P4NIPS: P4 Network Intrusion Prevention System

Eduardo Vudala Senoski[1], João Ribeiro Andreotii[1], Raphael Kaviak Machnicki, Jorge Pires Correia[1], Vinicius Fulber-Garcia[1]

[1] Department of Informatics
Federal University of Paraná – Curitba, PR – Brazil

{evsenoski,jrandreotti,rkmachnicki,jpcorreia,vinicius}@inf.ufpr.br

Abstract. *An IPS must be deployed in line with the network traffic to be able to drop or accept packets according to evaluations done. Due to this, traditional IPS usage typically has an adverse effect on the network's latency and throughput. To mitigate the efficiency problem, we propose P4NIPS, an IPS built with P4 intended to run on a programmable switch. Its viability is demonstrated in a simulated environment, where the architecture of the Tofino 2 chip is used.*

1. Introduction

Computer networks transfer data, including sensitive data, across various domains and contexts. Therefore, the security of these networks is essential to a wide variety of applications. Among the alternatives to enhance network security, two systems stand out: the Intrusion Detection System (IDS) and the Intrusion Prevention System (IPS).

IDS and IPS differ in several key ways. The former can be used to monitor and analyze mirrored network traffic passively based on a set of heuristics, where it evaluates each network packet and flags it as benign or malicious. The latter is an active and inline system that can discard malicious packets flagged by the detection system. Packets steered through these systems are evaluated based on a set of fingerprints, also known as malware signatures and rules, that each or a set of packet may present. Signatures and rules are derived from previously known attacks.

A manner for implementing IDS and IPS is by utilizing Application-Specific Integrated Circuits (ASICs) integrated into contemporary switches, which can be programmed to perform high-speed packet processing. With this, it is possible to create Software-defined Networks (SDNs) that incorporate security mechanisms without severe performance penalties. Emerging programming languages, such as Programming Protocol-independent Packet Processors (P4), enable the direct control of ASICs to perform high-performance packet processing on switches, creating an ideal scenario for implementing Intrusion Detection Systems (IDSs) and Intrusion Prevention Systems (IPSs).

In particular, programmable network switches have been used to implement IDS solutions in state-of-the-art works. For example, [Chen et al. 2024] proposed an IDS with high-throughput capabilities, working on rates of 8 million packets per second (mpps). However, the current state-of-the-art works on programmable SDN do not approach the implementation of IPS solutions. It led us to our research question: Is it possible to make a programmable network switch work as an IPS? If so, what are the challenges to implementing it, and what could be its impact?

To answer the previous questions, we propose P4NIPS, a proof-of-concept P4-powered IPS. We simulated and tested the execution of the P4NIPS in the Intel Tofino

2 ASIC [1]. The results show that it can actively prevent specific malicious traffic from flowing through the switch. These results indicate the viability of deploying the proposed IPS in real-world scenarios, which opens up the opportunity to achieve high performance in terms of network traffic processing, particularly in terms of throughput and latency.

2. Background

The proposed solution presented in this work uses P4-supported network switches to create a network-based intrusion prevention system. IPSs are important to network security, but with the ever-growing network sizes, their management complexity levels grow along. Therefore, SDNs are used to centralize the network rules.

2.1. Intrusion Prevention Systems

[Abdulganiyu et al. 2023] categorizes IDSs concerning their positions on the network. In a Host-based Intrusion Detection System (HIDS), the IDS is executed on end-user computers, and all incoming and outgoing traffic is evaluated on each machine, leading to potential performance issues on those hosts. Another approach is to implement a Network-based Intrusion Detection System (NIDS), which relays the processing to a dedicated computer or a network switch. Similarly, IPSs can be classified into these two types. They can be referred to as Host-Based Intrusion Prevention Systems (HIPS) or Network-Based Intrusion Prevention Systems (NIPS).

In terms of infrastructure architecture, NIDSs are placed in parallel to the original network flow, requiring a copy of each packet to be redirected to them. NIPSs, however, are placed in line with the network flow, thereby potentially affecting the network throughput and latency directly.

