Computer Networks 213 (2022) 109116

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Which open-source IDS? Snort, Suricata or Zeek

Abdul Waleed, Abdul Fareed Jamali *, Ammar Masood

Devices and Network Security Lab, National Center for Cyber Security, E-9, Air University, Islamabad, 44000 Capital Territory, Pakistan

ARTICLE INFO

Keywords:
Intrusion Detection and Prevention System (IDPS)
Network Intrusion Detection and Prevention System (NIDPS)
Network security
Performance assessment

ABSTRACT

Driven by the high and diverse network traffic and increase in the number of active attackers, nearly every organization including government institutions and enterprises are forced to deploy Network Intrusion Detection and Prevention Systems (NIDPS). It becomes a challenging task for NIDPS to process high-speed traffic that results in packet drops and consequently leads to degraded system performance. Thus, necessitating the requirement to assess the NIDPS performance in terms of various parameters such as resource consumption, packet processing, packet drop rate, throughput, and latency. In this study, we scrutinized three Open-Source Intrusion Detection and Prevention Systems (IDPS) Snort (both variants: single-threaded and multi-threaded), Suricata, and Zeek; while, using similar performance parameters normally used to evaluate commercial IDPS solutions. Such a comprehensive performance review has not been considered in the prior works. We considered typical NIDPS deployment in Small and Medium-sized Enterprises (SMEs) and conducted system assessment by using variations of various factors such as packet capturing modules, detection engine algorithms, packet size, and ruleset size. Such a thorough analysis is expected to be of great value for both the practitioners and the researchers in the selection and further enhancement of optimum IDPS configuration as per their requirements.

1. Introduction

An increase in the number of networking devices and readily available internet to every device has rapidly increased the share of internet usage throughout the globe [1]. Wide-scale network connectivity has led to a tremendous increase in end-users; thus, bringing people and communities closer through ease in communication [2]. Computer networks have now become a crucial part of nearly any business or organization [3]. Enterprises, governments, e-commerce, and many other businesses use these networks to build knowledgeable, convoluted information highways which amalgamate diverse technologies such as file sharing, remote access, data protection, distributed data storage systems, access to the remote information through web services, instant messaging, Voice over IP (VoIP) and Internet of Things (IoT) [4].

Driven by the high and diverse network traffic and increase in the number of active attackers, nearly every organization requires the deployment of network security solutions to protect its network [5,6]. Firewalls and Network Intrusion Detection and Prevention Systems (NIDPS) are two popular network security solutions that are normally used in conjunction. Setting up a secure perimeter between trusted and untrusted zone, and then enforcing policy is the crucial and basic firewall role. While firewalls act as the first line of defense by actively monitoring all traffic; yet, defense in depth entails coupling of firewalls with other security devices such as NIDPS for deeper inspection of packets for malicious content [7,8]. As NIDPS undertake complex packet processing it becomes important to ensure that the deployed solutions are able to efficiently differentiate between legitimate and malicious traffic, at the network line rate to assure that the processing overheads do not lead to throughput throttling.

The general block level diagram of any NIDPS solution can be represented as shown in Fig. 1. There are a total of six different modules:

1. Packet Capturing Module: It is the first building block in any NIDPS solution which is used for packet acquisition from the network. The module can be used in two configurations: inline mode for NIDPS or passive mode (port mirroring based) for NIDS (no support for prevention). Multiple variants exist for implementing this module such as Libpcap which only supports passive mode; whereas, AF_Packet supports both the inline as well as passive mode. In passive mode (NIDS) packets could be temporarily stored in buffers while awaiting processing but in the inline mode, packets have to be processed in real-time which limits the performance of NIDPS.
2. Packet Decoder Module: Packet Capturing module is then followed by Packet Decoder which applies protocol-specific rules to decode contents and further establish packets conformance with standards. As an example for packet sizes different than the

* Corresponding author.
E-mail addresses: dnslab@students.au.edu.pk (A. Waleed), ajamali.msee17seecs@seecs.edu.pk (A.F. Jamali), ammar.masood@mail.au.edu.pk (A. Masood).

https://doi.org/10.1016/j.comnet.2022.109116
Received 29 March 2022; Received in revised form 2 June 2022; Accepted 15 June 2022
Available online 22 June 2022

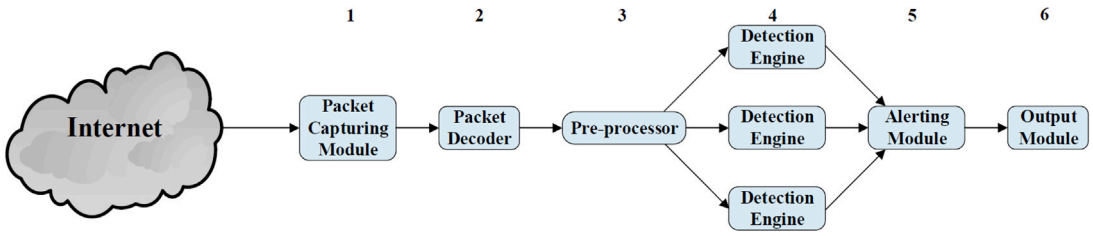


Fig. 1. A general block level representation of NIDPS.

predefined standards, this module would drop such malicious packets while generating an alert.

