HALIDS: a Hardware-Assisted Machine Learning IDS for in-Network Monitoring

B. Brandino[*†], E. Grampin[*], K. Dietz[‡], N. Wehner[‡], M. Seufert[§], T. Hoßfeld[‡], P. Casas[†]
[*]INCO–FING, Universidad de la República, [†]AIT Austrian Institute of Technology
[‡]University of Würzburg, [§]University of Augsburg

***Abstract*—Early decision-making at the network device level is crucial for network security. This entails moving beyond traditional forwarding functions towards more intelligent network devices. Integrating Machine Learning (ML) models into the data plane enables quicker processing and reduced reliance on the control plane. This paper explores the development of a ML-driven Intrusion Detection System (IDS) where network devices autonomously make security decisions or defer to an expert Oracle, relying on in-band and off-band traffic analysis. Programmable devices, such as those using P4, are essential to enable these functionalities and allow for network device re-training to adapt to changing traffic patterns. We introduce HALIDS, a prototype for in-band ML-IDS using P4, complemented with off-band Oracles which support in-network ML-driven classification with more confident classifications, targeting an active learning logic for more accurate in-band analysis. We implement HALIDS using the open source software switch BMv2, and show its operation with real traffic traces publicly available.**

***Index Terms*—In-network machine learning; Programmable data plane; P4; Active learning; Oracles**

I. Introduction

The ever-growing volume of traffic in modern networks motivates the utilization of machine learning (ML) in networking [1]. When it comes to network monitoring for cybersecurity, a promising idea is to do traffic classification as early as possible, directly within network devices. Resources in network devices have been traditionally constrained, encompassing limitations in terms of memory, processing capacity, available operations, and more. Consequently, these devices are traditionally treated as “dumb” from a traffic monitoring perspective, performing only the essential functions required for the network to operate. The emergence of new data plane architectures raises the hope that network devices will perform functions beyond simple traffic forwarding. By doing so, the burden on the control and management planes is alleviated, and a portion of the processing is decentralized. Additionally, processing within the network device occurs more expeditiously, reducing the need for offloading to the control plane. Network programmability entails the ability to specify and modify algorithms in both the control and the data plane [2].

This paper involves the creation of an Intrusion Detection System (IDS), wherein the network device can make a decision with an appropriate level of confidence, or delegate this decision to an expert (oracle). Both the network device and the oracle employ a machine learning model for traffic classification. Considering programmable network devices, P4 (Programming protocol-independent packet processors) [3] stands as the most widely adopted programming language for the data plane and is the one used. This also provides us with the ability to re-train the network device, making it adaptable to changes in traffic patterns.

Within its limited resources, the network device is intended to make a prompt decision determining whether a flow is malicious or not. The concept revolves around leveraging the limited processing capabilities and memory of these devices to quickly process traffic, identifying packets that are evidently malicious. In instances where the decision carries a high level of confidence, actions are taken accordingly. Conversely, when the confidence in the decision is low, the responsibility is deferred to an expert (the oracle) for further assessment. The integration of machine learning into this concept is proposed, involving a simple model on the network device, reflective of its constraints, and deploying a more robust machine learning model on the oracle to make better decisions. In the case of the device, the model would normally be trained with fewer data than the oracle's model due to privacy reasons. It is crucial to find a ML model for the network device (the switch) that is not only straightforward but also naturally aligns with the programmable match-action pipeline. In line with the related work, Random Forests (RF) easily map to the match-action pipeline by associating each level of the tree with a table. Ideally deployed on a server, the oracle's model can not only be more complex but also trained with all available data to enhance the accuracy of its predictions.

Finally, it is proposed to re-train the network device model. This crucial step allows adaptability to changing network traffic. Once a designated metric is reached, such as after offloading a specific number of packets, the network device will undergo re-training using decisions made by the oracle. Consequently, as the traffic pattern evolves, the switch is expected to reach decisions that may not be sufficiently reliable. By re-training with more accurate decisions from the oracle, the network device aims to dynamically adapt to the evolving traffic conditions. Since the match-action pipeline associates each level of the tree with a table, re-training the switch simply involves rewriting the tables from the control plane.