There are widely used open-source solutions for both NIDS and NIPS. They can perform network traffic analysis using signatures or using behavior classification. Snort [Snort 2025], and Suricata [Suricata 2025] are primarily signature-based Intrusion Prevention Systems (IPSs), while Zeek [Zeek 2025] can perform both signature-based analysis and behavioral analysis. In particular, the Snort IDS/IPS provides a public set of rules used to pattern match against network packets. These rules can be written to look for malware signatures in a range of bytes at any specific byte offset within the analyzed packet's payload.

2.2. Software Defined Networks

Network architectures can be divided into two components: (i) the control plane, which is responsible for the logical topology of the network, being responsible for the definition of policies and routing protocols; (ii) the data plane, which executes the policies defined by the control plane, inspection all packets and forwarding them to the correct interface [Benzekki et al. 2016].

While control planes can be defined and configured using OpenFlow controllers such as Faucet [Faucet 2025], data planes can be implemented in domain-specific programming languages like P4. Primarily intended to manipulate packet headers, it was

[1] `https://www.intel.com/content/www/us/en/products/details/network-io/intelligent-fabric-processors/tofino-2.html`

first published in version $P4_{14}$ and then later updated to $P4_{16}$. The language syntax is based on the C programming language. Although the language was designed to be device-independent, it supports external libraries and functions that enable customization of switch functionalities.

3. Related Work

A widely used technique in modern SDN monitoring, particularly with the P4 language, is the utilization of in-network solutions. They can be applied in applications such as QoS monitoring, IDS, etc. FlowStalker [Castanheira et al. 2019] is a flow monitoring application that utilizes a robust cluster of P4-supported switches to perform both lightweight and heavyweight data processing.

The IDS solutions employ three different architectures, which differ in how the collected data is processed. First, using the switch to both collect and process data. This solution was applied in [Lewis et al. 2019] and [Chen et al. 2024]. P4ID [Lewis et al. 2019] processes each packet through a set of Snort rules, and P4-NIDS [Chen et al. 2024] processes each packet through a pre-trained Decision Tree algorithm. The second approach utilizes an external and dedicated computer to process the collected data, as seen in [Sahin et al. 2024] and [Pei et al. 2025]. Both of these solutions focus on machine learning algorithms to detect DDoS attacks. Finally, the third solution is a hybrid approach that utilizes both the switch and a dedicated computer to process data. The HALIDS application [Brandino et al. 2024] uses the Random Forest (RF) to classify malicious packets. The in-band packet analysis is performed, and based on a confidence threshold, it determines whether the packet should be sent to the off-band computer. Later, the RF switch parameters are updated based on the trained data of the dedicated computer.

We found two relevant IPS solutions using P4-supported switches. [Ahad et al. 2023] proposed an implementation of packet-dropping rules on malicious domain names based on a previously known list, blocking only DNS traffic. The solution proposed in [Lee et al. 2024] demonstrated an implementation utilizing machine learning for malicious traffic detection, which combines a detection algorithm executed on a general-purpose CPU with P4 code to pre-process the traffic. In contrast, the P4NIPS is an IPS that uses a programmable P4 switch and applies malware-specific Snort rules to drop malicious traffic. The rules are totally implemented in P4 code and aim to detect protocol-independent packets.

4. Design and Implementation

This section describes the design of the proposed IPS, called P4NIPS. Furthermore, it describes the P4NIPS implementation as a proof-of-concept tool, aiming to demonstrate the viability of detecting malware signatures using a programmable switch and then acting upon the detection. In this way, we first present the underlying technologies in the P4NIPS development; then, the signature detection model of the proposed IPS; and finally, we describe the architecture and implementation details of the P4NIPS tool.

4.1. Underlying Technologies

The P4NIPS tool is designed on top of Tofino 2 programmable switch ASIC, using the PISA architecture adapted to the Tofino Native Architecture. The relationship among

these technologies in the context of P4NIPS is presented in Figure 1, and key concepts about them are detailed next.

