *availability* and deny access of rightful users to the on-line services they need to work and play.**

In this work we protect a *location certification system* from network-level distributed-denial-of-service attacks. We called our solution ANTIDOSTE – antidote for DoS – and it combines the efficiency of using *custom rules* to detect specific attacks with the effectiveness of using *Machine Learning* on traffic patterns to detect previously unknown attacks. We evaluated the solution on a test-bed representing all the network nodes of the system, spread across a city. The results show that ANTIDOSTE can detect DoS attacks, both known and unknown, and can mitigate them using virtual local area networks (VLAN), created dynamically using software-defined network (SDN) mechanisms.

Index Terms—**Network Security, Distributed Denial-of-Service, Software-Defined Networking, Deep Learning, Logistic Regression, Location Certification System**

I. INTRODUCTION

Denial-of-Service (DoS) attacks can make the network resources unavailable for their intended users, temporarily or indefinitely [4]. An important sub-type of DoS attack is the Distributed Denial-of-Service (DDoS), a coordinated DoS attack that is generated by using many compromised network hosts simultaneously [16].

To fight DoS attacks, first it is necessary to detect them and distinguish them from benevolent spikes due to a popularity surge or a seasonal effect on traffic. Second, it is necessary to mitigate the effects of the attack.

Known DoS attacks can be detected with pre-defined rules that are able to detect the pattern of the attacks. However, rules are not effective for unknown attacks. Machine Learning (ML) has been used to tackle this challenge [21]. ML models can be trained to recognize malicious/abnormal packets, likely to be part of a DoS attack.

Regarding DoS attack mitigation, one possible way to improve it is to use Software Defined Networking [11] (SDN) architecture, with separate data and control planes. When the monitoring and analysis detect attacks, the SDN controllers can trigger a response system. The SDN controller can define "drop" rules in the SDN switch so that the packets from the malicious host do not arrive to its target.

In this paper we propose a prototype solution for DoS detection and mitigation, ANTIDOSTE, i.e. antidote for DoS. We created it with the intention of protecting a specific application, CROSS [15], from DoS attacks. CROSS is a distributed location certification system where users are rewarded if they complete itineraries across the city using the mobile application developed for the Android operating system [17]. The CROSS application uses several techniques to verify the user presence at the locations and *prevent location spoofing* attacks; however, before this work, it lacked protections against DoS attacks, and if the system was made unavailable, users would not be able to collect their rewards and would become unsatisfied tourists. Therefore, ANTIDOSTE aims to provide a detection and mitigation mechanism for attacks on *availability*. It uses SDN and ML technologies in conjunction so that they can reinforce each other. The key contributions paper are:

- DoS attack detection and mitigation for the CROSS location certification system;
- Test-bed for DoS attacks on CROSS.

In terms of security properties [2], the objective of this work is to preserve *availability*; the focus is *not* on integrity and confidentiality, as the protection of these other properties are addressed in previous work [15]. Also, in this work we only focus on *flooding* attacks [6] that affect the data plane of the SDN architecture.

ANTIDOSTE was developed for and tested with the CROSS application, but its design is generic, and the solution can be adapted for other domains of applications.

The remainder of the paper is structured as follows: Section II start by presenting our network management, followed by work on DoS detection, ending with previously proposed ways to mitigate these attacks; Section III presents our proposed solution against DoS attacks; Section IV describes the test-bed and Section V presents the experiments we conducted and their results. Finally, Section VI states our conclusions.

II. BACKGROUND AND RELATED WORK

In this Section we start by presenting the target system to be protected by ANTIDOSTE. Next, we present the core network management mechanisms required for our solution and also give an overview of previous works on DoS attack detection and mitigation.

A. Target system

CROSS [15] is a location certification system for Smart Tourism [7]. In Smart Tourism clients use their personal devices to interact with existing or newly added infrastructure in emblematic city locations and record sensor data that can later be used to verify location claims. Therefore, in CROSS, the system operation starts when the tourist installs the smartphone application and signs up for an account. Before starting the trip, the application downloads the catalog of locations. During its use, the application logs visits to locations. The location sensing relies on Wi-Fi and leverages the regular scans already performed by the mobile operating system. At the end of the trip, the logging stops, the application submits the collected information to the server, and rewards will be issued, if the conditions have been satisfied.

The authors of CROSS use 3 strategies for the location proofs: Wi-Fi scavenging, Wi-Fi beacons and interactive kiosks. These strategies provide increasing security against *spoofing* and *Sybil* attacks [5]. However, they do not protect from DoS attacks.

B. Network infrastructure management

Software-defined anything (SDx) is recent technology trend that proposes that everything should be programmable, i.e., software should be "in command" of the hardware infrastructure. SDx includes Software-defined networking (SDN) [11], which is the most recognized technology in this trend. Its core feature is the separation of the control plane and data plane in the network. SDN allows the flexible control of network traffic. The devices in SDN are programmable, and thus the networks themselves are more dynamic, manageable, and cost-effective.