II. Related Work

Several papers addressing different parts of the raised issue have been found, all using P4. Firstly, SwitchTree [4] proposes the integration of RF into the data plane for abnormal traffic

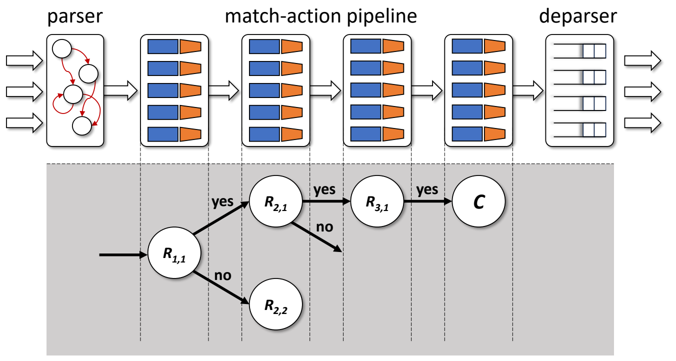


Fig. 1. An instance embedding of a tree within the Random Forest deployed in the switch [4]. $R_{i,j}$ represents the decision rule at node j at depth i.

identification. It extracts flow-level features with early detection (calculating the features in each packet) and incorporates RF as tables. pForest [5] introduces a similar idea, but various RF are trained for different phases of the flow, additionally proposing a confidence percentage for decision-making. CML-IDS [6], the work most similar to ours, proposes an RF in the data plane, and an oracle used to provide more accurate decisions in case of a lack of confidence in the switch's decision. CML-IDS has a strong focus on providing more powerful algorithms in the oracle but does not involve switch re-training. The aforementioned solutions are all implemented in software. MARINA [7] is a hardware-based solution where the network device extracts the necessary (more complex than those used in other solutions) features and sends them to an ML server with a powerful prediction model. Flowrest [8] proposes a solution designed for hardware implementation, integrating a RF into the data plane.

III. The HALIDS System

We choose SwitchTree [4] as the basis for HALIDS' programmable switch implementation, while the communication mechanism with the oracle is inspired by CML-IDS [6]. SwitchTree consists of a P4 program responsible for extracting and calculating features for each packet and processing them with the previously trained RF. Traffic classification occurs at flow level, performing early detection, meaning it does not wait for the flow to complete before classifying it. For each packet arriving at the switch, a hash is applied using the standard quintuple IPs/ports/protocol as a key to identify each flow. For every incoming packet, both stateless (such as destination port) and stateful features (such as flow duration) are calculated, and a decision is made for each packet, based on the flow information obtained up to that moment. As the flow progresses, more information is gathered (more packets arrive), and if it is eventually detected as malware, the remaining packets of the flow are marked accordingly. Most of the stateful features are approximated since they involve floating-point operations, not supported by P4.

The classification is done by the RF, which is embedded in tables within the switch. In Figure 1, a Decision Tree embedded in a match-action pipeline is illustrated. Each level

