Real-time Pipeline Reconfiguration of P4 Programmable Switches to Efficiently Detect and Mitigate DDoS Attacks

Amir Al Sadi*, Marco Savi‡, Davide Berardi*, Andrea Melis*, Marco Prandini*, Franco Callegati*
*Department of Computer Science and Engineering - DISI - University of Bologna (Bologna, Italy)
‡Department of Informatics, Systems and Communication - DISCo - University of Milano-Bicocca (Milano, Italy)

***Abstract*—In this work we demonstrate the integration of P4 enabled switches with high level AI techniques with the aim to improve efficiency and performance of DDoS detection and mitigation. Powerful ML-based strategies are adopted only when a suspicious behaviour is occurring in the network, and its activation is triggered by a coarser-grained and lightweight strategy fully executable in the data plane.**

I. Introduction

Distributed Denial of Service detection and mitigation is still a very complex topic to deal with. First of all prompt and automated detection [1] is required, followed by strategies to implement effective countermeasures, all of which is also not trivial. In this work we combine AI and Software Defined Networking to achieve both detection and mitigation by network reconfiguration in an original way.

The use of Machine Learning techniques for the reconfiguration of SDN networks is widely used in today literature, both for cybersecurity-oriented solutions for attack mitigation [2], [3], [4], and for QoS guarantee by redistributing the workload traffic [5], [6], [7]. The main issue in using AI for DDoS detection is the challenge to provide a detailed and meaningful traffic analysis for anomaly detection at line rate. This is the issue we address in this demonstration, which has the goal to show the effectiveness of adopting P4-based programmable data planes and P4Runtime for real-time switch reconfiguration, combined with fine-grained Machine Learning (ML)-based logic when an abnormal behaviour is identified.

II. Context

The demonstration proposed in this work is built exploiting P4 programmable switches and P4Runtime to master the reconfiguration of such switches in real time.

The P4 language allows to program in detail the forwarding behavior of P4 enabled switches (also called the forwarding *pipeline*), as well to analyse, modify and process packets at line rate in the switches. As a companion to P4 the P4Runtime API has been developed with the aim to:

- enabling run-time control of P4-defined switches;
- defining program-independent interaction (the API does not change if the P4 program is modified);
- enabling to push a new P4 program without recompiling the switch software stack.

It adheres to a client-server model; the server resides in the data plane, integrated within the switch, while the client can be integrated into a local or remote control plane component which interacts with the server to load the pipeline/P4 program, write and read pipeline state (e.g. table entries, meters, groups, etc.) and sends/receives packets.

To achieve the goals of this demonstration we implemented a control plane components that integrates the P4runtime client. It behaves as a high level controller which has access to a *library* of switch programming options and can install and/or modify them in real time according to the needs.

In particular with reference to DDoS this capability to change the P4 pipeline at run-time can improve:

1) detection effectiveness, since it is possible to introduce specific policies on the data plane to counteract specific attacks while they are observed at run-time;
2) reaction cost and efficiency, since it is possible to restore the network pipeline and reset the standard forwarding behaviour after the attack is mitigated.

III. Demonstration use case description and architecture

The specific use case presented in this demo is designed to provide a solution to detect and mitigate network DDoS attacks as follows:

- the P4 switches are enabled at start-up with a P4 program that can calculate some simple statistics aiming at providing a coarse detection capability of network anomalies;
- if this P4 program detects some network flow which sounds suspicious it triggers the control plane that will use the P4runtime interface to re-program the switch in order to collect fine grained statistics that are fed back to the control plane;
- an AI engine based on CNN will retrieve the fine grained statistics from the control plane to provide a deeper analysis of the suspicious flow;
- if the AI engine confirms a malicious behavior the control plane installs a new P4 traffic dropping rule that stops the malicious flow, thus enabling DDoS mitigation.

Key to achieve the aforementioned goals are:

1) *Real-time data plane pipeline reconfiguration* This mechanism is enabled by P4Runtime and is used to modify at run-time the behavior of the switch.