3. Preprocessor Module: This module provides support for both low-level and high-level protocols-based packet preprocessing. At the lower level packets are processed for defragmentation, reassembly, and session conformance; while, the high-level plugins validate packets with respect to the various application protocols (HTTP, FTP, etc.).
4. Detection Engine Module: The Detection Engine is the most crucial part of any NIDPS solution where rules are applied to compare incoming packets against defined malicious patterns. There are three types of detection engines: signature-based which uses a static rule set to identify malicious flows (misuse detection), anomaly-based which aims to identify deviations from normal behavior, and hybrid systems which are based on the combination of the prior two. Open-Source solutions by default support signature-based detection engines only, and hence our focus remains towards such detection engines for the rest of this study.
5. Alerting Module: The detection engine is followed by Alerting module that generates an alert when a rule matches with the content of a packet. This is the module that logs all the alerts to a specific folder. In the case of IDS, this module only generates an alert whereas in the IPS mode it actively blocks the malicious packets after generating alerts.
6. Output Module: It is the last module in the building block of the NIDPS solution. The output module generates statistics regarding the packets traversing the NIDPS solution. For the open-source solutions, the output module is normally based on Command Line Interface.

Every module in the pipeline is dependent on the performance of the prior module. Hence, limitations in the operations of any module in the pipeline can adversely affect the overall performance of the entire NIDPS solution. Thus it becomes important to systematically analyze the impact of each module on the performance of the complete system — the approach which has been followed in this work.

Moreover, for objective analysis, the performance review should be based on a standard, or at least a commonly-used, NIDPS testing paradigm. As NSS labs have been previously providing such a reference architecture for commercial solutions; thereby, we used their defined benchmarks for our methodical evaluation of targeted open-source NIDPS. However, most of the prior works have failed to consider such an extensive review (as discussed in detail in Section 3) and thus cannot be used for objective selection among the open-source NIDPS. Moreover, prior works have also not targeted all the current well-known open-source solutions: Snort [9], Suricata [10], and Zeek [11]. Moreover, Snort 3 stable version has not been considered in any past research as it has been recently released in January, 2021 [12].

In our work, we have conducted a performance assessment of the three open-source IDPS solutions by considering the possible variations offered by the underlying solution for all the modules depicted in Fig. 1. However, as the tested solutions only provide alternates for the two modules, Packet Capturing and Detection Engine, so varying combinations were considered for these only. Since the focus of our study remains around Small and Medium-sized Enterprises (SMEs), hence the corresponding network profiles (up to 1000 Mbps) were considered for the assessment. The System Under Test (SUT) was deployed in a testbed and performance was assessed under varying traffic conditions, while considering the alternate implementations for the Packet Capturing and Detection Engine modules. Our evaluation covered both modes of operation — detection mode (IDS) and prevention mode (IPS).

Our performance evaluation (details given in Section 4) indicates that Suricata outperforms Snort and Zeek solutions in both IDS and IPS modes. However, it is highlighted that Suricata's better results are by the virtue of its efficient exploitation of underlying hardware multi-threaded architecture. While we considered both variants for Snort, version 2.9.16 single-threaded and version 3.1.7 multi-threaded; yet, even the multi-threaded version displayed degraded performance under high traffic rates due to inefficient implementation of the multi-threaded architecture. Zeek also performed well in our tests; yet, the product is limited to IDS mode only, with no IPS support.

It is considered that the comprehensive performance review of the three open-source NIDPS products conducted by us would be beneficial to both the practitioners and researchers in not only selecting the right combination for a specific use case but also opening further avenues of research for performance improvement of tested solutions. Based on the SME network profile, our review can be used to select the optimum set of variants available for different modules. This study can be further used as a baseline to further investigate the development of newer variants for the NIDPS modules.

Our major contributions are:-

- Performance review of the open-source Snort, Suricata, and Zeek NIDPS products while considering all the possible variants of the underlying modules.
- Ensuring comprehensiveness of the assessment by basing it on an established bench-marking standard (NSS labs [13]) previously used for evaluating commercial IDPS solutions.

The rest of the paper is organized as follows. Section 2 discusses the three considered open-source NIDPS solutions individually in detail. Section 3 includes related work that has been conducted in the domain and a detailed comparison of our work with the others. The proposed performance evaluation strategy, devised to benchmark the targeted open-source solutions is elaborated in Section 4, followed by the results of our evaluation in Section 5. The results are further analyzed in Section 6, and Section 7 concludes the paper while discussing future avenues of research.

2. Open-source NIDPS

While considering the open-source NIDPS products we have targeted the current well-known solutions in this category — Snort, Suricata, and Zeek. These open-source products are widely used to protect the networks [14] and support both the IDS and IPS modes (except for Zeek that only supports IDS mode). The individual solutions are discussed in detail next.