In SDN, network intelligence is logically centralized in software-based controllers (the control plane), and network devices such as OpenFlow Switches become simple packet forwarding devices (the data plane) that can be programmed via an open interface, the OpenFlow protocol [19]. Such SDN works as follows: an OpenFlow switch has one or more flow tables. These tables are used to control packets (e.g., "forward" or "drop") according to packet-handling rules, called the *flow rules*, and received from a centralized controller. Therefore, according to the controller policy that manages flow tables, the OpenFlow switch can act as a router, switch, or firewall, or exhibit similar functions that depended on the packet-handling rules. For example a VLAN[1] can be created through the SDN controller and the flow rules can be programmed, for that VLAN, in the SDN switch.

And so, with this idea of a centralized network control plane and the introduction of this new type of programmability to the network devices, which can streamline network management and enable run-time security strategies. SDN can rapidly react to network anomalies and malicious traffic, such as DoS attacks, by filtering out sources of attack and isolating parts of the network, if necessary.

C. Denial-of-Service detection

SDN provides various techniques for DoS attack detection [23]. For example, to perform DoS attack detection, Yin et al. [25] have proposed an algorithm that calculates the cosine similarity of the vectors of packets. It checks the incoming packets (`packet_in`) in each port of the boundary switches and then determines whether a DoS attack has occurred based on the value of the cosine similarity.

SDN and ML can be combined to deal with DoS attacks [20], [18]. In particular, Ravi et al. [21] have demonstrated the effectiveness of using SDN in conjunction with ML by crafting a learning-driven detection mitigation mechanism (LEDEM). The authors used a semi-supervised [21] ML model, the semi-supervised deep extreme learning machine (SDELM) model, which is a mixture of unsupervised, where unlabeled data is used, and supervised, where labeled data is used to train the detection of DoS traffic.

D. Denial-of-Service mitigation

The strategy to mitigate these attacks, in a SDN environment, is to set "drop" rules for the malicious packets as done in the work of Yin et al. [25]. When an attack is detected, a new "drop" rule is added to the flow table to drop all the packets originating from the malicious devices. But we need to consider that if an attacker has a large botnet [1], composed of many devices, setting individual rules for the malicious devices will saturate the limited flow table space in switch and lead to overflowing issues in the data plane of the SDN [22]. So, Ravi et al. [21] have proposed a mitigation strategy that prevents saturation; they create a VLAN in the switch, and then define "drop" rules for it and group all the malicious hosts in that VLAN. This is one of the strategies used in our proposed approach, and will be further explained in the next Section.

III. antiDoSte

antiDoSte is a prototype solution that was created with the intention of protecting CROSS system from DoS attacks. The strategy of the solution depends, mainly, on the use of SDN to provide deep packet inspection on the network controllers, so that necessary *features* from the packet are extracted to differentiate malicious from normal network traffic. In previous work, by Yin et al. [25], although the results of the proposed method were good, compared to previous methods, there is a weak point in the solution: the authors made the assumption that the generated packets to perform DoS attacks are always similar, which we cannot assume in a real-world scenario. The same thing happens to ML approaches since they are based on prediction, i.e. what the model decides the traffic is might not correspond to the reality; it is possible for the approach to not detect an attack or not detect it in reasonable time. Therefore, in antiDoSte we propose a hybrid solution that combines SDN and ML techniques to provide better DoS detection and mitigation. This way, it is possible to use the strong points of some of the detection techniques to make up the weak points of the other techniques.

A. Detection strategy

antiDoSte has three approaches that work together to detect DoS attacks, that are explained next.

[1]A VLAN is a Virtual Local Area Network. It is a broadcast domain that is partitioned and isolated at the data link layer.

1) Threshold-based detection: The first strategy sets a *Threshold* for traffic volume. This approach limits the number of packets a client device can send over the network. For example, the TCP-SYN flood attack that uses the 3-way handshake to flood the server with TCP-SYN packets. In a normal scenario, a client is expected to send an TCP-ACK packet after sending TCP-SYN packet, if there is no problem in the network. We can set a maximum threshold for the number of TCP-SYN packets send by the client without TCP-ACK packets. First, we set a predefined threshold N and then when an TCP packet is received on the controller we check if the packet contains an SYN or ACK flag. If the packet has a SYN flag, then the counter of that client is increased and if the packet has an ACK flag, the counter of that client is decreased by one. The reason why the counter is decreased by one and not reset to zero is because of the attacker might know of this mechanism beforehand and when the counter is approaching the predefined threshold he could just send TCP-ACK packet, resetting the counter, and continuing with the attack, making it possible for the attacker to counter this strategy. When the counter reaches the maximum threshold, the mitigation strategy is applied. Figure 1 shows the flow diagram of this approach.

