P4-ONIDS: A P4-based NIDS optimized for constrained programmable data planes in SDN

Kairo Tavares[1], **Tiago Ferreto**[1]

[1]PPGCC - Graduate Program in Computer Science
PUCRS - Pontifical Catholic University of Rio Grande do Sul

kairo.tavares@acad.pucrs.br, tiago.ferreto@pucrs.br

Abstract. *Network Intrusion Detection Systems (NIDS) are one of the key defense mechanisms employed to detect and mitigate network-based threats. Several works explored the ability to offload NIDS pre-filtering capabilities to hardware platforms in order to reduce resource usage saturation and improve detection accuracy. Among them, network data plane solutions in SDN aim to leverage the hardware speed and the recent flexibility of programmable switches. However, those solutions are designed without considering a constrained data plane with limited table sizes and memory space, thus reducing accuracy detection and vulnerability buffer saturation attacks. This paper proposes P4-ONIDS, a solution that improves the parsing and compilation of NIDS rules for the data plane alongside sketch-based solutions for suspicious flow pre-filtering while maintaining a low usage of resources and leveraging the hardware speed of the data plane. We evaluate the compiler and our pre-filtering data plane capabilities in an emulated environment using Mininet with Snort NIDS. Results have shown more than 400x reduction on generated P4 rules. Some experiments reach an accuracy of approximately 90% with 40% of packets filtering.*

Resumo. *Sistemas de Detecção de Intrusão de Redes de Computadores (NIDS) são um dos principais mecanismos de defesa empregados para detectar e mitigar ameaças baseadas em redes de computadores. Vários trabalhos exploraram a capacidade de descarregar recursos de pré-filtragem de NIDS para plataformas de hardware para reduzir a saturação de usos de recursos e melhorar a precisão da detecção. Entre eles, as soluções de plano de dados de redes em SDN visam usufruir da velocidade do hardware e a flexibilidade recente dos switches programáveis. No entanto, essas soluções são projetadas sem considerar um plano de dados restrito com tamanhos de tabela e espaço de memória limitados, reduzindo assim a detecção de precisão e ataques de saturação do buffer de vulnerabilidade. Este artigo propõe P4-ONIDS, uma solução que melhora a análise e compilação de regras NIDS para o plano de dados ao lado de soluções baseadas em sketches para pré-filtragem de fluxo suspeito, mantendo um baixo uso de recursos e aproveitando a velocidade de hardware do plano de dados. Avaliamos o compilador e nossa capacidade de pré-filtragem no plano de dados em um ambiente emulado usando Mininet com Snort NIDS. Os resultados mostraram uma redução de mais de 400x nas regras P4 geradas. Alguns experimentos alcançam uma precisão de aproximadamente 90% com 40% de filtragem de pacotes.*

1. Introduction

Network Intrusion Detection Systems (NIDS) are key technologies to secure communication infrastructures. The ability to detect malicious traffic in the network is usually achieved using signature and anomaly-based techniques. However, as network bandwidth increased, those systems were challenged due to their single high-throughput choke points, leading to dropped packets when their processing limits are reached. The resulting packet loss due to a saturated IDS/IPS provides increased potential for false negatives [Khalil 2015]. Moreover, those detection mechanisms add significant overhead to the network for deployments that require the traffic to be inspected inline in the ingress or egress of the network. This issue is due to the high computational complexity of detection techniques and the time to process packets in software. Especially in modern high-speed networks, it poses performance challenges for practical deployments. Therefore, to achieve the highest quality of detection, NIDS should process as much relevant data as possible without becoming the bottleneck of a network connection. Moreover, NIDS implementation should be flexible enough to accommodate detection methods of ever emerging new security threats.

According to established taxonomies [Hoque et al. 2014], intrusion detection systems can be categorized according to the type of detection method: Anomaly-based NIDS uses behavior-based techniques by defining a model of normal network behavior and then detecting deviations to this model. On the other hand, knowledge-based systems use a precise definition of the attack and match incoming traffic against this definition. The most widespread variants of knowledge-based systems are signature or rule-based. In signature-based NIDS, a detection engine applies a rule-set to all received data. The majority of state-of-the-art rules contain patterns that are matched against the payload of the received packets. These patterns range from selected bytes to complex Regular Expressions (Regex) matching individual packets and the payload in a flow of packets. The performance of a signature-based NIDS mainly depends on the number of rules [Erlacher and Dressler 2018]. Thus, in practical applications, the rule-set needs to be adapted to the domain-specific use case while considering how the number of rules impacts the risk of not detecting possible intrusions.

A common solution to achieve low overhead is to offload the detection capabilities to dedicated hardware specialized functions, such as multi-core processing, GPUs, and FPGAs. The performance gap between the execution speed of security software and the amount of data to be processed is ever-widening. However, continuously expanding signature databases has become a major impediment to achieving scalable hardware-based pattern matching. Additionally, evolutionary rule databases have necessitated real-time online updating for reconfigurable hardware implementations. In contrast, instead of entirely focusing on pattern comparison for performance improvement, researchers have also proposed SDN-based solutions [SDxCentral 2020] to improve performance by taking advantage of some properties in network traffic [Nam and Kim 2018] [Xing et al. 2013]. They indicated that malicious packets make up only a small share of total traffic. Consequently, they adopted hybrid architectures in which hardware devices handle pre-filtering and PC-based software implements Snort for final identification. A new breed of switches referred to as programmable suggests an evolution path to allow programmers to define how packets are processed all the way down to the wire. The advent of programmable hardware has also brought new programming languages to the fore, such

as P4 [Bosshart et al. 2014] and Click [Kohler et al. 2000]. Thus, the flexibility provided by the programmable data plane switches opens the possibilities to create or improve solutions at the data plane level.