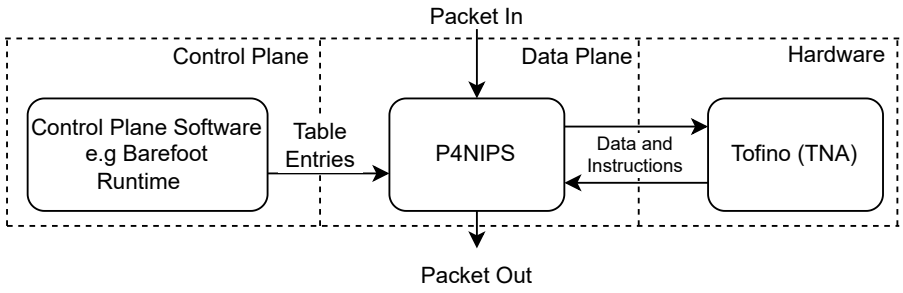

Figure 1. High-level representation of the interaction between the components involved in P4NIPS implementation

`Programmable Switch`. Programmable switch chips are on the way to replacing fixed-function switch chips. The study in [Zolfaghari et al. 2020] provides examples of architectures that implement programmable chip designs, where it is possible to highlight Tofino and Tofino 2. Taking these programmable switches (Tofino) into the spotlight, it is essential to note that the Tofino P4 software stack has recently been open-sourced [2], allowing developers to experiment with custom data plane designs without restrictions. With it, Intel P4 Studio - Intel's P4 Software Development Kit (SDK) for Tofino - is made available under a GitHub repository named Open P4 Studio [3].

Open P4 Studio implements the P4 compiler, a Tofino chip simulator, and Barefoot Runtime, which is the control plane software. These tools can be used to implement data plane algorithms and homologate their functionalities, which then could be ported to real switch ASICs. In this way, the P4NIPS tool is built for demonstration on top of the Open P4 Studio stack, where a Tofino 2 is used as a simulated switch, running compiled P4.

`PISA Architecture`. Programmable switch ASICs implement the Protocol-Independent Switching Architecture (PISA), which is composed of three essential components [Peterson et al. 2022]. The first is a Parser, which is programmed to define what header fields (and their location in the packet) are to be recognized and matched by later stages. The second is a sequence of Match-Action Units (MAUs), each of which is programmed to match (and potentially act upon) one or more of the identified header fields. The third is the Deparser, which re-serializes the packet metadata into the packet before it is transmitted on the output link. The deparser reconstructs the over-the-wire representation for each packet from all the in-memory header fields processed by earlier stages.

Figure 2 shows a high-level view of the pipeline of a PISA chip. Once it uses programmable switches ASICs, P4NIPS entirely considers all the characteristics, components, and operational model of the PISA architecture. However, the arrangement of PISA components can be organized in different ways by vendors, according to their chip architectures. Among these organizations, there exists a specific one adopted by Tofino.

`Tofino Native Architecture`. Tofino implements a pipeline design referred to

[2] `https://p4.org/intels-tofino-p4-software-is-now-open-source/`
[3] `https://github.com/p4lang/open-p4studio`

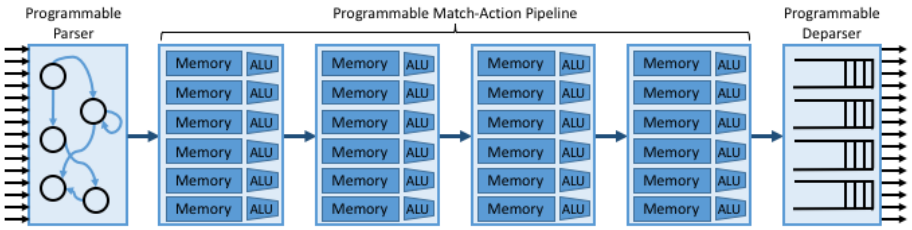

Figure 2. High-level overview of PISA's multi-stage pipeline [Peterson et al. 2022].