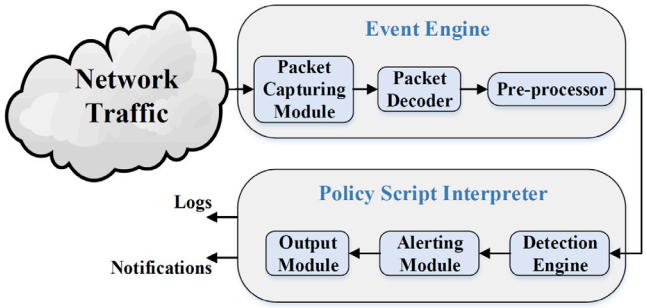


Fig. 2. Zeek block level diagram.

2.1. Snort

Snort [9] is one of the most widely used signature-based IDPS solutions which support both IDS and IPS mode. Snort is a valuable NIDPS tool that is quite easy to configure. It can monitor traffic in the network, compare the received packets against signatures, log attacks, and is also able to present attack statistics on the console if the rules are matched.

Snort uses Libpcap as default for the packet capturing stage, which is then followed by the decoder to decode packets (general NIDPS architecture indicated in Fig. 1). Normalization of packets is done by a preprocessor which converts the traffic to a form that the detection engine can understand. The detection engine applies rules to the traffic to detect if any malicious packet is present. Snort only uses misuse detection and does not support anomaly-based detection by default. In IDS mode, it only generates alerts based on detection; while, it blocks the malicious packets in IPS mode. The last component is the output block which simply generates a text file for the user to view later.

Snort was developed in 1998, and since then, it has undergone many updates facilitated by the highly active Snort community. Snort lacks Graphical User Interface (GUI), but it can be overcome by using open-source visualization tools such as Snorby [15] and Base [16]. The multi-threaded Snort variant was introduced as Snort 3, and for our evaluation, we have considered both single and multi-threaded architectures — Snort 2.9.16 single-threaded and Snort 3.1.7 (released in June 2021) multi-threaded variants.

2.2. Suricata

Suricata [10] is a very strong alternative NIDPS. Suricata supports multiple detection engines due to multi-threading (Fig. 1 block 4); thereby, it can handle more network traffic in comparison with Snort which only supports a single detection engine. In terms of signature-based detection, Suricata uses the same format as employed by Snort for rules declaration and also has similar detection algorithms. Similar to Snort, Suricata also supports both the IDS and IPS modes; while, at the output stage the alerts can be stored either in a simple text file or can be stored in JSON format.

2.3. Zeek

Zeek [11] is another open-source NIDS tool that only supports IDS mode. In the Zeek framework, agents called workers are deployed on the network devices and these workers send their logs to the manager. Zeek manager has basically two components, depicted in Fig. 2; an event engine that converts every packet that is received from the network into an event and passes that to the next component that is the policy script interpreter [17]. The main function of the policy script interpreter is to apply the Zeek rules to the events generated from the event engine; that may lead to alerts in case of detection of malicious activity.

Zeek architecture is highly scalable in which performance improvements can be easily achieved by dedicating more hardware resources to the workers and the manager [18]. Zeek coined its importance in the environments where zero-day attacks are concerned as it also supports anomaly-based detection; a feature missing in both Snort and Suricata which only support misuse detection. Zeek provides an in-depth analysis of network traffic, and it is quite an efficient, easily deployable, and flexible open-source solution but has a limited default number of signatures/rules, thus restricting its widespread use. On the contrary, Snort and Suricata have a much wider rule base with an actively contributing community; thereby, finding more acceptance for deployments.

3. Related work

Open-source NIDPS solutions, in particular Snort and Suricata, have been studied a lot in prior research; however, the treatment has not been comprehensive in every aspect. Moreover, there is very limited work on the analysis of Zeek in comparison with the other two solutions, as Zeek has been mainly used only as a network monitoring system. Prior work on the evaluation of open-source systems is elaborated next to highlight the requirement of the comprehensive review considered in this work. In this section, whenever Snort will be mentioned, it will be referred to as the single-threaded variant unless otherwise specified.

Qinwen et al. [31] compared Snort, Suricata, and Zeek open-source IDS solutions based on default configurations of Data Acquisition (DAQ) and Detection engine. While the number of parameters such as; memory/CPU utilization and packet receive/drop rate was analyzed; yet, stress testing in terms of packet size and the number of rules was missing. Moreover, IPS mode was also not considered. Although the paper reported results of the multi-threaded variant of Snort, the solution was in the beta phase, whereas, in our study, we have considered the stable version of Snort 3.

The performance comparison between Snort and Suricata at 10 Gbps network traffic has also been conducted in a study [26]. The results indicated Suricata's performance was better than Snort in its ability to process higher network loads; but, at the cost of more intense usage of hardware resources. The work was specifically targeted towards high-speed networks and did not consider generic stress testing. Moreover, Zeek was also not evaluated against the other two NIDPS.