Several works [Teofili et al. 2011, Lopez and Duarte 2015, Xing et al. 2013, Nam and Kim 2018] have been proposed to leverage the data plane programmability to offload NIDS capabilities. Among them, P4ID [Lewis et al. 2019] proposes a solution composed of a NIDS rule parser and a P4-based packet processing data plane to reduce traffic processed by a NIDS. However, these solutions present several limitations. First, pre-filtering accuracy can be reduced when a high number of P4 rules generated by the P4ID parser cannot be deployed in a size-constrained rules table in the data plane. Second, the P4ID data plane is vulnerable to buffer saturation attacks due to the hash table structures needed for tracking ongoing suspicious flows. Third, the pre-filtering traffic reduction capability is limited to the pre-defined set of TCP/UDP well-known ports.

This paper proposes P4-ONIDS (P4 - Optimized Network Intrusion Detection System), a P4-based pre-filtering NIDS solution optimized for constrained programmable data planes in SDN. Our proposal reduces the number of generated P4 rules by introducing a NIDS P4 rule compiler that applies additional aggregation steps while leveraging port range matching capabilities. Moreover, a sketch-based solution is used to pre-filter suspicious flows to leverage the hardware speed while maintaining a low usage of resources in the data plane. We evaluate the P4-ONIDS compiler and data plane implementations using public available NIDS rulesets and research network PCAP dataset from CICS2017 [Panigrahi and Borah 2018] with malicious attacks on an emulated environment using Mininet.

In summary, this paper presents the following contributions: (i) a NIDS rule compiler to reduce NIDS P4 rules carnality, (ii) implementation of a sketch-based solution for pre-filtering suspicious flows to a NIDS while enabling flow expiration with timing-based deletion without control plane interaction, and (iii) a comprehensive evaluation of the compilation of the P4-ONIDS rules and snort accuracy over different pre-filtering settings against intrusion detection evaluation datasets in an emulated environment. Moreover, the experiments have shown a significant reduction of generated P4 rules and detection accuracy of approximately 90% while pre-filtering suspicious flows to a NIDS.

The rest of this document is organized as follows. Section 2 presents related work relevant to our solution. The P4-ONIDS compiler and data plane solutions are presented in Section 3. Section 4 presents experimental results. Finally, Section 5 concludes the paper.

2. Related Work

Different solutions that offload NIDS capabilities to hardware have worked towards interpreting NIDS rules to match the hardware capabilities for better performance. Depending on the target platform, the compilation of rules can lead to optimization in space, parallel processing, or better accuracy. In this section, we classify those solutions into three groups: (i) Multi-core and GPU, (ii) NetFPGA, and (iii) SDN-based networks

2.1. Multi-core and GPU

The authors in [Wan et al. 2012], and [Chen et al. 2009] propose the compilation of rules to a multi-core processor platform to benefit from the processing power and speed up NIDS detection of network packets. Those works focus mainly on improving multi-pattern matching and load balancing in parallelization for a multi-core platform. However, they are limited to generic software-based platforms and not specialized network-based hardware solutions.

Other authors [Lin et al. 2012] propose a novel parallel algorithm to accelerate the exact string matching process of Snort NIDS on GPU. This method achieved an improvement in processing and can utilize the parallel processing capabilities of GPUs. However, this method requires an extra copy of the packets to be sent to the GPU. Also, it transfers the matching result back to the CPU to make an action.

2.2. NetFPGA

Hardware-based NIDS are faster than software-based but have several disadvantages including limited flexibility and relatively high cost. NetFPGA is a low-cost open-source hardware platform, primarily designed as a tool for teaching networking hardware and router design. NetFPGA has been widely used in networking and security applications. Some examples include: real-time URL extractor, hijack incoming packet's header and re-compute the checksum, online classifier of network traffic, providing accurate timestamp system for network measurements, and precise traffic generator.

Motivated by the practical impossibility to pack a large amount of legacy Snort rules over a resource-constrained hardware device, the authors in [Teofili et al. 2011] focus on adapting and simplifying Snort rules, to support a commercial, low-end, NetFPGA board, meanwhile providing good filtering performance. They chose about one thousand Snort rules randomly drawn from the complete ruleset and performed experiments applying these rules to a training dataset composed of a relatively large traffic trace collected from a regional ISP backbone link. The goal was to determine how these rules can be simplified meanwhile retaining a comparable detection performance to the original, non-adapted, rules. Finally, they validated the performance of the adapted rules against additional collected traffic traces. Their results show that about one thousand adapted Snort rules can be supported over a low-end FPGA-based Snort pre-filter, with 93% data reduction efficiency.

2.3. SDN

As the new SDN paradigm emerged to separate the control plane from the data plane, it opens up new research opportunities. SDN enables more flexible and predictable network control and makes it easier to extend the network with new functionality through the controller's programmability.

Surica Openflow [Nam and Kim 2018] and SnortFlow [Xing et al. 2013] proposes an elastic and distributed IDPS (Intrusion Detection and Prevention System) for defense against DoS attacks in virtualized SDN. Those inherit the intrusion detection capability from Snort and Bro and flexible network reconfiguration from OpenFlow. SnortFlow has created three modules for NIDS rules: SnortFlow daemon, alert interpreter, and rules generator. The parsed and filtered information is passed to the rules

generator that generates the rules to be injected into the OpenFlow device to reconfigure the network.