as Tofino Native Architecture (TNA), which utilizes the same building blocks as PISA, differing from it in two main aspects: (i) First, the pipeline includes a new fixed-function stage: the Traffic Manager. This stage is responsible for queuing, replicating, and scheduling packets. This stage can be configured in well-defined ways but cannot be reprogrammed in a general-purpose way; (ii) Second, the pipeline is divided into two stages by the Traffic Manager: ingress processing and egress processing. Each stage can be composed of up to 20 MAUs, which are distributed by the compiler based on the P4 code implementation.

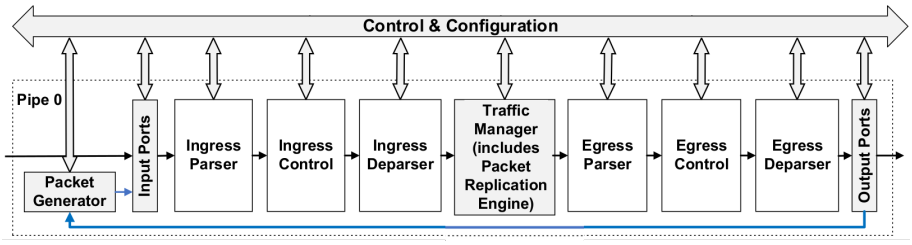

Figure 3. TNA single pipe, adapted from [Intel 2021].

Figure 3 depicts the TNA. It is worth noting that the pipeline includes additional mechanisms beyond PISA components. However, the relevant ones for P4NIPS are the ingress stage's parser, deparser, and control blocks.

4.2. Malware Signature Model

P4NIPS inherit the Snort rules model. Snort rules can be used to evaluate location-specific data in a packet, effectively evaluating if a packet has a malicious signature or not. Initially, the keywords of *offset*, *depth*, *distance*, and *within*, a part of a robust collection of keywords that indicate how Snort must execute matching heuristics on specific packet slices [4], are considered for the proposed IPS.

The *offset* keyword specifies the offset from the beginning of the packet from which the IPS must start searching for a pattern. *depth* is used in conjunction with the *offset* keyword to specify the number of bytes to be analyzed, starting from the specified offset. The *distance* keyword acts pretty much as *offset*, except the offset is not specified from the beginning of the payload but from the previous match. *within* keyword is similar to *depth*, indicating how many bytes should be analyzed starting from the position of the previous match.

[4] `https://docs.snort.org/rules/options/payload/oddw`

```
alert tcp $EXTERNAL_NET any -> $HOME_NET $HTTP_PORTS (
    msg:MALWARE-CNC Ios.Backdoor.SYNful inbound connection;
    content:text, depth 4, offset 78;
    content:\ |00 00 00| \, within 3, distance 1;
    content:\ |45 25 6D| \, within 3, distance 1;
    [...]
)
```

Listing 1. Snort rule to detect Synfull Knock

For example, we can consider a Synfull Knock attack. It enables malicious entities to compromise Cisco routers, rendering them vulnerable to interception, interruption, modification, and fabrication attacks [Hau et al. 2015]. To mitigate the attack, Cisco provided an IPS/IDS rule, a simplified version of which is presented in Listing 1.

This mitigation rule for Synfull Knock attack mainly indicates to the IDS/IPS to execute four checks on a packet: (I) check if the transport protocol is TCP, with any source port and some HTTP destination port; (II) check if it from an offset of 78 bytes from the beginning of the payload, the following four bytes are `text`; (III) if the previous check is positive, from a distance of 1 byte from that match, look for the bytes `00 00 00` in the next 3 bytes; and (IV) if the previous check is positive, from a distance of 1 byte from that match, look for the bytes `45 25 6D` in the next 3 bytes. If all the above conditions are confirmed, the security system will generate an alert or take some action, indicating that the evaluated packet contains a malicious signature.