In another work [32] Snort (both variants) and Suricata were compared on the basis of various parameters such as Memory/CPU utilization, packet processing speed, and packet dropping rate. Various experiments were conducted to evaluate the performance of Snort and Suricata by varying traffic throughput from 1 Gbps to 100 Gbps. The network traffic consisted of both normal as well as the malicious activity which was achieved using a combination of IPERF3 and pytbull. Although the work considered high network traffic for the evaluation of open-source Snort and Suricata; yet, Zeek was not evaluated, nor the systems were evaluated in the IPS mode.

Based on the packets drop rate, Memory/CPU utilization, Eugene et al. [21] compared Snort and Suricata. The outcome of their work was that Suricata's performance was better compared to the Snort but at the cost of higher computational resources. They compared Snort and Suricata under real traffic and monitored packet drop rate, Memory utilization, and CPU consumption and established that by increasing the number of CPU cores, the performance of Suricata can be improved. While the review delivered useful analysis; yet, it lacked a review of Zeek, the impact of variations in detection algorithms, and the effect of IPS mode.

A comparison of the performance of Snort and Suricata under single and multiple cores architectures has also been conducted [23]. In single-core deployment, Snort performed better than Suricata; but with multiple cores available, the performance of Suricata excelled. Suricata was thus established to be more scalable and efficient. The work again

Table 1
Scope of the analysis conducted in literature for the comparison of open-source IDPS.

SN	Ref	O-S IDPS				PCM		PMA			TP	LT	ST		Mode	
		Sn	Sn3	Su	Z	LP	AFP	BM	WM	AC			PS	RS	IDS	IPS
1	Changwei et al. [19]	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗
2	Gaurav et al. [20]	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓
3	Eugene et al. [21]	✓	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗
4	Waleed et al.[22]	✓	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✓	✗	✓	✗
5	David et al. [23]	✓	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✓	✗
6	Padmashani et al. [24]	✓	✗	✗	✗	✗	✗	✓	✗	✓	✗	✗	✗	✓	✓	✓
7	Alhomoud et al. [25]	✓	✗	✓	✗	✗	✗	✗	✗	✗	✓	✗	✓	✗	✓	✗
8	Syed Ali et al. [26]	✓	✗	✓	✗	✗	✗	✗	✗	✗	✓	✗	✓	✗	✓	✗
9	Anagnostakis et al. [27]	✓	✗	✗	✗	✗	✗	✓	✗	✓	✗	✗	✓	✓	✓	✗
10	Chris et al. [28]	✓	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✓	✓	✓	✓
11	Haiyang et al. [29]	✓	✗	✓	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓	✓	✗
12	Joshua et al. [30]	✓	✗	✓	✗	✗	✗	✗	✗	✓	✓	✗	✓	✓	✓	✗
13	Qinwen et al. [31]	✓	✗	✓	✓	✓	✓	✗	✗	✓	✗	✗	✗	✗	✓	✗
14	Qinwen et al. [32]	✓	✓	✓	✗	✓	✓	✗	✗	✓	✓	✗	✓	✗	✓	✗
15	Our Work	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Abbreviations & Terms:
AC: Aho–Corasick; AFP: AF_Packet (version 2.4.6); BM: Boyer Moore; LP: Libpcap (version 1.8.1); LT: Latency; O-S: Open-Source; PCM: Packet Capturing Module PMA: Pattern Matching Algorithm; PS: Packet Size; RS: Ruleset; Sn: Snort (Single-threaded); Sn3: Snort3 (Multi-threaded); ST: Stress Testing; Su: Suricata; TP: Throughput; WM: Wu-Manber; Z: Zeek.

lacked comprehensive analysis based on the possible variations in underlying NIDPS modules and also missed stress testing.

Alhomoud et al. [25] analyzed the performance of Snort and Suricata on three different platforms (ESXi server, Linux 2.6, and free BSD) while keeping the traffic rate up to 2 Gbps. Review of Zeek, variations in Pattern Matching Algorithm and Packet Capturing Module, stress testing, and IPS mode were not covered.

An FPGA-based technique was proposed for the implementation of an efficient, reliable, high speed, and scalable network intrusion detection system based on Snort [28]. The work was focused on stress testing of Snort only and lacked a similar analysis for Suricata and Zeek.

Waleed et al. also considered the stress testing of Snort under high traffic volume by implementing a real network in which they analyzed the packet receiving and dropping rate [22]. Keeping in view the high packet drop at increased traffic rate, the work proposed a parallel architecture. Again the work remained focused on Snort only.

In another work Snort and Suricata were compared based on stress testing [30]. A rigorous test framework was considered to analyze Snort and Suricata by scaling the system's computational resources such as the number of CPU cores while increasing the network traffic. The results indicated Suricata to be better than Snort; but, again the work lacked the comparison of Zeek and also did not consider the IPS mode.