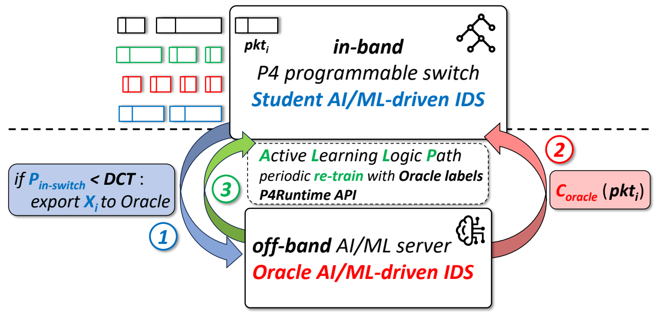
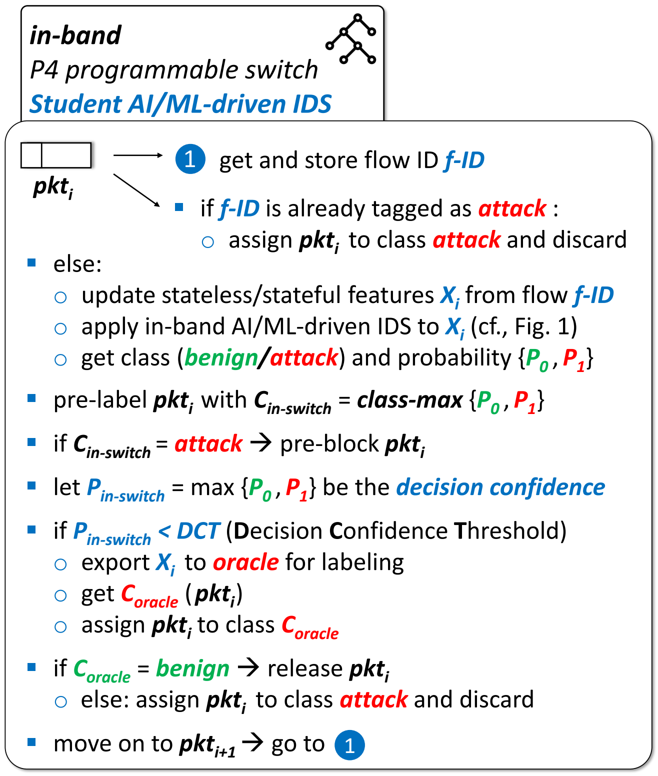


Fig. 2. HALIDS architecture for in-band/off-band traffic classification, an eventual re-training through Active Learning.

of the tree is mapped to a match-action stage, where a feature is checked at each level. Depending on whether the condition is satisfied or not, processing proceeds to the next level. This process continues until reaching a leaf, where a decision is made, and a class is assigned to the packet. The tables at each level will match the node and previously evaluated feature and have two types of action, one that checks the next feature (providing the necessary values) and one that sets the class.

Figure 2 depicts the HALIDS architecture. We enhance SwitchTree in several ways. Firstly, the entire switch training process is automated. Secondly, the concept of *oracle* is introduced. The oracle is an ML model - in this case also a RF - but trained with all the available data and with a higher model complexity as compared to the switch. The oracle establishes a connection with the switch via P4Runtime [9] and installs the P4 program. It also trains the switch for the first time using a reduced set of available data and a smaller RF. For this, it trains the model, generates the rules, and writes them into the switch's tables. Then, for each packet received by the oracle, it feeds the received features to the RF model and predicts the label, sending this value back to the switch. P4Runtime is utilized for communication between the switch and the oracle, encompassing packet exchange, configuration, and switch re-training, all in execution time. The analysis workflow logic of HALIDS is detailed in Figure 3.

At the switch, the confidence percentage of the decision made by the RF is incorporated. To achieve this, these values must be obtained during the training stage. Empirical probabilities are calculated as the number of samples of the label divided by the number of samples reaching a leaf. If the classification probability surpasses a pre-defined Decision Confidence Threshold (DCT), the normal forwarding is executed, taking into account the class predicted by the tree. If not, the packet is marked to be sent to the oracle, allowing for a more confident class assignment. When more than one tree are present, instead of a simple majority vote, these probabilities are weighted by the number of trees, and the classification is determined by the one with the highest value. To send the packet to the oracle, packet IO support from the P4Runtime shell [10] is employed. In this context, the goal is to transmit to the oracle the necessary features for class determination, along with some data required for feature approximation. Additionally, the oracle will store all



```
in-band
P4 programmable switch
Student AI/ML-driven IDS

pkt_i → (1) get and store flow ID f-ID
      → ▪ if f-ID is already tagged as attack :
            ○ assign pkt_i to class attack and discard
▪ else:
    ○ update stateless/stateful features X_i from flow f-ID
    ○ apply in-band AI/ML-driven IDS to X_i (cf., Fig. 1)
    ○ get class (benign/attack) and probability {P_0 , P_1}
▪ pre-label pkt_i with C_in-switch = class-max {P_0 , P_1}
▪ if C_in-switch = attack → pre-block pkt_i
▪ let P_in-switch = max {P_0 , P_1} be the decision confidence
▪ if P_in-switch < DCT (Decision Confidence Threshold)
    ○ export X_i to oracle for labeling
    ○ get C_oracle (pkt_i)
    ○ assign pkt_i to class C_oracle
▪ if C_oracle = benign → release pkt_i
    ○ else: assign pkt_i to class attack and discard
▪ move on to pkt_i+1 → go to (1)
```