4.3. P4NIPS Architecture and Implementation

P4NIPS utilizes TNA, along with the P4 programming language, to define how it handles malicious packets. Figure 4 illustrates the packet processing architecture through the system. Note that the metadata in Figure 4 indicates where information about the packet is stored without having to modify the packet itself. For example, the port to which the Traffic Manager must redirect the packet is represented in this structure.

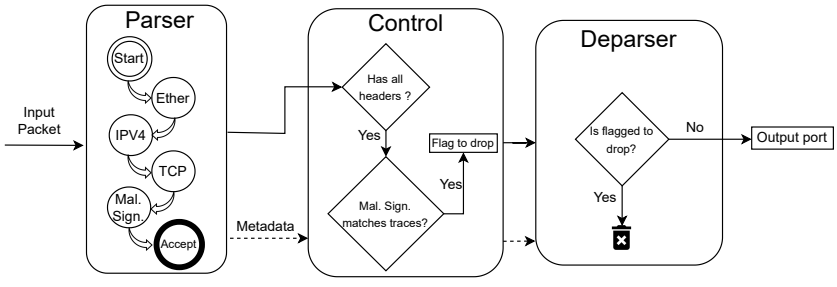

Figure 4. High-level view of P4NIPS packet flow.

$P4_{16}$ supports the *header* and *struct* keywords to define a data aggregate. The former is primarily used to specify network protocol headers, and the latter can be used to join related data. Each incoming packet is parsed into a set of P4 headers. In the case of P4NIPS, it is parsed considering the *struct* presented in Figure 5.

To successfully parse a packet, the switch ensures that the packet size is at least the sum of each header size. Then, if a packet is successfully parsed, the switch sets the header structure to a valid state. If not, an error is thrown, and the packet is dropped.

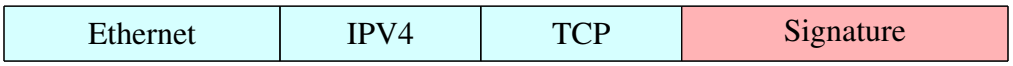

Figure 5. P4NIPS header structure (red fields are malicious traces in the structure, blue fields are benign data).

After parsing the headers, the switch can begin evaluating whether it is handling a malicious packet using P4 control blocks. First, P4NIPS needs to match the rule's list of source/destination IP/Port pairs, which is performed using a set of MAUs. This solution allows the valid IP addresses and ports to be configured via the control plane at runtime (an important detail since these fields can vary depending on the rule).

78 bytes	4 bytes	1 byte	3 bytes	1 byte	3 bytes
Unused	0x74657874	Unused	0x000000	Unused	0x45256D

Figure 6. Enlargement on header *Signature* from Figure 5: gray fields are unused data, red fields are malicious traces in the structure.

Then, to validate a malicious signature, the P4NIPS starts to verify the *signature* header structure. This structure is organized to match each signature pattern as shown in Figure 6. Evaluating the header for malicious patterns is performed via *IF* statements within the ingress control block. If all the conditions are met, the packet is considered malicious. It is then set to be dropped via a Tofino-specific flag in the metadata, which instructs the Deparser to either forward it or not.

5. Experimentation and Results

To create a generic and portable environment for the simulated tests, a Docker container is used. This container includes the compiled SDK and simulation model. Then, a set of Shell scripts and Python programs was developed to build P4NIPS, followed by the configuration and execution of the switch model. Finally, a Docker Compose file is used to integrate the P4NIPS tool and auxiliary scripts into the container. The experiments using the P4NIPS were based on real-world network traffic [Wang et al. 2025], consisting of a collection of different end-user use cases, *e.g.*, video streaming, general web browsing, and more. Due to this, the dataset does not aim to include any malicious signatures. The malicious network traffic is generated artificially and consists of a single packet being sent in a loop. For the tests, the malicious traffic emulated a Synfull Knock attack.