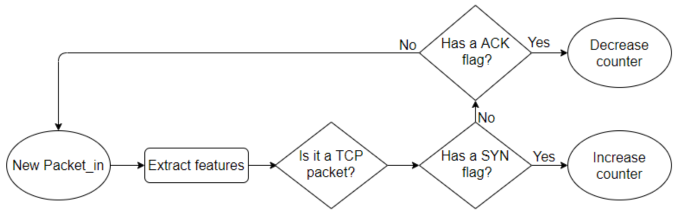


Fig. 1. Flowchart for Threshold-based attack detection.

However, despite being the fastest approach to detect an attack, as Section V-B will demonstrate, this technique requires a considerable knowledge of the attack, since we need to know a weak point that we can exploit to prevent the attack. Therefore, this approach is suitable for simpler attacks that are easier to find their weaknesses, such as the previously mentioned TCP-SYN flood attacks. The implementation of this approach, applied to TCP-SYN flood attacks, is shown in Algorithm 1.

2) Machine Learning-based detection: The second detection approach is based on ML techniques. In our solution we use two different models, Deep Learning (DL) and Logistic Regression (LR).

DL [12] has been previously applied to traffic classification [24]. In this work we use a DL model that was initially proposed by Yuan et al. [26] to detect DoS attacks. When training this model, we specialize it to detect certain attacks, i.e., we provide data for a certain type of attack so that it has high accuracy when detecting that attack. In this case, we chose the *ICMP flood* attack.

The DL approach is easier to train, in comparison with the Threshold-based approach, since we only need a dataset of the attack for training, while the previously presented

Algorithm 1: Threshold mitigation approach.

```
Result: Increase/Decrease counter from host or
        mitigate attack
$Threshold \leftarrow N$;
$tracker \leftarrow \{\}$;
if PacketIn.type != TCP then
    return
end
if PacketIn.flags != SYN or ACK then
    return
end
if PacketIn.flags == SYN then
    if PacketIn.scr not in tracker then
        tracker[PacketIn.scr] = 1 #src means packet
          source IP
    else
        tracker[PacketIn.scr] += 1
        if tracker[PacketIn.scr] == Threshold then
            Apply mitigation strategy
        end
    end
end
if PacketIn.flags == ACK then
    if PacketIn.scr not in tracker then
        tracker[PacketIn.scr] -= 1
    end
end
```

approach requires manual intervention to create or improve rules. However, the DL approach is the slowest, as Section V-B will demonstrate, and is more suited to detect more elaborated attacks that require a more complex analysis.

The logistic regression (LR) [9] model is trained to recognize normal behaviour of the traffic. In this case we use less entries from each attack, when comparing to the DL model, because we want the LR model to have a general knowledge about what is not normal traffic, in contrast to the DL model where we want the model to know the attack. Once the LR model detects unexpected behaviour, the mitigation strategy is applied. However there is also another purpose for this approach, which is to improve the previously mentioned detection approaches, i.e., Threshold and DL. When the LR model detects abnormality and the other two approaches do not, a log/dataset is generated from the packets originated by the attack to make the attack features known to the other two approaches. Once the log is generated, it is analysed by a network administrator to verify if it is really an attack, since it might be a peak in traffic that LR model does not recognize as normal traffic. If the traffic corresponds to an attack, we first apply the log/dataset to train the DL model, which is faster to prepare for attack detection, and so we have a more reliable way to detect an attack, than the LR model. Meanwhile, we keep studying the log to create new rules for the Thresholds, so we can detect the attacks faster than the DL approach.

Therefore, for the *DL* and *LR* approaches, represented in

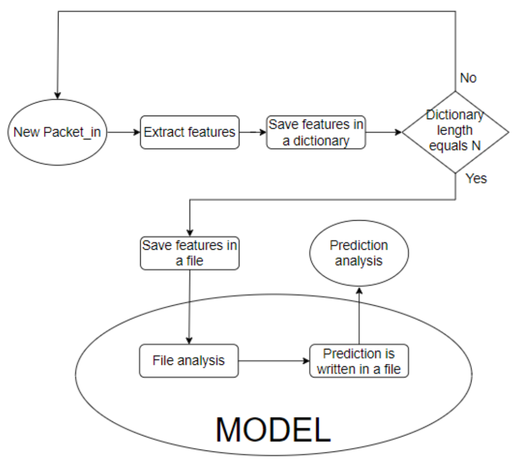


Fig. 2. Flowchart for the DL and LR attack detection.

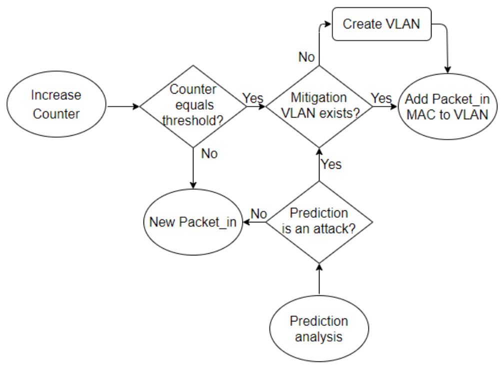


Fig. 3. Mitigation strategy for Threshold and DL approaches.