Fig. 3. HALIDS traffic analysis workflow.

the classifications it generates (along with the corresponding feature values) and use this information for re-training the switch at runtime.

Note that our focus with HALIDS is to provide a closed loop, where a *student ML model* deployed at the switch can make quick decisions with a certain degree of confidence using a simple machine learning model in the data plane. In cases where the confidence level is not sufficiently high, the decision is offloaded to the oracle, which integrates a more powerful model. The oracle returns the decision made on the packet to the switch, which acts accordingly. Simultaneously, with the decisions made by the oracle, the switch is re-trained to keep it adaptive to changing traffic, therefore closing the loop. This periodic re-training logic is driven by an Active Learning Logic Path, which lets the student ML model to select those samples which require further analysis from the oracle, using the resulting labels for re-training after a certain number of queries has been met.

IV. Preliminary Results

We implement and deploy HALIDS using the SimpleSwitchGrpc version of BMv2 [11], employing virtual interfaces. Traffic traces are injected using TCP-replay [12]. Training and validation of the ML models is done using the UNSW-NB15 dataset[1], whereas testing is performed on top of a small packet trace from the same dataset (1000 packets). Recall that HALIDS works at the flow-level, but analyzing

[1]https://research.unsw.edu.au/projects/unsw-nb15-dataset

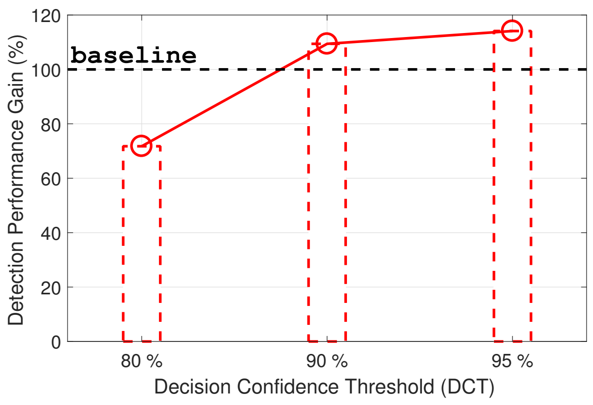


Fig. 4. Detection performance gain using HALIDS.

each incoming packet for early flow classification. In all the the evaluation examples provided next, the detection of the malware flows is realized with almost perfect accuracy, given the small size and characteristics of the trace used for testing. Therefore, we proceed with the classification evaluation at the packet level, where a better model performance is realized when the Recall for the packets of the malware flow is higher.

We assess the operation of HALIDS in two different setups: firstly, we verify that the functioning of the in-switch student model and the oracle model realize the same performance when integrating the same model, as a means to verify the correct implementation of HALIDS. We refer to this scenario as the baseline. The implementation is validated by processing all the testing packets through the switch, and alternatively by processing all these packets through the oracle. In both cases, student and oracle ML models' training is done with the all the training data, using the same DT architecture with a depth of 5-levels. As expected, and as a means for verification, obtained results are exactly the same at the switch and at the oracle.