Gaurav et al. reviewed Snort with respect to its capability to block brute force attacks [20]. Snort was used in IPS mode with AF_Packet as its packet capturing module; while its ruleset was augmented to provide protection against Denial of Service (DoS) attacks. Stress testing and analysis of other NIDPS were not considered in the work.

Implementation of a distributed Snort in a campus network was conducted in one of the works to improve the speed and accuracy of the NIDPS [19]. The distributed architecture was achieved through protocol analysis and multiple detection engines but further stress testing was not carried out. Moreover, the work was limited to Snort only.

Haiyang et al. [29] proposed a parallel IDS architecture exploiting the features of multi-core processors to achieve high throughput, flexibility, reliability, and scalability. The review did not consider Zeek and also lacked stress testing.

Performance comparison of Snort with different pattern matching algorithms has also been undertaken [27]. The review indicated that the pattern matching algorithm efficiency is dependent on the packet size and memory cache of the host system. Snort remained the center of the work and other open-source solutions were not considered.

Padmashani et al. proposed a method to handle DOS flooding [24]. They proposed a Snort that uses the Aho–Corasick pattern matching algorithm which is better in performance compared to the traditional Snort and named it BSnort. Again, solutions other than Snort were not reviewed.

The review of prior work on the subject, as consolidated in Table 1, highlights that most of the previous research lacked a thorough evaluation of all the three well-known open-source solutions: Snort (both variants), Suricata, and Zeek. Thereby, in this work, we have conducted a rigorous performance review of the open-source Snort (both variants), Suricata, and Zeek NIDPS products while considering all the possible variants of the underlying modules. Moreover, comprehensiveness of the assessment is ensured by basing it on an established bench-marking standard (NSS labs [13]) previously used for evaluating commercial IDS solutions. Next, we discuss in detail the main contours of our evaluation strategy.

4. Evaluation strategy

Both variants of Snort, Suricata, and Zeek all support IDS mode but the former two solutions also support IPS mode, which Zeek lacks. For the comprehensive evaluation of the selected Systems Under Test (SUTs), the impact of the different underlying modules (Fig. 1) on NIDPS performance was analyzed through a set of test runs. The details are presented next.

4.1. Impact of modules

As the selected open-source solutions only allow adaptations in the Packet Capturing Module and Detection Engine (Pattern Matching Algorithm); thus, the same have been varied across the different test runs to analyze the impact of specific selection on the system performance.

For the Packet Capturing Module, two widely used options, Libpcap (version 1.8.1) and AF_Packet (version 2.4.6), have been considered. Although there exist other variants such as DPDK and PF_RING but these are quite much hardware dependent [33–35]; thus not considered as their usage would entail restricting the users to a specific hardware platform.

Both Snort variants and Suricata have been studied in IDS mode with the Libpcap and AF_Packet integrated. Zeek does not provide support for the AF_Packet; hence, it has only been evaluated with the Libpcap. A simple test profile was considered where UDP traffic was injected into the SUT and **packet loss** for the variations in Packet Capturing Module was analyzed while keeping all the other modules in the default condition. For the subsequent analysis based on variations of Detection Engine, the Packet Capturing option (Libpcap or AF_Packet) that provided the lowest packet loss was picked.

While considering the options for Detection Engine, Snort 2.9.16 was evaluated with all the three supported pattern matching algorithms — Boyer Moore, Aho–Corasick, and Wu-Manber; but, both Suricata and Zeek were assessed with the single default algorithm. As Aho–Corasick supports various instances such as ac-split, ac-q, and ac-bnfa-q so initially the SUT was evaluated for all the possible options by considering **packet loss** as the key parameter. The best configuration of Aho–Corasick was then further compared with the other two options (Boyer Moore and Wu-Manber) for Snort 2.9.16 as SUT.

It is highlighted that our evaluation of possible instances of Aho–Corasick was carried out only for Snort 2.9.16, as the best option would also perform similarly in Snort 3 and the other two NIDPS by virtue of the improved performance of the underlying algorithm for that instance. Moreover, for all the evaluations of Module level variations, the three SUTs were considered with a default rule set of 10k rules. As Zeek has only around 800 rules in the initial configuration, the same were duplicated to increase the ruleset size to 10k rules so that the three SUTs could be equally compared.

4.2. Performance benchmarking

Based on the best combination of the modules, after evaluating the possible variants as discussed in Section 4.1, further performance benchmarking of SUT's was carried out. The details of our approach are presented next.

IDS Mode Comparison: As NIDPS could be operated both in IDS and IPS mode, first we compared the performance of the three SUTs under IDS mode. All the systems were evaluated with the same number of rules in the ruleset (42k rules). It is highlighted that by default, Snort and Suricata are configured with 10k rules so the additional rules were augmented from the Snort community [36] and the website of Emerging Threats [37] which releases new rules on a regular basis that both Snort and Suricata support. Zeek has around 800 rules by default and it does not support Snort/Suricata rules; moreover, there is no community support for extending its ruleset. Thereby, for Zeek additional rules were duplicated from already available rules. The higher rule base, in comparison with the default 10k ruleset, was considered to carry out stress testing of the SUT detection engine.