Figure 2, the features are saved in a dictionary where the index is the source MAC, and the value is a list of packet features. Once the controller has received a pre-determined number of packets from that host, represented as N in the figure, these values from the dictionary are then saved in a file so the model can determine if an attack is happening. After the model has stated its prediction, the application running the model creates a new file and sends it to the controller.

And so, ANTIDOSTE takes the advantage of the three detection approaches and creates a hybrid detection to analyse the traffic simultaneously so that the detected attacks get mitigated as soon as they discovered by each approach. In summary: Threshold applies rules for specific attacks, DL recognizes attacks, and LR recognizes deviations from normality.

B. *Mitigation strategy*

The ANTIDOSTE mitigation strategy is based on the strategy proposed by Ravi et al. [21], and is used for the *Threshold* and *DL* approaches. The controller starts by creating a VLAN, if it does not exist on the SDN switch in the same sub-network as the malicious host. Then the source and destination MAC addresses are added to that VLAN with "no flow" rules, so that the packets get dropped. There is only one VLAN per sub-network which is responsible for every malicious device in that sub-network, making only one entry in the switch flow table needed to drop all the packets from all the malicious devices in one sub-network. This is done to avoid overloading the flow tables of the switches, as mentioned in Section II-D. In Figure 3 it is represented the mitigation strategy after the increase of the counter for the *Threshold* approach and for the *DL* approach after the prediction analysis.

The LR detection approach also uses the previously mentioned mitigation strategy. However, first it needs to apply a new strategy to minimize the possibilities of what was detected was not an attack. It might be just a high peak in traffic, as mentioned previously, or bad prediction from the LR model. Therefore, this new strategy consists in setting a certain number of times a host can have an unexpected behaviour (e.g. three) and so once the LR model detects an abnormal behaviour from that host, it is recorded that the host has committed one fault. Whenever the host has committed three faults (the specific total of faults can be adjusted), the mitigation strategy is applied and the packets from that host are recorded into the log/dataset and dropped. However, it is possible that the unusual traffic is coming from a non-malicious host that just had, for example, a high peak in traffic for a longer period of time. In those cases, a network administrator is responsible for analysing the log/dataset, to remove the host from the VLAN and re-establish the flow rules to what they were before the mitigation. There is also the possibility that the host, after committing one or two faults, starts to behave normally. In these cases the number of faults can be decreased after a timeout or if the model detects that the host has behaved normally a number of times in a row. For example, the host has two faults, but the model detects that he had a normal behaviour two times in a row and so the number of faults is decremented by one. If this happens again then the host will not have more faults.

Also for the LR model, since the DL model is the slowest at detecting an attack, as Section V-B will show, we created a LR model which is faster than the DL model, and thus, in case of an attack, the model needs to detect the attack a predetermined number of times (three in our example), making the detection time become more relevant than the accuracy, being this the reason why we use LR model instead of DL model. Figure 4 represents the mitigation strategy for the *LR* approach, after the prediction analysis, where N is the number of faults a device can do and M is the number of consequent well behavior detected, from a device, before reducing the number of faults.

IV. TEST-BED

In this Section we describe the test-bed, based of the network supporting CROSS, that we used to conduct our evaluation of ANTIDOSTE, starting by first describing the tool we use to emulate CROSS then we mention the hardware components and software components we emulate in our test-bed, ending with a brief overview of the network topology.

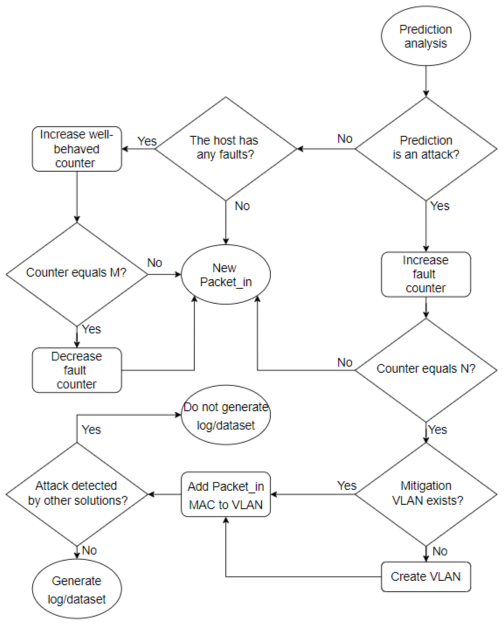


Fig. 4. Mitigation strategy for LR approach.

A. Network Emulator

Mininet [10] is an OpenFlow-based SDN emulator giving researchers an efficient way to test their SDN frameworks and measure their performance and reliability. The Mininet is an open source emulator written in the Python programming language. It is built over the Ubuntu Linux distribution. The elements of Mininet are organized into three main components: the *host*, which sends and receives the packets, the *switch*, which stores all the required rules to forward the packets to their destinations, and a *central controller* which handles the functionality of control and management operations in the network. Mininet supports different types of virtualized hosts, switches and controllers.