Using the baseline detection performance as basis for the analysis, we then evaluate the operation of HALIDS as a functional detection system. In this case, we train the student, in-switch model with only 50% of the training dataset, maintaining a tree depth of 5, while the oracle is trained with all the training data, using a significantly more dense architecture, with 100 trees of depth 15. Using this setup, we test the detection performance for three different decision confidence thresholds DCT, taking DCT = 80%, 90%, and 95%. In a nutshell, the higher DCT, the more packets which are sent for classification to the oracle. Figure 4 depicts the obtained results, normalized to the baseline performance. Setting DCT to 80% results in all the packets classified in-band at the switch. As expected, given that the model is trained in this case with half of the training data as compared to the baseline, detection performance significantly degrades, dropping by almost 30%. When we take higher DCT thresholds, more packets are sent to the oracle, and detection performance improves with respect to the baseline, by about 10% for DCT = 90%, and by almost 20% for DCT = 95%.

Acknowledgment

This work has been partially supported by the FWF Austrian Science Fund, Project I-6653 *GRAPHS4SEC*, by the Austrian FFG ICT-of-the-Future project *DynAISEC* – Adaptive AI/ML for Dynamic Cybersecurity Systems – project ID 887504, as well as by the Uruguayan Agencia Nacional de Investigación e Inovación (ANII).

References

[1] R. Boutaba, M. Salahuddin, N. Limam, S. Ayoubi, N. Shahriar, F. Estrada-Solano, and M. Caicedo, "A Comprehensive Survey on Machine Learning for Networking: Evolution, Applications and Research Opportunities," *Journal of Internet Services and Applications*, 2018.

[2] F. Hauser, M. Häberle, D. Merling, S. Lindner, V. Gurevich, F. Zeiger, R. Frank, and M. Menth, "A Survey on Data Plane Programming with P4: Fundamentals, advances, and applied research," *ArXiv*, vol. abs/2101.10632, 2021.

[3] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: Programming Protocol-independent Packet Processors," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, 2014.

[4] J.-H. Lee and K. Singh, "Switchtree: In-network Computing and Traffic Analyses with Random Forests," *Neural Computing and Applications*, 11 2020.

[5] C. Busse-Grawitz, R. Meier, A. Dietmüller, T. Bühler, and L. Vanbever, "pForest: In-network Inference with Random Forests," *CoRR*, vol. abs/1909.05680, 2019. [Online]. Available: http://arxiv.org/abs/1909.05680

[6] P. Golchin, C. Zhou, P. Agnihotri, P. Agnihotri, M. Hajizadeh, R. Kundel, and R. Steinmetz, "CML-IDS: Enhancing Intrusion Detection in SDN through Collaborative Machine Learning," in *2023 19th International Conference on Network and Service Management (CNSM)*, 2023, pp. 1–9.

[7] M. Seufert, K. Dietz, N. Wehner, S. Geissler, J. Schüler, M. Wolz, A. Hotho, P. Casas, T. Hossfeld, and A. Feldmann, "MARINA: Realizing ML-driven Real-time Network Traffic Monitoring at Terabit Scale," *IEEE Transactions on Network and Service Management*, 2024.

[8] A. T.-J. Akem, M. Gucciardo, and M. Fiore, "Flowrest: Practical Flow-level Inference in Programmable Switches with Random Forests," *IEEE INFOCOM 2023 - IEEE Conference on Computer Communications*, pp. 1–10, 2023.

[9] T. P. A. W. Group, "P4Runtime Specification," https://p4lang.github.io/p4runtime/spec/main/P4Runtime-Spec.pdf, [Online; Accessed: April 2024].

[10] P. L. Consortium, "Packet IO," https://github.com/p4lang/p4runtime-shell/blob/main/usage/packet_io.md, [Online; Accessed: April 2024].

[11] ——, "Simpleswitchgrpc - a version of Simpleswitch with P4Runtime Support," https://github.com/p4lang/behavioral-model/blob/main/targets/simple_switch_grpc/README.md, [Online; Accessed: April 2024].

[12] A. Turner and F. Klassen, "tcpreplay - pcap Editing and Replaying Utilities," https://tcpreplay.appneta.com, [Online; Accessed: April 2024].