Latency Based Comparison: This comparison has been inspired by the NSS evaluation methodology used to compare commercial NIDPS [13], where the considered latency values are 8, 16, 32, 64, 128, and 256 µs. The requisite latency values were generated by adjusting TCP window size and throughput and the general formula for computing latency is TCP window size divided by throughput. As an example for latency test with 16 µs, if the throughput is selected to be 512 Mbps then window size is configured to be 1024 bytes as $1024 \times 8/512$ Mbps = 16 µs. The aim of this comparison is to determine the responsiveness of SUT in being able to handle low latency traffic. For latency-based benchmarking TCP protocol was used; whereas, in the rest of the evaluations only UDP protocol was utilized. The comparison was conducted with a higher ruleset (42k) to carry out stress testing of SUT.

Effect of Different Packet Sizes on SUT: This evaluation has also been inspired by the NSS methodology. By virtue of the intrinsic relation between packet size, the number of packets, and throughput, as the packet size is reduced the number of packets per second needs to be increased so as to maintain the same throughput. Thus, keeping the same throughput and using different packet size benchmarking acts as stress testing in validating SUT performance under varying packet loads. The required adjustments for packet size and packets per second were applied to the traffic generator to maintain the required throughput and SUTs performance was evaluated with three different (layer-2) packet sizes — 512, 1024, and 1538 bytes [13]. Such an assessment is important as in practical networks the applications use different packet sizes. The test was carried out with the default (10k) ruleset.

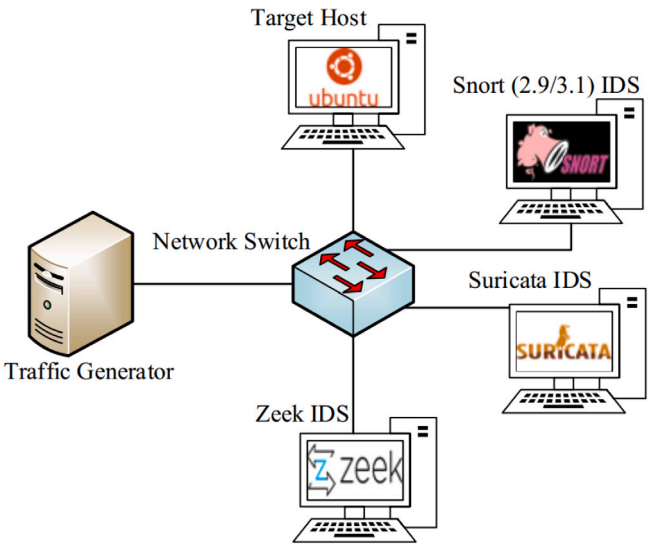


Fig. 3. Virtual Testbed setup for IDS mode.

IPS Mode Comparison: As already mentioned (Section 4) Zeek does not support IPS mode so in this evaluation only Snort and Suricata were considered. As IPS mode entails quite significantly more processing on the incoming traffic; therefore, an inefficient implementation can be easily overwhelmed on high traffic loads. Thereby, in this evaluation, we analyzed the SUTs capability to actively handle different traffic loads while considering the default ruleset of 10k rules. The packet size of 1538 bytes was chosen for the evaluation of considered SUTs in IPS mode.

For all these aforementioned evaluations the performance was primarily analyzed based on **packet loss**; whereas, two other secondary parameters Memory utilization and CPU consumption were also considered. All the test runs were conducted by increasing network traffic load (UDP traffic in general, but TCP in case of Latency comparisons) from 100 Mbps to 1000 Mbps with increments of 100 Mbps so as to carry out stress testing of the SUT. Details of the testbed used for subject evaluation are presented next.

4.3. Testbed setup

For the study of the Impact of modules (Section 4.1) and Performance benchmarking (Section 4.2), except for evaluation of detection engines and IPS mode comparison between Snort and Suricata, we have utilized the testbed depicted in Fig. 3. The testbed consisted of Traffic Generator (Ostinato), Target Host, and SUTs; Snort (both variants), Suricata, and Zeek deployed in IDS mode. The setup was deployed in VMWare Workstation 15 Pro (version 15.5.6 build-16341506), and the details of each Virtual Machine along with the hardware resources dedicated are indicated in Table 2. Only one SUT at a time was powered up during the specific process of its evaluation, while the other SUTs depicted in Fig. 3 were kept in shutdown mode.

Fig. 4 represents the architecture used for the evaluation of detection engine pattern matching algorithms and IPS mode comparison

Table 2
Software solutions and respective hardware resources dedicated.