B. Software and Hardware

On the top of the hardware infrastructure we have a POX controller [8] which is Python-based open-source OpenFlow/SDN. POX is used for faster development and prototyping of new network applications. The controller comes pre-installed with the Mininet virtual machine. By using them you can turn OpenFlow devices into hub, switch, load balancer, firewall devices. The POX controller allows easy way to run OpenFlow/SDN experiments. POX can be passed different parameters according to real or experimental topologies, thus allowing experiments to be run on real hardware, test-beds or in the Mininet emulator. To generate the traffic from each host we use a Python library, called Scapy. It is a powerful interactive packet manipulation program that can forge or decode packets of a wide number of protocols, send them on the wire, capture them, match requests and replies.

This test-bed consists mainly of a server, 10 devices that can be Kiosks, Smart Space Managers (SSM) or Wi-Fi Access Point (AP), 6 OpenFlow-enabled switches and a SDN controllers. The emulation was done in a machine with a Intel Core i7-8750H CPU at 2.20GHz and 16GB of RAM.

C. Network topology

As we can see in Figure 5 there are 6 sub-networks, across the city of Lisbon, Portugal. In the emulated framework, 5 of them are tourism points with 2 devices, per sub-network, for proof location, and the last one is for the CROSS server. Each sub-network has a switch which is connected to every device in the sub-network, working as a router for that sub-network. Note that in Figure 5 there is a controller per sub-network, but they are the same controller. This controller is responsible for setting rules in the switch for forwarding or dropping packets. Also, in Figure 5 it is internet with lower-case “i” since CROSS is a private network.

D. Attack scenarios

Some possible attack scenario in our test-bed are: An attacker uses a tourism based device, i.e. the Kiosk, Smart Space Manager or Wi-Fi Access Point, present in one sub-network to perform a DoS aimed at the application server; Perform a DoS from one tourism based device to another in the same sub-network, for example in Figure 5 in Jerónimos one Kiosk could attack a Wi-Fi Access Point, or in different sub-networks, for example a Kiosk in Sé could attack a Kiosk in Jerónimos; All the tourism based device in a sub-network can be used to perform a DDoS attack to the application server or a tourism based device present in other sub-network; And it is possible to use all the tourism based devices present in the test bed to perform a DDoS attack to the application server.

V. Evaluation

To evaluate our solution, we used the test-bed. The CROSS server was the target of the DoS attack and it is located in the Alvalade sub-network, as represented in Figure 5. The types of DoS tested in our experiments were the TCP-SYN flood and ICMP flood, which are known by the Thresholds or the DL approaches, and the UDP flood which is considered an unknown attack. The remainder of this Section will disclose the approaches we followed for the evaluation of each detection strategy, as well as the reason why we chose those approaches.

A. Approach

For the Thresholds detection strategy we evaluate the time it takes to detect a DoS attack. This metric is important to evaluate how well the detection strategy behaves when an attack is happening. For the DL and LR approaches we also evaluate the time it takes to detect DoS attack, for the same reason of the Thresholds approach, and we evaluate metrics such as: *Accuracy* of the model, to see how well the solution predicts the status of the network (benign or under attack) correctly; *True Positive rate/Recall*, to verify how well the solution can detect when an attack is occurring; *False Positive rate*, to check how well the solution can see that an attack is not happening; *Precision*, to examine at what precision (i.e., DoS is not marked as benign) the model predicts benign

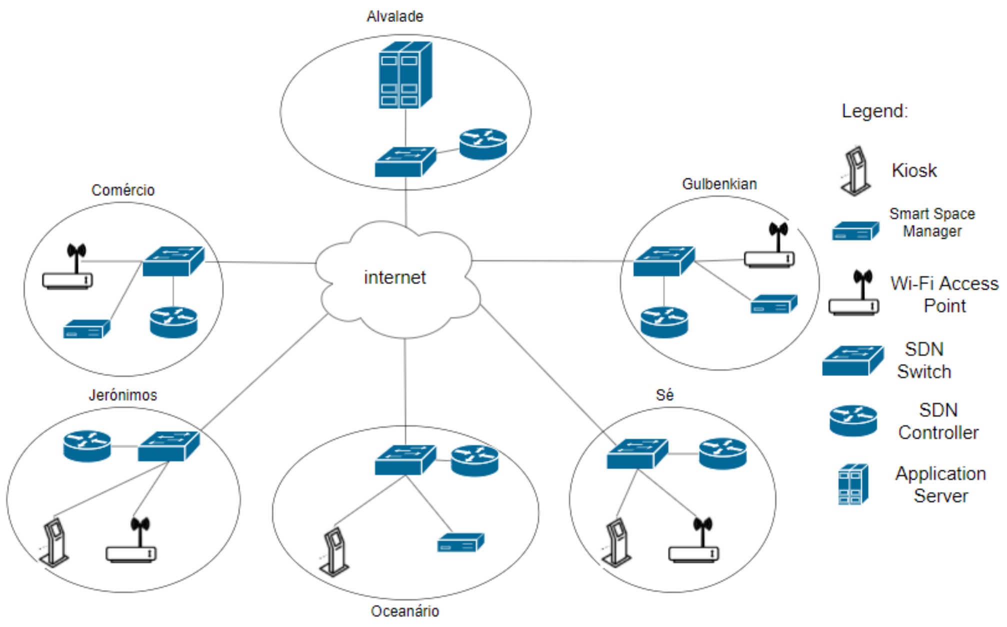


Fig. 5. CROSS network topology.