The flexibility of IDS rules specification and the resource constraints of hardware platforms for offloading IDS capabilities becomes a problem for high-speed networks. For the IDS use cases, even OpenFlow in version 1.5 [Open Network Foundation. (2013)] is limited in three aspects: action capabilities, matching capabilities, and table capacity. First, the only action required in the specification is output, group, and drop. Therefore, one can only drop or mirror traffic as action. Second, the matching capabilities on the source and destination ports are limited to exact matching or masked match, thus not suited for range port queries. The limitation on matching capabilities impacts directly the number of rules that need to be compiled for an OpenFlow table, with a straightforward port range definition that can lead to a high carnality number of OpenFlow rules. For instance, a destination port defined as 1024 or higher (specified as [1024:] in Snort rules) can lead to more than ten rules if the port mask is used correctly and without considering the permutation required if other fields like source port also have multiple values.

Programmability in SDNs is confined to the network control plane. The forwarding plane is still primarily dictated by fixed-function switching chips. To solve this problem, the advent of programmable data planes allows programmers to define how packets are processed all the way down to the wire. This is possible by a new generation of high-performance forwarding chips. At the high-end, PISA (Protocol Independent Switch Architecture) chips promise multi-Tb/s of packet processing. Moreover, this novel approach has also brought new programming languages to the fore. P4 [Bosshart et al. 2014] is a declarative language for expressing how packets are processed in the data plane using a forwarding model consisting of packet parser and MAT stages.

P4ID [Lewis et al. 2019] is a solution composed of a NIDS rule parser and a P4-based packet processing data plane to reduce traffic processed by a NIDS. It combines rulesets designed for traditional Intrusion Detection Systems such as Snort and applies pre-filtering in the data plane. Therefore, this technique allows handling packets in the network itself without the direct involvement of the NIDS. However, P4ID presents some limitations. First, the P4ID parser final step generates P4 rules based on a ternary match for port range, increasing the rules cardinality while limiting further optimization. Second, the P4ID data plane stateful filtering stage is based on a 5-tuple flow identifier and register to keep track of ongoing suspicious flows that can lead to buffer saturation attacks. And third, the pre-filtering traffic reduction capability is limited to the pre-defined set of TCP/UDP well-known ports.

3. Proposed Solution

As previously discussed, offloading NIDS pre-filtering capabilities to the network data plane can help NIDS avoid saturation and improve accuracy due to the reduction of the traffic needed to be inspected. Offloading NIDS rules to the data plane require two main components: i) the parsing and compilation of NIDS rules to a target data plane system, and ii) a data plane implementation capable of filtering suspicious traffic. However, the target data plane is a constrained resource system. Its capability to filter suspicious traffic is directly related to the number of NIDS rules it can host.

This paper proposes P4-ONIDS (P4 - Optimized Network Intrusion Detection System), which improves the parsing and compilation of rules while leveraging sketch-based solutions for pre-filtering to address those limitations identified in P4ID while leveraging the hardware speed and maintaining a low usage of resources in the data plane. P4-ONIDS is based on the P4 language, used in programmable data planes. Figure 1 illustrates the two main solution components: the P4 NIDS compiler and the P4 Switch data plane. The P4 NIDS Compiler aims to generate the best P4 Rules given a specific compiler goal for the target data plane from the available NIDS rules and a NIDS configuration (e.g. internal and external network subnet, rule priority, etc.). In contrast, the P4 Switch data plane is responsible for providing switching capabilities to network traffic, i.e., detecting, filtering, and redirecting suspicious traffic to the NIDS based on the P4 rules generated by the compiler.

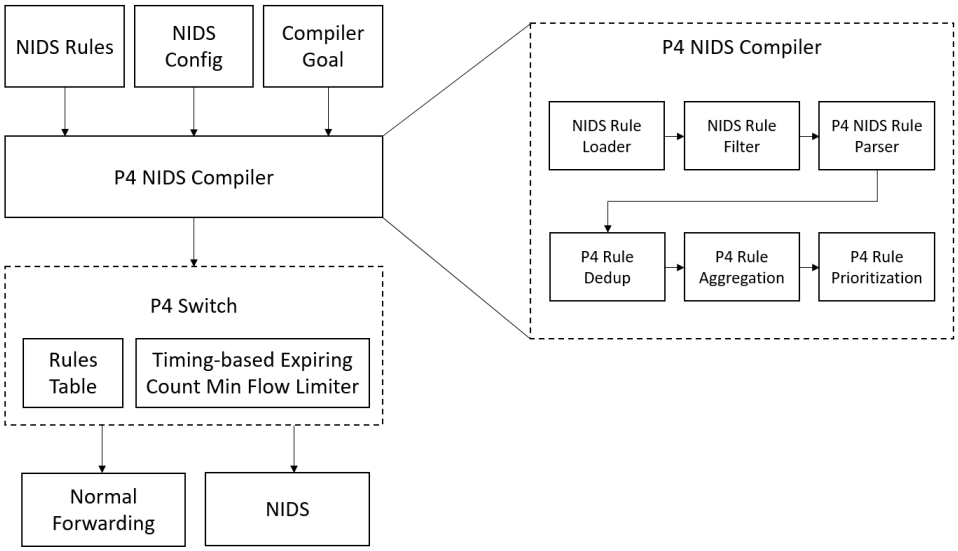

Figure 1. P4-ONIDS Architecture

The rules used in this work are based on the widely used rule syntax of open sources NIDS, such as Snort and Suricata. The rules themselves consist of a rule header and a rule body [El-Bakry and Mastorakis 2010]. For example, the following rule generates an alert when any TCP request is made from the subnet 10.0.0.0/8 to the destination in the configured home network at port 80 to 90 and 8080.

alert tcp 10.0.0.0/8 any → $HOME_NET [80:90, 8080]
(msg:"Found attack"; content:"attack"; sid:1; rev:3;)

The number of rules alongside its configuration can lead to a generation of a large number of P4 rules. Thus, it is essential to reduce the number rules by compiling them into the smallest ruleset possible while being aware of the constraints of the target data plane (e.g., table size, available matching capabilities, and controller channel bandwidth).