Each test is performed by executing the *tcpreplay* [5] utility, which sends both real-world network traffic and malicious traffic. The traffic is redirected to a pre-defined virtual network interface that is connected to the simulation model of a Tofino switch running the P4NIPS. Then, to monitor each experiment, a Python sniffer was used to compare the number of dropped packets and the total number of forwarded packets.

[5] `https://tcpreplay.appneta.com/`

The tests were executed by first sending the real-world traffic, followed by the malicious traffic. Assessing the effectiveness of P4NIPS is done by monitoring what is actually received on the output port of the switch, where the packet should have been redirected. Note that the objective is to validate the execution of P4NIPS, not its efficiency, as this requires a real Tofino switch.

Figure 7 represents the experiment. At timestamp (T) 5, the real web traffic (TXb) begins, and all packets are correctly routed by the switch, arriving at the destination port and increasing the RX count. At T20, the malicious traffic (TXa) begins to flow, but RX remains unaffected. From T30 and on, the real web data stops being transmitted, and RX from this point and on drops down to 0, even when the malicious packets are still flowing to the switch. This result indicates that P4NIPS is effectively filtering out malicious data. Thus, in this particular scenario, we could validate the security efficacy and operational behavior of P4NIPS.

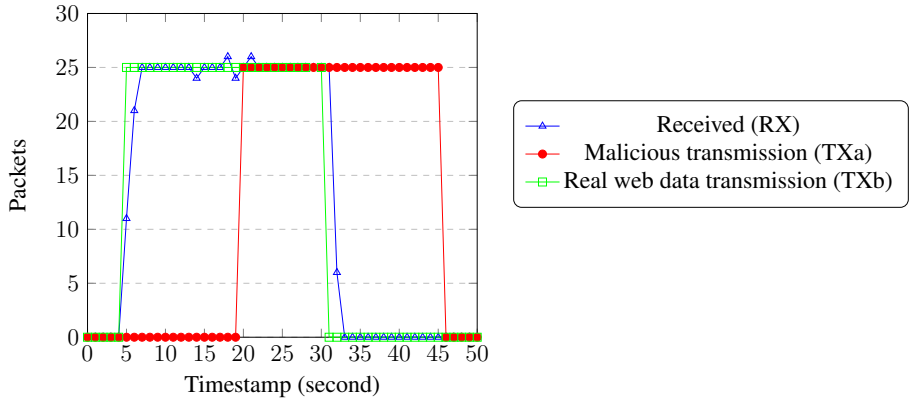

Figure 7. Transmitted packets compared to received packets.

All experimentation is performed on an x86_64 machine with Ubuntu 22.04 as the operating system, 16GB of RAM, and 8 CPU cores. A video of the tool in use can be found at `https://youtu.be/ezTnBPJ5JUw`.

6. Conclusion

This paper proposes a Proof-of-Concept Intrusion Prevention System in the data plane of a SDN. The solution was developed using the P4 programming language and tested using a Tofino 2 switch simulated model. The experiments expose the viability of implementing an IPS solution in a network programmable switch. Using malware-targeted signatures in the architecture can have a negligible impact on network throughput or latency, since Tofino ASICs process packets in at most 40 cycles, expanding the possible use cases of IPSs. The $P4_{16}$ programming language and the Tofino 2 switch impose a lot of limitations on the packet processing techniques besides the max-cycles limit. The unavailability of loops implies that it is not possible to perform complex string matching algorithms, e.g. Aho-Corasick [Aho and Corasick 1975]. To overcome these limitations, a small subset of rules has to be carefully selected before being implemented.

For future work, and to assess both latency and throughput, a real Tofino 2 switch will be used. Another future work is to develop a Snort rule to $P4_{16}$ code parser. This parser could be used to test a new rule using a fully automated workflow.

Aknowledgements

This work was supported by the Coordination for the Improvement of Higher Education Personnel (CAPES) - Program of Academic Excellence (PROEX) - Funding Code 001 and by C3SL - Centro de Computação Científica e Software Livre (`https://www.c3sl.ufpr.br`).