activity in the network; *F-score*, to help us find out the optimum threshold between Recall and Precision in the model, and so that the model can be compared with other classifiers in future work. These metrics were evaluated for the DL and LR approaches, since these approaches are based on predictions and not on rules, as with the Thresholds.

The deployment of the models was done in an application running alongside the controller. The controller is responsible for extracting the *features* from the received packets and then sending these features to the application where the model is running. The extracted features can be seen in Table I, alongside some examples and their types. Both models analyse a window of traffic (multiple packets at the same time) instead of analysing a single packet, making values such as the time between packets more meaningful, which is an important feature to detect the flooding attacks. For the DL model, these metrics were evaluated, with a dataset generated from normal traffic of the test-bed and a dataset, BUET-DDoS2020 [14], for a specific type of DoS attack, namely ICMP flood attack, both with 10 000 entries. For the LR model we used a dataset generated from normal traffic of the test-bed, 50 000 entries, and all the attacks present in the test dataset from BUET-DDoS2020, TCP-SYN flood, UDP flood, HTTP flood, DNS flood and ICMP flood, being approximately 10 000 entries tested for each attack.

TABLE I
FEATURES EXTRACTED.

Field	Field Example	Field Type
frame number	1	Numerical
frame.len	805	Numerical
ip.protocol	tcp	Text
ip.ttl	127	Numerical
tcp.srcport	2090	Numerical
tcp.dstport	443	Numerical
tcp.syn	1	Numerical
tcp.ack	0	Numerical
tcp.rst	0	Numerical
time	0	Numerical
http.version	1.1	Float
http.type	GET	Text
http.request	synchronize with server	Text

B. Results

For measuring the time each approach takes to detect and mitigate the attack and the bandwidth impact from the normal, attack and mitigated traffic, we simulated a DoS attack aimed at the server, while normal traffic is circulating. We captured the traffic in the Alvalade sub-network of CROSS between the server and the SDN switch. Figure 6a shows what are the normal values for the bandwidth, from the test-bed.

1) TCP-SYN flood: The TCP-SYN flood attack will be detected by the Thresholds approach. The time of detection depends on the packet rate of the attack and the predefined Threshold. This is because the detection happens when the attacker sends pre-defined number of packets, in our experiments it is the number of TCP-SYN packets. And so, if the packet rate is low, the detection will take more time, also, the higher the threshold the more time it takes to mitigate. But of course, if the packet rate is low the impact on the victim server will be less than if the packet rate was higher, and so the attacker has little interest in low packet rates, for flooding attacks. In addition, we cannot set the threshold too low because this might cause that legitimate user will not be able to communicate with the server when the network is saturated with other requests from legitimate clients. As

TABLE II
MACHINE LEARNING MODELS EVALUATION METRICS.

Model	Accuracy	True Positive Rate	False Positive Rate	Precision	F1-score
DL	99.97%	99.95%	100%	100%	99.97%
LR	74.97%	59.90%	89,27%	38,90%	47.17%

mentioned before, the attack we use for this experiment is the TCP-SYN flood attack and it was performed with a Python library named Scapy [3]. In Figure 6b, we start the TCP-SYN flood attack at 5 seconds, which is represented in the graph by the spike. However, since we do the attack with high packet rate, and the Threshold is 20, it is mitigated almost instantly (after the first 20 TCP-SYN packets have passed through the controller, the mitigation strategy is applied), and so the attack does not have an impact like what will occur in the other approaches. Note that we set the Threshold to 20 as an example of what a real-world scenario could be. There was no investigation to determine what is the optimal Threshold since the emulation of the network is done in the same machine, without packet loss. In this case, the optimal Threshold for this network was 2, which does not make sense in a real network.

2) ICMP flood: The DoS attack detected by the DL approach was an ICMP flood, also performed by using Scapy. The effect on our solution can be seen in Figure 6c. We started the attack around the 5 second mark and it is mitigated at 10 seconds (black arrow), making our solution able to detect and mitigate the attack in approximately 5 seconds.

3) UDP flood: The LR approach was used to detect UDP flood attack, which was simulated with the hping3 tool [13]. As stated in Section III-B, we need to predefined a value, for the mitigation strategy, that is the number of times an abnormality needs to be detected so the traffic is mitigated, and so we predefined that value to be 3. As you can see from Figure 6d, the attack started at 1 seconds and is mitigated at 2 seconds, taking about 1 second for the LR model to detect and mitigate the attack (black arrow), being faster than the DL model despite having to pass through 3 verifications.

