In-Network ML Feature Computation for Malicious Traffic Detection

João R. Amado[†], Francisco Pereira[†], Salvatore Signorello[‡], Miguel Correia[†], Fernando M. V. Ramos[†]

[†] *INESC-ID, Instituto Superior Técnico, Universidade de Lisboa,* [‡] *Telefonica Research*

ABSTRACT

We present Peregrine, a malicious traffic detector that offloads part of its computation to a programmable switch. The idea is to partition detection, by moving the ML feature computation module from a middlebox server to a switch data plane. The key innovation unlocked—computing the ML input features over *all* traffic—results in a significant improvement in detection performance: in our evaluation, up to 5.7x over the state of the art.

ACM Reference Format:
João R. Amado[†], Francisco Pereira[†], Salvatore Signorello[‡], Miguel Correia[†], Fernando M. V. Ramos[†]. 2023. In-Network ML Feature Computation for Malicious Traffic Detection. In *ACM SIGCOMM 2023 Conference (ACM SIGCOMM '23), September 10, 2023, New York, NY, USA.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3603269.3610866

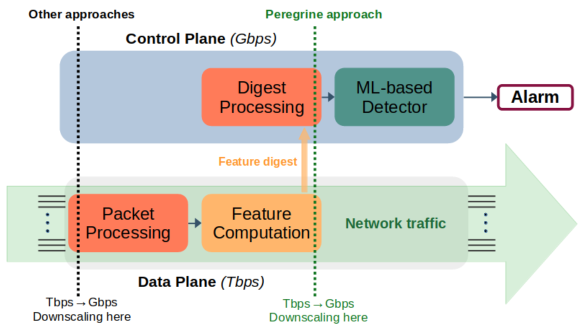

Figure 1: Peregrine approach vs. SOTA

1 INTRODUCTION

Network operators deploy Network Intrusion Detection Systems (NIDS) that capture and analyze packet flows to identify malicious traffic. The *ideal* NIDS should fulfil four requirements: **(R1)** observe and analyze *all* network traffic at high speed to **(R2)** detect *any* attack, **(R3)** *as it happens*, **(R4)** *without* generating false positives.

The most common NIDS detect and prevent attacks based on their (known) signatures [13, 14, 16]. Their limitation is that they are unable to detect zero-day attacks (**R2**) [5]. Another class of malicious traffic detectors focus on spotting deviations from regular traffic profiles, enabling detection of attacks for which there are no defined signatures (**R2**). The most promising solutions of this class leverage machine learning (ML) algorithms to learn traffic profiles and identify statistical variations [4, 6, 7, 11, 17], achieving impressive performance for zero-day detection (**R4**). However, these systems face a *performance challenge*, due to the processing overhead of the ML pipeline. As a result, most run offline [2, 3, 10, 12], precluding real-time detection (**R3**). Recent middlebox-based detection solutions [4, 11] perform detection online, but their limited packet processing capabilities does not match the requirements of today's Terabit networks. Their practical deployment requires heavy sampling (**R1**), seriously compromising detection performance.

In this work, we ask: *is it possible to perform malicious traffic detection in Tbps networks without compromising detection performance?* We present Peregrine, a malicious traffic detector that responds to this problem with a cross-layer approach that partitions the detector pipeline. In Peregrine, the feature computation module runs in a network switch, computing features *over all traffic* (**R1**), while the online (**R3**) ML-based detector (**R2**, **R4**) runs in a middlebox server.

The *key challenge* of our approach is dual: achieving generality while complying with the severe computational constraints of a network switch. On the one hand, our features need to be useful enough for a diverse set of ML-based detection systems. On the other hand, they need to be computable in a network switch. We achieve this by computing several dozens of commonly used features (close to one hundred)—for generality—and by resorting to stream-like, approximated computations—to fit the switch constraints. The *key takeaway* is that it is better to compute these approximated features over all traffic (**R1**) than to compute exact features over sampled traffic only (as existing systems). In other words, it is better to perform sampling *after* computing the ML features, even if only approximated (Figure 1).

2 SYSTEM DESIGN

Peregrine computes features for every packet traversing the switch. By moving feature computation to the data plane, we avoid the packet sampling required by SOTA detection systems in real network environments. Alongside per-packet computations, a feature digest is periodically "pushed" into an ML inference pipeline in the control plane. In our design process, we have abstracted the following main principles.