The P4 NIDS compiler is composed by several modules. First, the Rule Loader interprets and parses the NIDS rules based on the NIDS Configuration. Next, the Rule Filter module filter out the supported rules for the target platform as specified in the compiler goal. The P4 NIDS Rule Parser module translates the filtered rules into P4-based rules. Then, the Rule Dedup and Aggregation modules reduces and optimizes the

number of P4 generated rules. Finally, the Rule Prioritization module selects the subset of rules that will be pushed to the P4 data plane based on a goal (e.g. rules with high severity first) given the specified target constraint (e.g. maximum table size).

The generated P4 rule comprises a 6-tuple match with the following fields: the protocol type, source/destination addresses, source/destination port ranges, and TCP flags if available. In general, the ternary field match type is applied to those packet fields. However, to achieve the smallest subset of generated rules, P4-ONIDS relies on the range field match type to better model the source and destination port range. Figure 2 illustrates a generated P4 rule:

```
redirect_nids 0x6&&&0xFF 0x00000000&&&0x00000000 0->65535 0xAC100000&&&0xFFFF0000 80->90 0xB&&&0xFF => 2
    (action) (protocol src_network src_port_range dst_network dst_port_range tcp_flags) => (priority)
```

Figure 2. Example of P4 generated rule

The match of port range type already reduces the number of generated rules compared to the usage of ternary matching for those fields. Moreover, it allows the compiler to aggregate rules further by merging the rules with conflicting port ranges. Therefore, more generic generated rules that have the same effect of a more specific rule are merged within the more generic, leading to a reduction in redundant rules beyond simple deduplication.

Listing 1. Pseudo-code for aggregation of deduplicated rules

```
1  def is_rule_within(r1_match, r2_match):
2      if (r1_match.proto == r2_match.proto and r1_match.proto_mask == r2_match.proto_mask) and \
3          (r1_match.src_port_start >= r2_match.src_port_start and r1_match.src_port_end <= r2_match.src_port_end) and \
4          (r1_match.dst_port_start >= r2_match.dst_port_start and r1_match.dst_port_end <= r2_match.dst_port_end) and \
5          (r1_match.src_network == r2_match.src_network or r1_match.src_network in r2_match.src_network) and \
6          (r1_match.dst_network == r2_match.dst_network or r1_match.dst_network in r2_match.dst_network) and \
7          (r1_match.flags == r2_match.flags and r1_match.flags_mask == r2_match.flags_mask):
8            return True
9      return False
10
11 def aggregate_deduped_rules(rules):
12     final_rules_map = generate_rules_map_by_match(rules)
13     aux_rules_list = generate_aux_list_copy(rules)
14     while len(aux_rules_list) > 0:
15         current_rule = aux_rules_list.pop()
16         for aux_rule in aux_rules_list:
17             if current_rule in final_rules_map and aux_rule in final_rules_map:
18                 if is_rule_within(current_rule.match, aux_rule.match):
19                     merge_specific_rule_into_generic(specific=current_rule, generic=aux_rule, map=final_rules_map)
20                 elif is_rule_within(aux_rule.match, current_rule.match):
21                     merge_specific_rule_into_generic(specific=aux_rule, generic=current_rule, map=final_rules_map)
22     return final_rules_map.values()
```

Listing 1 shows the pseudo-code used for aggregating deduplicated rules. Function *aggregate_deduped_rules* at line 11 receives a list of already deduplicated/unique rules. It checks for all rules if either the current rule or auxiliary one is within each other. Suppose a rule is detected as part of a more generic one. In that case, the function will merge the more specific rule into the more generic rule by updating the generic rule metadata and deleting the specific from the final rules map. After all rules are compared, the final generated rules are the values that remained on the rules map. To evaluate if a rule is within another, function *is_rule_within* at line 1 will check if the 6-tuple fields are equal or within the values range. In this work, we limit the compiler to check for

equivalence on both proto and TCP flags. Additionally, since the aggregation process keeps the priorities and identification of the original rules in the generic rule metadata, the compiler can prioritize the final P4 generated rules based on specific goals for the target constrained data plane. For instance, the N more critical aggregated rules, the rules representing more aggregated rules, or both.

A P4 rule will help the data plane detect the suspicious traffic sent to the NIDS for better inspection. However, the NIDS is interested in the suspicious flow and not only on the packet that matched the table entries. Therefore, the data plane needs to keep track of the ongoing suspicious flow during its lifetime. Moreover, there are scenarios, like in DDoS network saturation attack, where the small subset of suspicious traffic is responsible for the large portion of the network traffic. Those scenarios require the data plane to have additional safeguards to avoid the saturation of the NIDS.