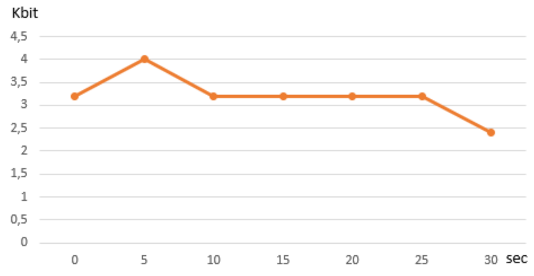
C. Discussion

By analyzing the results, we can conclude that the detection strategy based on SDN rules (Thresholds) is the fastest at detecting DoS attacks, but to use this approach we need to have considerable knowledge of the attack behaviour so we can know, if possible, which features, of the attack, we can explore to stop it. The DL approach is the slowest but is the easiest to implement since we only need data from the attack to train the model, not requiring much knowledge, as the Thresholds strategy needs about the attack to detect it, being suitable for cases where the mentioned attack "weakness" does not exist. Also, in Table II we can see the performance results of the DL and LR models. As expected, the DL model is much better than the LR model, not only because of the DL models have better accuracy than LR models, but also because the DL model that we evaluated, is specialized to detect specific attack, in this experiment ICMP flood attack, while the LR model was trained to detect abnormal behavior which requires much more data and knowledge of the network where the model is deployed. This is the reason why we apply a different mitigation strategy for the LR approach. In addition, these results also demonstrate why we cannot deploy the behavior analysis strategy by itself, since most of the attacks could pass without being detected, and so, this strategy serves more of a support to the other two, by providing the data gathered from the attack to improve them. However, it is a great addition to the other two strategies since it can help with the development of the prototype solution, by improving the other two solutions with logs/datasets created from abnormal traffic.

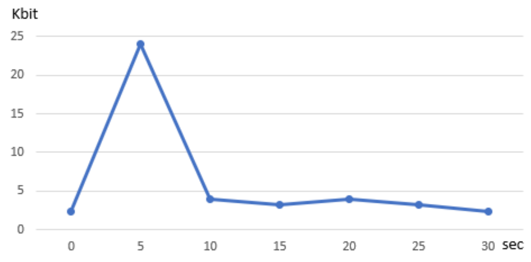
VI. CONCLUSION

We developed ANTIDOSTE to protect a specific location certification system from Denial-of-Service (DoS) attacks. ANTIDOSTE uses Software Defined Networking (SDN) techniques in conjunction with Machine Learning (ML) techniques to detect DoS attacks, making it a hybrid solution that combines benefits of different technologies. By using more than one approach to detect attacks it is possible to cover the weaknesses of some detection strategies. ANTIDOSTE has a semi-automatic improvement thanks to the Logistic Regression (LR) model detection approach that is responsible for generating logs/datasets from the detected unexpected behaviour of the attacks. These logs/datasets are analysed by a network manager to verify if they represent an attack, and if so, they can be used to improve the detection, with a manual rule for the *Thresholds* approach or with automatic training using DL.

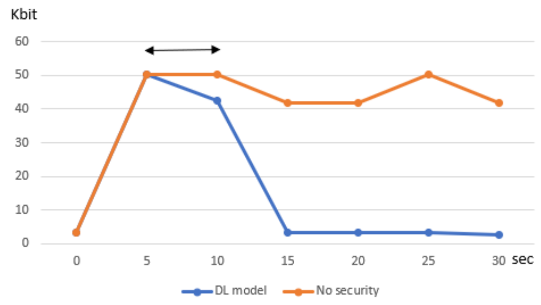
In this work we demonstrated the capabilities of our prototype solution for a specific use case, therefore, we plan to test our solution in different systems to prove it is a general solution, as we designed it to be. Moreover, we want to improve the LR mitigation strategy. The original idea was to limit the traffic bandwidth to a predefined value that would represents the maximum bandwidth a host should have during a normal behaviour and prioritise the traffic from other hosts in the same sub-network. However, this was not possible due to the limitation of the OpenFlow protocol version that POX controller supports, 1.0, in which is not possible to set, dynamically, a limit to the bandwidth in the SDN switches. In contrast, version 1.2 of OpenFlow protocol allows for maximum rate setting, and this will be used in future versions. Also, we aim to improve our solution to detect more and different types of DoS/DDoS attacks and improve the used ML models so attacks can be detected better and faster.



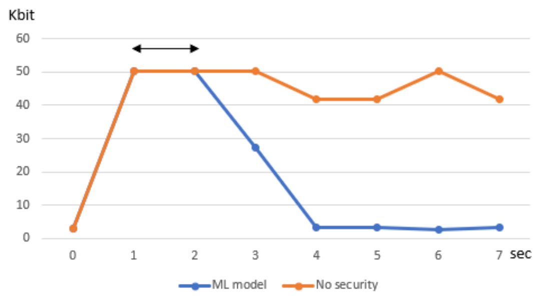
(a) Normal bandwidth.