Virtual machines	Hardware resources	Software solutions
Windows 10	2 CPU cores, 4 GB RAM	Ostinato
Ubuntu 18.04	4 CPU cores, 16 GB RAM	Snort (2.9.16)
Ubuntu 18.04	4 CPU cores, 16 GB RAM	Snort (3.1.7)
Ubuntu 18.04	4 CPU cores, 16 GB RAM	Suricata (5.0.0)
Ubuntu 18.04	4 CPU cores, 16 GB RAM	Zeek (3.0.11)
Target Host	2 CPU cores, 4 GB RAM	Ubuntu (18.04)

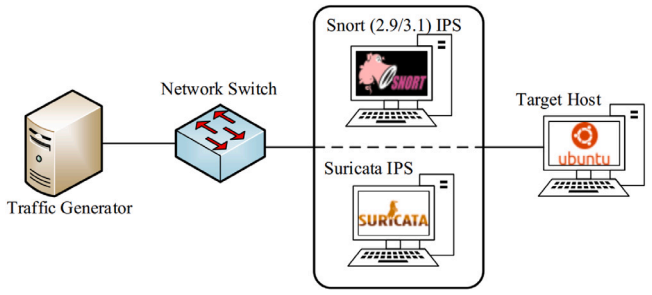


Fig. 4. Virtual Testbed setup for IPS mode.

among both variants of Snort and Suricata. Zeek was not considered for the IPS mode testing since the solution does not support inline mode. This testbed setup was also deployed in VMWare Workstation 15 Pro (version 15.5.6 build-16341506), and the details of each Virtual Machine along with the hardware resources dedicated are again given in Table 2. Only one SUT at a time was powered up during the specific process of its evaluation, while the other SUTs depicted in Fig. 4 were kept in shutdown mode.

In the testbed only legitimate traffic was generated by using Ostinato [38] traffic generator. Network traffic generator Ostinato can be used in two modes to generate legitimate traffic; normal mode and burst mode. In our work, we have utilized the normal mode of the Ostinato only, as the subject mode provides controlled traffic generation required as per our evaluation strategy. The underlying details of the various parameters configured in Ostinato for the required traffic profiles are provided in Appendix A.

In order to ensure that our assessment of the SUTs is rigorous and repeatable, each test was conducted five times and the results, as given in Section 5, were averaged out over the five runs. The system resource utilization was analyzed using the nmon tool [39]. Packet loss statistics of considered SUTs during various evaluation strategies were obtained under different traffic rates, from their corresponding Command Line Interface (CLI).

5. Results

Results for the test runs carried out as per the evaluation strategy of SUTs discussed in Section 4 are presented next. All the graphs in this section utilize packet loss rate as an indication of the SUTs performance.

5.1. Impact of modules

5.1.1. Packet capturing module

The comparison between Libpcap and AF_Packet in Snort 2.9.16 indicates that Libpcap starts dropping packets even at a 200 Mbps traffic rate, as depicted in Fig. 5, and the drop becomes substantial at higher traffic rates. AF_Packet shows no drops even at 1000 Mbps traffic rate, thus indicating that AF_Packet has a higher packet capturing capacity as compared to Libpcap. So, as a recommendation, the default packet capturing module Libpcap should be replaced by AF_Packet in Snort 2.9.16 to increase the packet capturing capability.

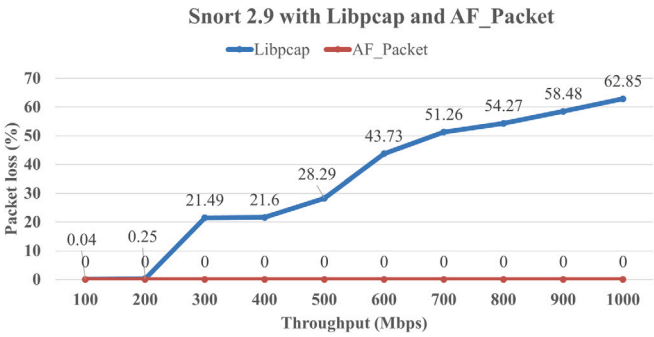


Fig. 5. Snort 2.9 with Libpcap and AF_Packet in IDS mode.

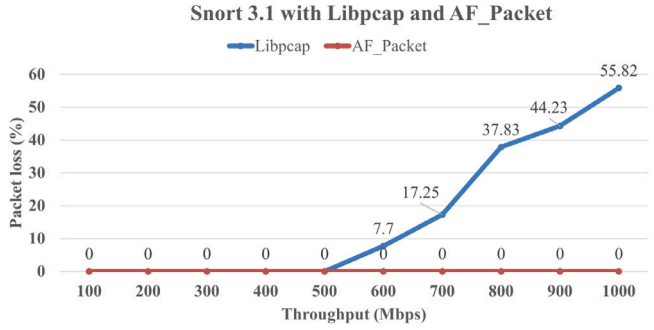


Fig. 6. Snort 3.1 with Libpcap and AF_Packet in IDS mode.