P4-ONIDS tackles this problem by sending the first N packets of any suspicious flow, defined by the 5-tuple fields, to the NIDS. This threshold N can be dynamically configured by the control plane to protect the NIDS from saturation, with the downside of reducing the classification accuracy. Our solution makes use of a Count-Min (CM) sketch [Cormode and Muthukrishnan 2005] to keep track of the suspicious flow packets that are redirected to the NIDS. Similar to counting bloom filters [Song et al. 2005], a CM sketch is a probabilistic data structure that serves as a frequency table of events in a stream of data. It uses hash functions to map events to frequencies, but unlike a hash table, it uses only sub-linear space at the expense of overcounting some events due to collisions. Thus, a CM sketch may overestimate but never underestimate the true count in a point query.

The ability to delete idle flows from the data structure is vital to allow new flows to be filtered. However, a typical Count-Min sketch does not have the functions to delete items from an event stream. Thus, without this ability, a CM sketch can saturate and lose to count the frequency of the items. To solve this issue, we leverage the timing-based deletion mechanism described in [Bonomi et al. 2006] to ensure that an uncompleted deletion eventually happens. The exact mechanism can be applied to a Count-Min sketch. This mechanism resets the values of idle counters of the Count-Min data structure when a phase transition occurs. In our solution, this phase transition happens when new packets arrive in the data plane and a given time duration threshold D is exceeded. To clean the idle flow from the data structure, a predefined set of operations corresponding to the size of the counting bloom filter that composes the count-min sketch of depth d times the width w, needs to be performed. Therefore, this timing-based expiring sketch solution does not require interaction from the control plane and leverages the hardware speed while maintaining a low usage of resources in the data plane. In summary, the proposed solution can be described as follows:

1. A P4-ONIDS compiler that improves the reduction of P4 rules generated from a ruleset while prioritizing the rules for a target data plane
2. Utilization of a space-efficient data structure, a count-min sketch, to track and limit the redirect flow to the NIDS, therefore better controlling and safeguarding usage of the NIDS.
3. Utilization of a timing-based deletion mechanism to expire idle flows from the count-min sketch that don't require control plane interaction.

4. Evaluation

In this section, we conduct a comprehensive set of simulation experiments and analyze the results to evaluate the performance of the P4-ONIDS two main components: compiler and data plane. We show that the compiler has a smaller generated P4 ruleset than the P4ID parser and how the data plane filtering with different packet filtering limits impacts NIDS detection.

4.1. Experimental Setup

We implemented the P4-ONIDS compiler in Python and the data plane using the P4 programming language targeting the second version of the P4 software switch, also known as behavioral model or BMV2. P4App [P4App 2020] provides a framework that allows developers to build, run, test, and debug P4 BMV2 switch implementations on Mininet [De Oliveira et al. 2014] testbeds. We ran our experiments on a Virtual Machine running Ubuntu Server 18.04.3 with the following configuration: 16 Cores, 32GB of RAM, and 250 GB of disk space.

4.1.1. Compiler Experiments

As mentioned in Section 2, P4ID provides a parser and data plane to offload NIDS rules. We compare our compiler solution with the P4ID parser approach, where the rule reduction strategy is deduplication and uses ternary matches. Two public available NIDS rulesets are used: the Snort Community [1] and Emerging Threats [2]. The rulesets are loaded with the same configuration used in the data plane experiments and filtered with the same criteria. The current implementation only supports rules that are IPv4, unidirectional, and don't have negation statements. Additionally, rules with a port range greater than 512 are filtered out to avoid too permissive rules, the same as in P4ID experiments. Finally, we compare the output of those rulesets to validate the reduction of generated P4 rules.

4.1.2. Data Plane Experiments

Based on the data plane experiments setup from P4ID, we reproduce the same environment and test methodology to have more consistent results. As illustrated in Figure 3, the Mininet setup automated by P4App, consists of one P4 BMV2 Switch connected to three hosts: PCAP Sender, NIDS, and Normal Forwarding. The sender uses *tcpreplay* to send the five PCAPs from the CICIDS2017 dataset [Panigrahi and Borah 2018] that contains benign and the most up-to-date common attacks at a rate of 2000 pps (packets per second), which resembles the actual real-world data (PCAPs). The NIDS is a host that runs a Snort 2.9 process to analyze the filtered data sent by the data plane. Finally, the Normal Forward host only records the data that was not sent filtered.

The Count-Min sketch filter was set to a depth d of 4 hash functions with the width w of 256 positions where each position corresponds to a register of 16 bits. The 4 different hash functions chosen between the supported in the P4_16 data plane are: *crc16*, *csum16*,

[1] https://www.snort.org/downloads

[2] https://rules.emergingthreats.net/open/snort-2.9.0/

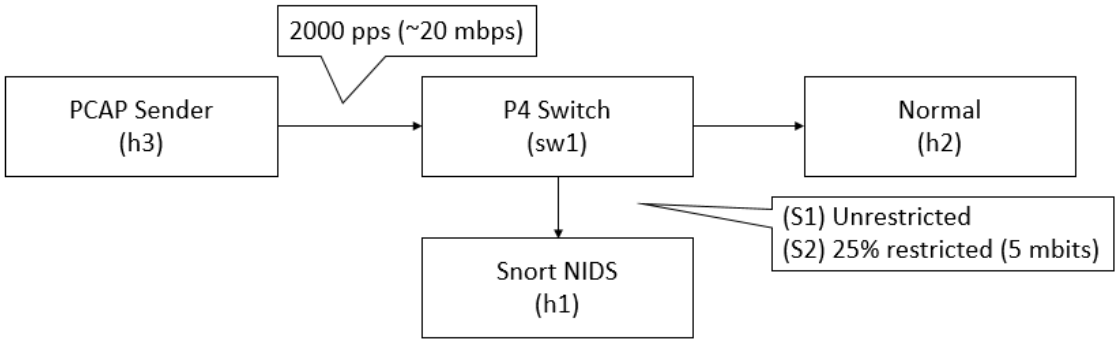

Figure 3. Data Plane Experiment Topology

crc32, and *csum32*. The timing-based deletion mechanism was composed with the same depth and width, but with only a 1-bit register. The time duration D of expiration was set to 10 seconds during all experiments. The size of the timing-based sketch solution is approximately 3 KB.