(b) TCP-SYN flood with Threshold detection.



(c) ICMP flood with DL model detection.



(d) UDP flood with LR model detection.

Fig. 6. Network traffic volume for different configurations of ANTIDoSTE.

Acknowledgements

This work was supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) with reference UIDB/50021/2020 (INESC-ID) and through project with reference PTDC/CCI-COM/31440/2017 (SureThing).

References

[1] Angrishi, K.: Turning internet of things (iot) into internet of vulnerabilities (iov): Iot botnets. arXiv preprint arXiv:1702.03681 (2017)

[2] Avizienis, A., Laprie, J.C., Randell, B., Landwehr, C.: Basic concepts and taxonomy of dependable and secure computing. IEEE Transactions on Dependable and Secure Computing **1**(1), 11–33 (Jan 2004)

[3] Biondi, P., community., T.S.: (2021), https://scapy.net/

[4] Chen, W., Ding, D., Dong, H., Wei, G.: Distributed resilient filtering for power systems subject to denial-of-service attacks. IEEE Transactions on Systems, Man, and Cybernetics: Systems **49**(8), 1688–1697 (2019)

[5] Douceur, J.R.: The sybil attack. In: International workshop on peer-to-peer systems. pp. 251–260. Springer (2002)

[6] E Chou, R.G.: Distributed Denial of Service (DDoS). O'Reilly (2018)

[7] Gretzel, U., Sigala, M., Xiang, Z., Koo, C.: Smart tourism: foundations and developments. Electronic markets **25**(3), 179–188 (2015)

[8] Kaur, S., Singh, J., Ghumman, N.S.: Network programmability using pox controller. In: ICCCS International Conference on Communication, Computing & Systems, IEEE. vol. 138, p. 70. sn (2014)

[9] Kleinbaum, D.G., Dietz, K., Gail, M., Klein, M., Klein, M.: Logistic regression. Springer (2002)

[10] Lantz, B., Heller, B., McKeown, N.: A network in a laptop: rapid prototyping for software-defined networks. In: Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks. pp. 1–6 (2010)

[11] Larry L. Peterson, C.C., Brian O'Connor, T.V., Davie, B.: Software-Defined Networks: A Systems Approach. Systems Approach LLC (2020)

[12] LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. nature **521**(7553), 436–444 (2015)

[13] Limited, O.S.: hping3 (2021), https://tools.kali.org/information-gathering/hping3

[14] M. Hasan, S.I.: Buet-ddos2020 (2020), https://data.mendeley.com/datasets/bzgf9r36kp/2

[15] Maia, G.A., Claro, R.L., Pardal, M.L.: Cross city: Wi-fi location proofs for smart tourism. In: International Conference on Ad-Hoc Networks and Wireless. pp. 241–253. Springer (2020)

[16] Matta, V., Di Mauro, M., Longo, M.: DDoS attacks with randomized traffic innovation: Botnet identification challenges and strategies. IEEE Transactions on Information Forensics and Security **12**(8), 1844–1859 (2017)

[17] Mednieks, Z.R., Dornin, L., Meike, G.B., Nakamura, M.: Programming android. " O'Reilly Media, Inc." (2012)

[18] Niyaz, Q., Sun, W., Javaid, A.Y.: A deep learning based ddos detection system in software-defined networking (sdn). arXiv preprint arXiv:1611.07400 (2016)

[19] Nunes, B.A.A., Mendonca, M., Nguyen, X.N., Obraczka, K., Turletti, T.: A survey of software-defined networking: Past, present, and future of programmable networks. IEEE Communications surveys & tutorials **16**(3), 1617–1634 (2014)

[20] Polat, H., Polat, O., Cetin, A.: Detecting ddos attacks in software-defined networks through feature selection methods and machine learning models. Sustainability **12**(3), 1035 (2020)

[21] Ravi, N., Shalinie, S.M.: Learning-driven detection and mitigation of DDoS attack in iot via sdn-cloud architecture. IEEE Internet of Things Journal **7**(4), 3559–3570 (2020)

[22] Shin, S., Gu, G.: Attacking software-defined networks: A first feasibility study. In: Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking. pp. 165–166 (2013)

[23] Wang, J., Wen, R., Li, J., Yan, F., Zhao, B., Yu, F.: Detecting and mitigating target link-flooding attacks using sdn. IEEE Transactions on Dependable and Secure Computing **16**(6), 944–956 (2018)

[24] Wang, Z.: The applications of deep learning on traffic identification. BlackHat USA **24**(11), 1–10 (2015)

[25] Yin, D., Zhang, L., Yang, K.: A DDoS attack detection and mitigation with software-defined internet of things framework. IEEE Access **6**, 24694–24705 (2018)

[26] Yuan, X., Li, C., Li, X.: Deepdefense: identifying DDoS attack via deep learning. In: 2017 IEEE International Conference on Smart Computing (SMARTCOMP). pp. 1–8. IEEE (2017)