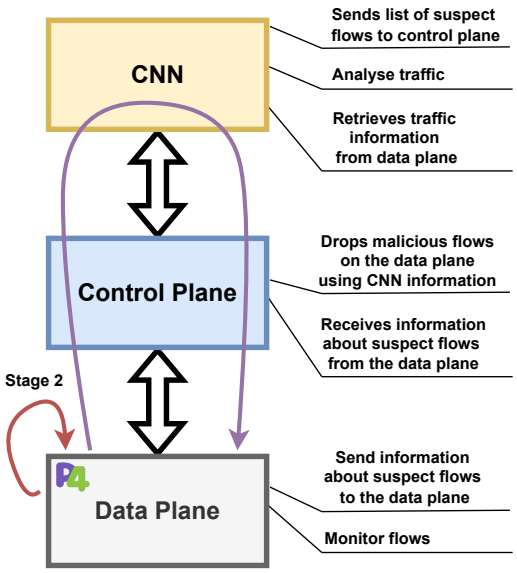

Fig. 1. Architectural blocks of the prototype

2) *Coarse traffic flow anomaly detection* A basic sketch-based strategy is used to detect whether a possible DDoS attack is occurring.
3) *Interaction between programmable data plane and a supervised CNN engine:* This is achieved by extracting aggregated features on packets/flows from the data plane by means of a custom P4 program and then feed them to an external AI engine (CNN) to infer with more accuracy abnormal behaviours in the data plane.

Figure 1 shows the three architectural components of the system, the Data Plane, the Control Plane and the CNN and their logical interaction.

A. Coarse anomaly detection in the data plane

The coarse anomaly detection mechanism in the data plane uses a strategy which relies on *flow classification* based on the packets that traverse the switch. In particular an Asymmetric flow detection algorithm is used. This technique is defined by observing the behaviour of a DDoS attack, which usually generates a large number of packets - e.g. SYN attacks - that try to establish a connection without being successful. Asymmetric flow detection counts and compares the number of packets sent from a source IP to a destination IP (tuple $\{$*source IP, destination IP*$\}$ is identified as flow) and the ones sent in the opposite direction. The difference between the number of packets in the two directions in a given time interval provides the *asymmetry rate* between the flows. If the absolute value is greater than a specific threshold, the flow is labelled as *suspect* and stored in a P4 register.

From the implementation point of view the algorithm exploits *Count-min sketch*, a data structure which is used to estimate the number of occurrences in a data stream, here used to sum up the number of packets per flow in any direction. We adopt this compact data structure because storing per-flow counters in P4 registers would be intractable from a memory consumption perspective. The sketch-based asymmetry flow detection strategy is used to identify suspect DDoS flows. From now we will refer to this combined strategy as *Asymmetric Count-min Sketch*.

B. Fine grain detection of DDoS flows

The AI engine that will perfoorm finer data flow anomaly detection is based on a Convolutional Neural Network similar to what proposed in [8]. It is designed to aggregate and process features coming from the control plane. The output of the neural network is a value indicating the probability of any flow being part of a DDoS attack. If the percentage is above 50%, the engine triggers a warning and forwards the flow's packets to the control plane, which adds a matching rule to the data plane that drops the flow (*mitigation* operation).

IV. Performance Evaluation

We argue that the architecture proposed and the modules developed for this demonstration may achieve realistic performance for the use case of DDoS, both from a resource consumption and from a detection capability point of view. The use case was simulated in a virtual environment: the single-switch emulated network runs on Mininet [9], and BMv2 [10] is the considered P4 softwared switch target. The CNN is developed using the Keras API [11] on top of TensorFlow [12]. All the tests were run in a Ubuntu 20.04 LTS PC with a 16GB of RAM and an i5 8th generation Intel processor.