Cross-layer design. To scale detection to Tbps speeds we split the ML-based detection task, placing its components into either

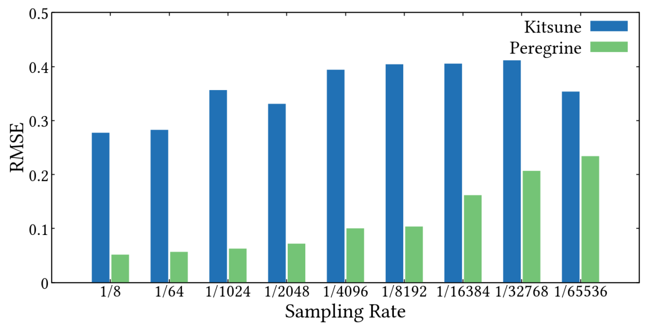

Figure 2: Peregrine vs Kitsune (SOTA) [11].

a middlebox server or a network switch. Specifically, by offloading ML feature computation processing to the data plane, we are targeting both high detection performance and deployability in high-throughput networks.

Per-packet feature computation on a switch. We instrument the data plane blocks to compute flow features as they process incoming packets. The stages of a PISA pipeline enable basic per-packet arithmetic operations and storing counters in stateful memory. By storing a few selected counters per-flow, we are able to compute a wide range of statistics and derive a sufficient number of features for ML inference, incrementally, as each packet traverses the pipeline.

Per-epoch ML inference in a middlebox server. We configure epoch values to define the sampling granularity at which the per-packet computed features are sent to the ML inference component. Our approach, a form of *enriched record sampling*, thus differs from traditional packet sampling—which effectively skips most packets—as the required sampling occurs only *after* feature computation (Figure 1).

For generality, Peregrine collects features common to many SOTA detectors, encompassing different flow keys: *[MAC src, IP src]*, *[IP src]*, *[IP src, IP dst]*, *[5-tuple]*. For each packet, we first increment three basic counters for each flow key: *number of packets*, *number of bytes*, *squared number of bytes*. Afterwards, we use these counters to compute statistics that broadly characterize the traffic patterns. Some statistics depend on a single flow direction (e.g., mean, standard deviation), while others encompass both inbound and outbound traffic (e.g., magnitude, radius, approximate covariance). We further observe the packet inter-arrival times for the monitored flows and apply decay factors (four in our implementation) to maintain statistics for different "time windows".

The data-plane functionality of Peregrine is not tied to any specific ML classification pipeline, by design. Rather, the features computed in the data plane are generic enough to be used as input to many different learning-based detection systems [1, 4, 11]. We implemented Peregrine's data plane components in an Intel Tofino switch [9]. The classification module used in the Peregrine prototype is Kitsune's KitNET neural network [11].

3 EVALUATION

The goal of this section is to show that Peregrine improves both system and detection performance by moving the feature computation component to the network data plane. We evaluate against Kitsune [11], a state-of-the-art NIDS, using datasets with labelled attack traces [11, 15].

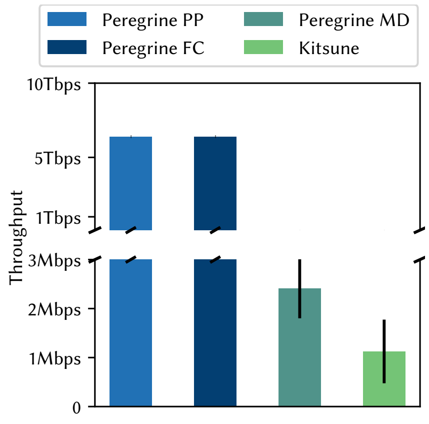

Figure 3: Runtime performance

Figure 2 presents the Root Mean Squared Error (RMSE) of the Area under the Curve (AUC) score—a global detection performance metric useful in addressing the "tension" between precision and recall—for each sampling rate over all attacks (lower results are better). The sampling rate has a different meaning in the two approaches (recall Figure 1). In Kitsune, it represents the rate at which packets entering the switch need to be sampled to fit within the processing limitations of the system, capable only of a few Gbps packet processing. In Peregrine, where all packets are processed in the data plane, sampling refers instead to the rate at which a digest is sent to the ML inference module, *after feature computation*. The obtained RMSE values are significantly smaller with Peregrine, decreasing up to 5.7x for the considered sampling rates. This demonstrates the value of sampling only after the feature computation module, thus computing statistics that consider all traffic.