The same Snort community ruleset and the NIDS configuration used in the compiler experiments have been used to generate P4 rules and for the Snort NIDS. For results fidelity, this ruleset was filtered based on the final rules generated by the P4-ONIDS compiler. Table 1 describes the PCAPS dataset used in the experiments alongside the expected number of snort alerts when only the PCAP is analyzed without any simulation. Notice that Monday PCAP is labeled as been composed only of benign traffic. However, it is still expected some alerts to be generated depending on the ruleset used. In this experiment, Monday PCAP has a low number of alerts in comparison to the others PCAPs. To better understand the impacts of the filtered traffic on the Snort alerts, we run the experiments with two scenarios: (Scenario 1) unrestricted NIDS link bandwidth, and (Scenario 2) NIDS link bandwidth restricted to 25% percent (5 Mbps) of the expected peak bandwidth (20 Mbps). Finally, we evaluate the percentage of traffic redirected to the NIDS and the percentage of alerts generated during the experiments.

Table 1. Data Plane PCAP Summary

PCAP	Attack Description	# of Packets	Expected # of Snort alerts
Monday	Benign (Normal Human Activity)	11709971	24
Tuesday	SSH and FTP Brute Force	11551954	11923
Wednesday	DoS, DDoS, and Heartbleed	13788878	175378
Thursday	Web Attacks and Infiltration	9322025	589
Friday	Botnet, Port Scan, and DDoS	9997874	329

4.2. Experiments Results

In this section, we discuss and analyze the results for the compiler and the data plane experiments. Figure 4 illustrates the results for the compiler experiments. In 4. We can see the performance of the P4-ONIDS compiler for two different rulesets alongside all different compilation stages in terms of the resulting rule counting. First, given the current syntax limitation as previously discussed, results show that the compiler can

process between 83% to 89% of both rulesets as valid rules. Second, the generated P4 Rules from the valid NIDS rules show an increase of more than 400x in rules count, thus requiring additional steps to reduce it further. Third, the deduplication shows the real number of different rules generated are from 1.5x to 7.4x greater than the valid NIDS ruleset. It also illustrates that many valid rules alongside different NIDS configurations can increase P4 generated rules. Finally, after applying the aggregation step where rules are combined when they are contained on others, it shows a reduction from 3.4x to 21.5x compared to the valid ruleset.

In Figure 4.B, we compare the number of P4 rules generated with P4ID and P4-ONIDS against the two rulesets. Because the usage of port range matches with the additional aggregation step, P4-ONIDS causes a significant reduction of generated P4 rules, between 5x to 188x, if compared to P4ID. One of the main reasons for the high number of rules generated by P4ID parser is the usage of the ternary type for port range. For example, a rule with port range defined as [1024:65535] generates six different ternary matches *0X400/0XFC00, 0X800/0XF800, 0X1000/0XF000, 0X2000/0XE000, 0X4000/0XC000, and 0X8000/0X8000*. In contrast, P4-ONIDS uses only one match *1024→ 65535* to represent this range. Additionally, if ternary is used for both source and destination range ports, the final number of generated rules will be the combination of all variations.

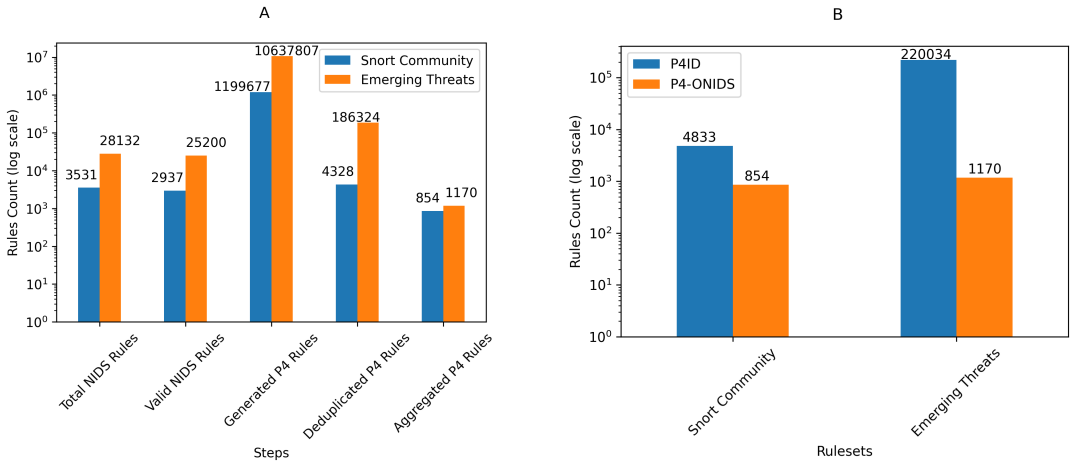

Figure 4. Compiler results

In the first scenario of the data plane experiments, we analyze the impact of different numbers *N* of packets that are sent to the NIDS upon match using the P4-ONIDS sketch-based implementation. The average loss of packets during all experiments was 0,35%, thus not impacting the overall results. Figure 5.A shows the percentage of filtered packets while Figure 5.B shows the percentage of generated alerts in comparison to the expected number of packets and alerts described in Table 1. Scenario 1 results show a tendency of a logarithm curve when the number of packets *N* increase, meaning that as *N* increases, it becomes less relevant on the overall number of filtered packets and can eventually saturate, in this case, close to 55%. Figure 5.A also shows that for all PCAP datasets, this filtering strategy has a consistent impact on the volume of packets filtered. Additionally, Figure 5.B show that as we increase the number *N*, the accuracy also increases for most of the PCAP datasets. It can be observed that when the traffic