References

Abdulganiyu, O. H., Ait Tchakoucht, T., and Saheed, Y. K. (2023). A systematic literature review for network intrusion detection system (ids). *Int. J. Inf. Secur.*, 22(5):1125–1162.

Ahad, A., Bakar, R. A., Arslan, M., and Ali, M. H. (2023). Dpidns: a deep packet inspection based ips for security of p4 network data plane. In *2023 International Conference on Smart Computing and Application (ICSCA)*, pages 1–8. IEEE.

Aho, A. V. and Corasick, M. J. (1975). Efficient string matching: an aid to bibliographic search. *Commun. ACM*, 18(6):333–340.

Benzekki, K., El Fergougui, A., and Elbelrhiti Elalaoui, A. (2016). Software-defined networking (SDN): a survey. *Security and communication networks*, 9(18):5803–5833.

Brandino, B., Grampin, E., Dietz, K., Wehner, N., Seufert, M., Hoßfeld, T., and Casas, P. (2024). Halids: a hardware-assisted machine learning ids for in-network monitoring. In *2024 8th Network Traffic Measurement and Analysis Conference (TMA)*, pages 1–4.

Castanheira, L., Parizotto, R., and Schaeffer-Filho, A. E. (2019). Flowstalker: Comprehensive traffic flow monitoring on the data plane using p4. In *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, pages 1–6.

Chen, Y., Layeghy, S., Manocchio, L., and Portmann, M. (2024). P4-nids: High-performance network monitoring and intrusion detection in p4.

Faucet (2025). Faucet SDN Controller. `https://faucet.nz/`. Accessed: 2025-05-16.

Hau, B., Lee, T., and Homan, J. (2015). SYNful Knock - A Cisco router implant - Part I. `https://cloud.google.com/blog/topics/threat-intelligence/synful-knock-acis/`. Accessed: 2025-05-16.

Intel (2021). Open Tofino. `https://github.com/barefootnetworks/Open-Tofino/blob/master/PUBLIC_Tofino-Native-Arch.pdf`. Accessed: 2025-05-16.

Lee, A. Y.-P., Wang, M. I.-C., Hung, C.-H., and Wen, C. H.-P. (2024). Ps-ips: Deploying intrusion prevention system with machine learning on programmable switch. *Future Generation Computer Systems*, 152:333–342.

Lewis, B., Broadbent, M., and Race, N. (2019). P4id: P4 enhanced intrusion detection. In *2019 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, pages 1–4.

Pei, J., Hu, Y., Tian, L., Pei, X., and Wang, Z. (2025). Dynamic anomaly detection using in-band network telemetry and gcn for cloud–edge collaborative networks. *Computers & Security*, 154:104422.

Peterson, L., Cascone, C., O'Connor, B., Vachuska, T., and Davie, B. (2022). Software-defined networks: A systems approach. `https://sdn.systemsapproach.org/switch.html`. Accessed: 2025-05-16.

Sahin, H., Bezerra, J., Brito, I., Frez, R., Chergarova, V., Lopez, L. F., and Ibarra, J. (2024). Leveraging in-band network telemetry for automated ddos detection in production programmable networks: The amlight use case. In *SC24-W: Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 793–802. IEEE.

Snort (2025). Snort: The open source network intrusion detection system. `https://www.snort.org`. Accessed: 2025-05-13.

Suricata (2025). Suricata. `https://suricata.io/`. Accessed: 2025-06-01.

Wang, W., Jiang, W., Zhang, B., Zhu, Q., and Liao, C. (2025). A real network environment dataset for traffic analysis. *Scientific Data*, 12(1):1–12.

Zeek (2025). The Network Security Monitor. `https://zeek.org/`. Accessed: 2025-06-09.

Zolfaghari, H., Rossi, D., Cerroni, W., Okuhara, H., Raffaelli, C., and Nurmi, J. (2020). Flexible software-defined packet processing using low-area hardware. *IEEE Access*, 8:98929–98945.