A. Resource consumption

We designed two tests to monitor the memory and CPU usage of the emulated switch considering an attack lasting 6 minutes, taken from the CICIDS2017 [13] dataset. We decided to cap the virtual bandwidth limited to 30Mbps, to avoid bottlenecks due to BMv2. We used Tcpreplay [14] to simulate a 100Mbps DDoS attack mixed with normal traffic. We evaluated the percentage of memory and CPU used by the CNN to analyze traffic in two different cases:

- Analyses of the entire traffic (labeled as "Traffic not filtered"). This is our baseline, considering a scenario where a data-control plane interaction constantly occurs.
- Analyses of traffic filtered by Stage 1 (labeled as "Filtered traffic") only. This is exactly what our two-stage proposed strategy enables.

As we can see from Figure 2, the memory consumed by the CNN process is way less when only the filtered traffic is analyzed. This is only possible if our two-stage strategy is considered, i.e., Stage 2 is activated and the custom P4 pipeline is installed in the data plane, allowing data aggregation. Figure 3 shows instead how CPU spikes last around twice in the case of unfiltered traffic, consuming on average more CPU while the attack is taking place. This means that our two-stage strategy saves CPU consumption with respect to the baseline. Moreover, during this tests we registered around 0.25% losses for about 1 second when the pipeline needed to be re-configured, which we consider as acceptable.

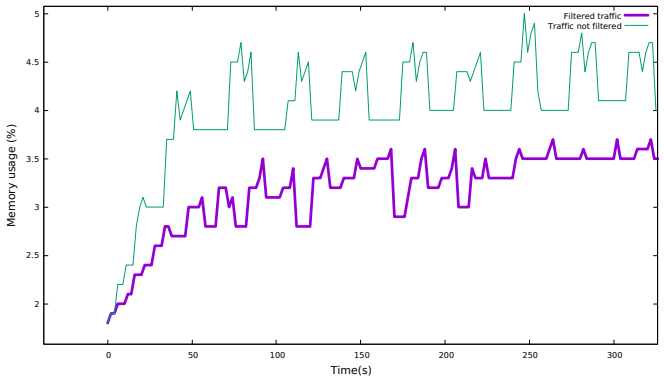

Fig. 2. Memory usage (%) throughout the attack

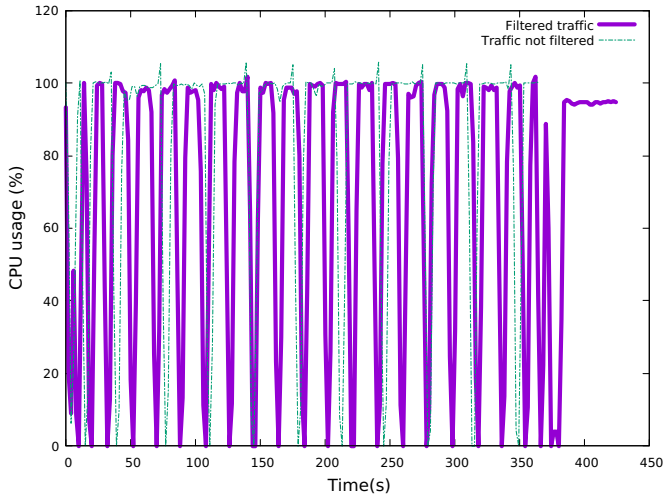

Fig. 3. CPU usage (%) throughout the attack

B. Detection capability

We designed three tests that use different asymmetry rates thresholds: 250, 500 and 1000 packets. Given the dataset (57 malicious IPs - true positives - and 512 benign - true negatives), we focused on the DDoS detection strategies of stages 1 and 2 and compared the false positives, false negatives, precision, recall, F1 score. Every stage lasts 30 seconds.

Table I shows that carefully choosing the asymmetry rate threshold is vital to be able to avoid false negatives. In fact, ideally stage 1 should avoid any false negative, while false positive can be tolerated. The adoption of the heavier strategy using CNN in stage 2 is then crucial to identify any false positive occurring in stage 1 and efficiently prune them, as well as to identify malicious sources. In the case of an asymmetry rate threshold of 1000 packets, we can see how 3 IPs (5% of the 57 malicious IPs) were not detected by the scheme. On the other hand, setting the threshold to 250 resulted in more false positives and more noise in the CNN, which was not able to detect two malicious IPs. Overall, both stage 1 and 2 have lead to an F1 score of at least 88%, and the F1 is higher when the CNN is involved.