Peregrine successfully compiles for the Tofino 2 T2NA [8] architecture, and the entire computation runs in a single pass of the pipeline. Therefore, both the Packet Processing (PP) and Feature Computation (FC) components run at 6.4Tbps line rate on the data plane (Figure 3). For the ML-based Detector (MD), our KitNET deployment was capable of processing at most around 2-3Mbps on average. Interestingly, the original Kitsune [11] achieved only half this detection throughput performance. This shows another benefit of offloading: with the FC component moved to the switch, we improve system performance (2×) and leave CPU cycles free for other tasks.

ACKNOWLEDGEMENTS

This work was supported by the European Union (ACES project, 101093126), by national FCT funds (Myriarch project, 2022.09325.PTDC), and INESC-ID (via UIDB/50021/2020). João R. Amado is supported by the FCT scholarship 2020.05965.BD.

REFERENCES

[1] Diogo Barradas, Nuno Santos, Luís Rodrigues, Salvatore Signorello, Fernando MV Ramos, and André Madeira. Flowlens: Enabling efficient flow classification for ml-based network security applications. In *NDSS*, 2021.
[2] Karel Bartos, Michal Sofka, and Vojtech Franc. Optimized invariant representation of network traffic for detecting unseen malware variants. In *25th USENIX Security Symposium*, 2016.
[3] Min Du, Zhi Chen, Chang Liu, Rajvardhan Oak, and Dawn Song. Lifelong anomaly detection through unlearning. In *Proceedings of the 2019 ACM SIGSAC*

Conference on Computer and Communications Security, 2019.

[4] Chuanpu Fu, Qi Li, Meng Shen, and Ke Xu. Realtime robust malicious traffic detection via frequency domain analysis. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 3431–3446, 2021.

[5] Ayyoob Hamza, Hassan Habibi Gharakheili, Theophilus A Benson, and Vijay Sivaraman. Detecting volumetric attacks on lot devices via sdn-based monitoring of mud activity. In *Proceedings of the 2019 ACM Symposium on SDN Research*, pages 36–48, 2019.

[6] Grant Ho, Asaf Cidon, Lior Gavish, Marco Schweighauser, Vern Paxson, Stefan Savage, Geoffrey M. Voelker, and David Wagner. Detecting and characterizing lateral phishing at scale. In *28th USENIX Security Symposium*, 2019.

[7] Austin Hounsel, Jordan Holland, Ben Kaiser, Kevin Borgolte, Nick Feamster, and Jonathan Mayer. Identifying disinformation websites using infrastructure features. In *10th USENIX Workshop on Free and Open Communications on the Internet*, 2020.

[8] Intel. P416 Intel® Tofino™ Native Architecture – Public Version. Retrieved 2023-02-15. URL: https://raw.githubusercontent.com/barefootnetworks/Open-Tofino/master/PUBLIC_Tofino-Native-Arch.pdf.

[9] Intel. The Intel® Tofino™ series of P4-programmable Ethernet switch ASICs. Retrieved 2022-10-20. URL: https://www.intel.com/content/www/us/en/products/details/network-io/programmable-ethernet-switch/tofino-series.html.

[10] L. Invernizzi, S. Miskovic, Rubén Torres, Christopher Krügel, Sabyasachi Saha, Giovanni Vigna, Sung-Ju Lee, and M. Mellia. Nazca: Detecting malware distribution in large-scale networks. In *NDSS 2014*, 2014.

[11] Yisroel Mirsky, Tomer Doitshman, Yuval Elovici, and Asaf Shabtai. Kitsune: an ensemble of autoencoders for online network intrusion detection. In *Network and Distributed Systems Security Symposium*, 2018.

[12] Terry Nelms, Roberto Perdisci, Manos Antonakakis, and Mustaque Ahamad. WebWitness: Investigating, categorizing, and mitigating malware download paths. In *24th USENIX Security Symposium*, 2015.

[13] Vern Paxson. Bro: a system for detecting network intruders in real-time. *Computer networks*, 31(23-24):2435–2463, 1999.

[14] Martin Roesch et al. Snort: Lightweight intrusion detection for networks. In *Proceedings of LISA'99: 13th Systems Administration Conference*, volume 99, pages 229–238, 1999.

[15] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A Ghorbani. Toward generating a new intrusion detection dataset and intrusion traffic characterization. *ICISSp*, 1:108–116, 2018.

[16] Zhipeng Zhao, Hugo Sadok, Nirav Atre, James C Hoe, Vyas Sekar, and Justine Sherry. Achieving 100gbps intrusion prevention on a single server. In *14th USENIX Symposium on Operating Systems Design and Implementation*, pages 1083–1100, 2020.

[17] Ziyun Zhu and Tudor Dumitraş. Featuresmith: Automatically engineering features for malware detection by mining the security literature. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016.