filtering with N=100 is on average approximately 40%, the accuracy of the alerts for Tuesday and Friday datasets goes near 90%.

The Wednesday PCAP dataset accuracy remains low even with the increase of N and the number of filtered packets. The reason is that this dataset is predominantly composed of DoS and DDoS attacks. The majority of Wednesday reference alerts are from a rule of SID 40063 that is only triggered upon a threshold rate of 200 packets per second is exceeded. Since our current strategy relies on sending the first N packets of a flow upon match and flows are inactivated after a time duration D, this strategy is not adequate for these types of rules that depend on a specific traffic rate to be triggered. To overcome this limitation, we plan to evolve the P4-OINDS compiler and data plane to have a similar filtering mechanism, but with the difference of having the ability to specify the number of packets N and a forced timing deletion at the rate defined per rule.

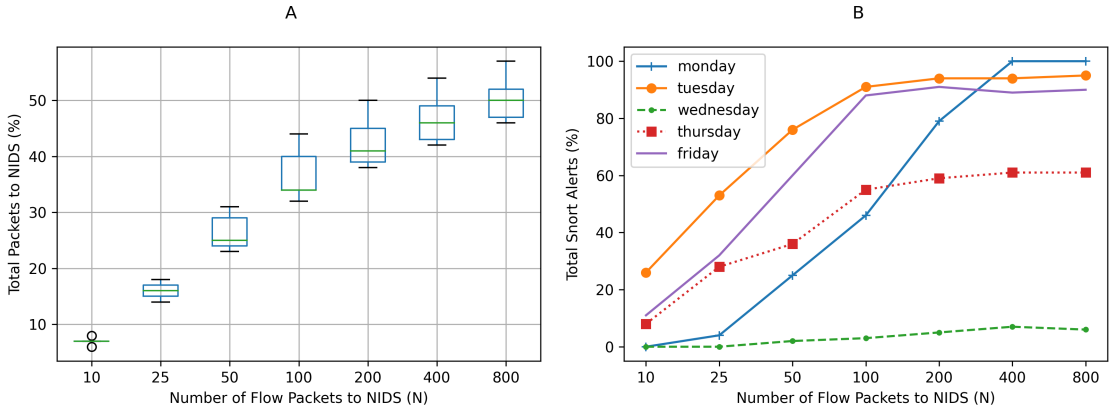

Figure 5. Scenario 1: Filtered Traffic vs NIDS Accuracy all PCAPs

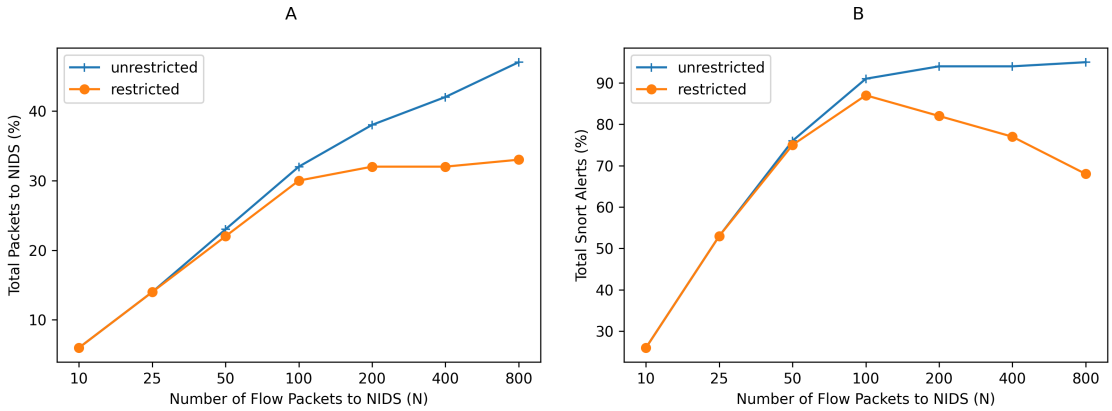

Figure 6. Scenario 2: Filtered Traffic vs NIDS Accuracy for Tuesday PCAP

Finally, in scenario 2, we restrict the Snort NIDS host bandwidth and compare the results with the same Tuesday PCAP dataset experiment with unrestricted bandwidth. The goal is to understand the impact on accuracy when restricting the traffic sent to the NIDS. Figure 6 shows that the restricted testbed starts saturating when N is equal to 100 packets and reaches 30% of packets filtered. The accuracy of generated alerts also starts dropping as more packets are dropped with the increase of saturation. In contrast, the unrestricted testbed continues to improve accuracy while packets filtered increase by 15%. It shows that network filtering is important for accuracy on bandwidth saturation scenarios.