Stage	THR	FN	FP	Precision	Recall	F1 Score
1 (In-line)	250	0	14	80%	100%	88%
2 (With CNN)	250	0	2	96%	100%	97%
1 (In-line)	500	0	7	89%	100%	94%
2 (With CNN)	500	0	0	100%	100%	100%
1 (In-line)	1000	3	6	90%	95%	92%
2 (With CNN)	1000	3	0	100%	95%	97%

TABLE I
DETECTION RATE COMPARISON WITH DIFFERENT THRESHOLD.

V. CONCLUSION

In this paper we demonstrated that it is possible to exploit real time data plane re-configuration to enhance detection and mitigation of DDoS attacks.

These results are achieved with P4 as a device programming language, with the P4Runtime protocol as device re-configuration protocol and with a controller developed on purpose; a combination that enables the re-configuration of the data plane pipeline at run-time.

REFERENCES

[1] A. Melis, D. Berardi, C. Contoli, F. Callegati, F. Esposito, and M. Prandini, "A policy checker approach for secure industrial sdn," in *2018 2nd Cyber Security in Networking Conference (CSNet)*, 2018, pp. 1–7.
[2] A. Melis, S. Layeghy, D. Berardi, M. Portmann, M. Prandini, and F. Callegati, "P-scor: Integration of constraint programming orchestration and programmable data plane," *IEEE Transactions on Network and Service Management*, vol. 18, no. 1, pp. 402–414, 2020.
[3] D. Ding, M. Savi, F. Pederzolli, M. Campanella, and D. Siracusa, "In-network volumetric ddos victim identification using programmable commodity switches," *IEEE Transactions on Network and Service Management*, vol. 18, no. 2, pp. 1191–1202, 2021.
[4] A. da Silveira Ilha, A. C. Lapolli, J. A. Marques, and L. P. Gaspary, "Euclid: A fully in-network, p4-based approach for real-time ddos attack detection and mitigation," *IEEE Transactions on Network and Service Management*, vol. 18, no. 3, pp. 3121–3139, 2020.
[5] H. Song, S. Guo, P. Li, and G. Liu, "Fcnr: fast and consistent network reconfiguration with low latency for sdn," *Computer Networks*, vol. 193, p. 108113, 2021.
[6] A. Destounis, S. Paris, L. Maggi, G. S. Paschos, and J. Leguay, "Minimum cost sdn routing with reconfiguration frequency constraints," *IEEE/ACM Transactions on Networking*, vol. 26, no. 4, pp. 1577–1590, 2018.
[7] S. Troia, R. Alvizu, and G. Maier, "Reinforcement learning for service function chain reconfiguration in nfv-sdn metro-core optical networks," *IEEE Access*, vol. 7, pp. 167 944–167 957, 2019.
[8] R. Doriguzzi-Corin, S. Millar, S. Scott-Hayward, J. Martínez-del Rincón, and D. Siracusa, "Lucid: A practical, lightweight deep learning solution for ddos attack detection," *IEEE Transactions on Network and Service Management*, vol. 17, no. 2, pp. 876–889, 2020.
[9] K. Kaur, J. Singh, and N. S. Ghumman, "Mininet as software defined networking testing platform," in *International conference on communication, computing & systems (ICCCS)*, 2014, pp. 139–42.
[10] P. L. Consortium *et al.*, "p4lang/behavioral-model," 2019.
[11] N. Ketkar, "Introduction to keras," in *Deep learning with Python*. Springer, 2017, pp. 97–111.
[12] M. Abadi, "Tensorflow: learning functions at scale," in *Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming*, 2016, pp. 1–1.
[13] "Cicids2017 dataset," *https://www.unb.ca/cic/datasets/ids-2017.html*.
[14] A. Turner, "Tcpreplay," *http://tcpreplay. synfin. net/trac/*, 2011.