5. Conclusion

Several works explored the ability to offload NIDS pre-filtering capabilities to hardware platforms to reduce resource usage saturation and improve detection accuracy. Among them, network data plane solutions in SDN aim to leverage the hardware speed and the recent flexibility of programmable switches. Due to the limited resources on the data plane, those solutions can be vulnerable to buffer saturation attacks when tracking suspicious flows and susceptible to reducing NIDS accuracy because the number of rules to be offloaded is greater than the resources available in the data plane.

To solve this problem, we present P4-ONIDS. It relies on an optimized NIDS to P4 rules compiler to reduce the number of P4 rules installed on the data plane, and a data plane implementation based on probabilistic data structures (sketches) to track and filter the number of packets redirected of suspicious flows. Results have shown a 4x to 188x reduction of generated rules in comparison with another work generation strategy. In our experiments with the data plane under different configurations, P4-ONIDS filtering strategy presents an accuracy level of approximately 90% with 40% of packets filtering. Future work includes improving the compiler capabilities to include more types of rules and modifying the sketch-based filtering solution to enhance accuracy for rules triggered by rate thresholds like commonly used in DDoS scenarios. Additionally, we intend to evaluate P4-ONIDS on a real testbed using P4 switch hardware.

References

Bonomi, F., Mitzenmacher, M., Panigrah, R., Singh, S., and Varghese, G. (2006). Beyond bloom filters: From approximate membership checks to approximate state machines. *ACM SIGCOMM Computer Communication Review*, 36(4):315–326.

Bosshart, P., Daly, D., Gibb, G., Izzard, M., McKeown, N., Rexford, J., Schlesinger, C., Talayco, D., Vahdat, A., Varghese, G., et al. (2014). P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review*, 44(3):87–95.

Chen, X., Wu, Y., Xu, L., Xue, Y., and Li, J. (2009). Para-snort: A multi-thread snort on multi-core ia platform. *Proceedings of Parallel and Distributed Computing and Systems (PDCS)*.

Cormode, G. and Muthukrishnan, S. (2005). An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75.

De Oliveira, R. L. S., Schweitzer, C. M., Shinoda, A. A., and Prete, L. R. (2014). Using mininet for emulation and prototyping software-defined networks. In *2014 IEEE Colombian Conference on Communications and Computing (COLCOM)*, pages 1–6.

El-Bakry, H. M. and Mastorakis, N. (2010). Fast packet detection by using high speed time delay neural networks. In *Proc. 10th WSEAS Int. Conf. Multimedia Systems and Signal Processing*, pages 222–227.

Erlacher, F. and Dressler, F. (2018). Fixids: A high-speed signature-based flow intrusion detection system. In *NOMS 2018-2018 IEEE/IFIP Network Operations and Management Symposium*, pages 1–8. IEEE.

Hoque, N., Bhuyan, M. H., Baishya, R. C., Bhattacharyya, D. K., and Kalita, J. K. (2014). Network attacks: Taxonomy, tools and systems. *Journal of Network and Computer Applications*, 40:307–324.

Khalil, G. (2015). Open source ids high performance shootout. https://www.sans.org/reading-room/whitepapers/intrusion/open-source-ids-high-performance-shootout-35772.

Kohler, E., Morris, R., Chen, B., Jannotti, J., and Kaashoek, M. F. (2000). The click modular router. *ACM Transactions on Computer Systems (TOCS)*, 18(3):263–297.

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.

Lin, C.-H., Liu, C.-H., Chien, L.-S., and Chang, S.-C. (2012). Accelerating pattern matching using a novel parallel algorithm on gpus. *IEEE Transactions on Computers*, 62(10):1906–1916.

Lopez, M. A. and Duarte, O. C. M. (2015). Providing elasticity to intrusion detection systems in virtualized software defined networks. In *2015 IEEE International Conference on Communications (ICC)*, pages 7120–7125. IEEE.

Nam, K. and Kim, K. (2018). A study on sdn security enhancement using open source ids/ips suricata. In *2018 International Conference on Information and Communication Technology Convergence (ICTC)*, pages 1124–1126. IEEE.

P4App (2020). https://github.com/p4lang/p4app. (visited on Mar. 14, 2021).

Panigrahi, R. and Borah, S. (2018). A detailed analysis of cicids2017 dataset for designing intrusion detection systems. *International Journal of Engineering & Technology*, 7(3.24):479–482.

SDxCentral (2020). Software-Defined Networking (SDN) Definition. https://www.sdxcentral.com/networking/sdn/definitions. (visited on Feb, 2020).

Song, H., Dharmapurikar, S., Turner, J., and Lockwood, J. (2005). Fast hash table lookup using extended bloom filter: an aid to network processing. *ACM SIGCOMM Computer Communication Review*, 35(4):181–192.

Teofili, S., Nobile, E., Pontarelli, S., and Bianchi, G. (2011). Ids rules adaptation for packets pre-filtering in gbps line rates. In *Trustworthy Internet*, pages 303–316. Springer.

Open Network Foundation. (2013). Openflow switch specification v1.5. https://opennetworking.org/software-defined-standards/specifications/.

Wan, Z., Liang, G., and Li, T. (2012). Multi-core processors based network intrusion detection method. *Journal of Networks*, 7(9):1327.

Xing, T., Huang, D., Xu, L., Chung, C.-J., and Khatkar, P. (2013). Snortflow: A openflow-based intrusion prevention system in cloud environment. In *Research and Educational Experiment Workshop (GREE), 2013 Second GENI*, pages 89–92. IEEE.