The memory utilization at 1000 Mbps traffic rate turned out to be at most 25% for AF_Packet and 15.15% for Libpcap, whereas in the case of AF_Packet peak CPU consumption is 52% and for Libpcap that value is 48%. While AF_Packet requires more memory and CPU resources; yet, the fractional increase is insignificant considering its higher packet capturing efficiency in comparison with Libpcap.

Snort 3.1.7 performance comparison between Libpcap and AF_Packet, as given in Fig. 6, indicates that Libpcap starts dropping packets after 500 Mbps, and packet loss becomes substantial at higher traffic rates. On the contrary, AF_Packet shows no packet drop even at 1000 Mbps traffic rate, indicating dominance over Libpcap based on performance. The maximum memory utilization at 1000 Mbps traffic rate turned out to be 28.7% for AF_Packet and 18.9% for Libpcap, whereas for AF_Packet, peak CPU consumption was 55%, and for Libpcap, it turned out to be 50%.

The results of the comparison between Libpcap and AF_Packet in Suricata have been plotted in Fig. 7, and indicate that Libpcap starts dropping packets right after 100 Mbps traffic rate. The packet drop of Libpcap in Suricata increases with the increase in traffic rate. AF_Packet shows no packet drop even at 1000 Mbps traffic rate; thus, AF_Packet depicts a higher packet capturing capacity when compared to Libpcap. The maximum memory utilization at 1000 Mbps traffic rate turned out to be 29.74% for AF_Packet and 19.54% for Libpcap, whereas for AF_Packet peak CPU consumption was 57%, and for Libpcap, it turned out to be 53%.

5.1.2. Pattern matching algorithm

The aforesaid discussion clearly indicates AF_Packet to be the better option for the Packet Capturing Module; thereby, for further comparison between Pattern Matching Algorithms, the Packet Capturing Module was picked to be AF_Packet. Different instances of Aho–Corasick (such as ac-split, ac-q, ac-bnfa-q, etc.) were studied in Snort 2.9.16 in IPS mode (Fig. 4), as the inline mode puts the detection engine under much more stress in comparison with IDS mode due to the simultaneous detection and blocking of packets and, thus this is the common testing paradigm for detection engines [24].

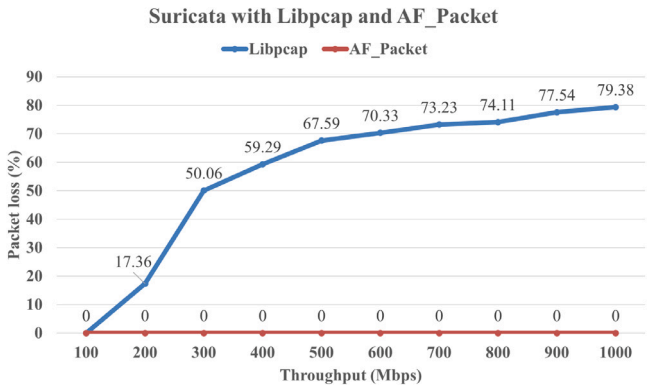


Fig. 7. Suricata with Libpcap and AF_Packet in IDS mode.

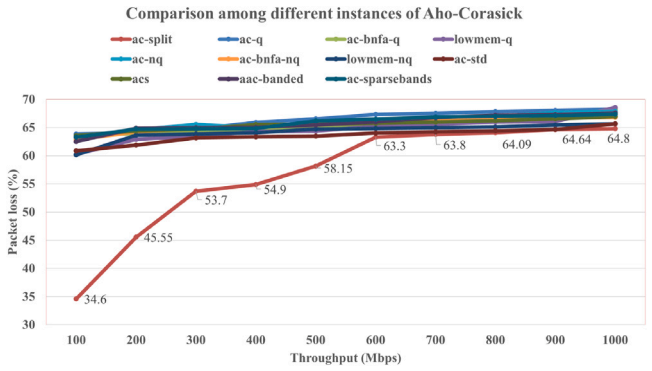


Fig. 8. Comparison of different Instances of Aho–Corasick.

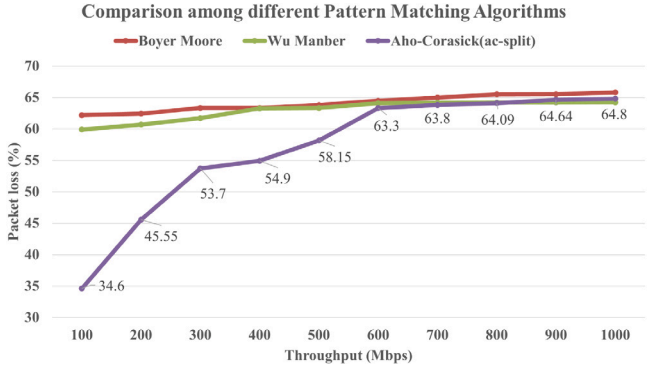


Fig. 9. Comparison of different Pattern Matching Algorithms.

The results of this evaluation for different Aho–Corasick instances, as given in Fig. 8, indicates that the detection engine starts dropping packets even at a 100 Mbps traffic rate. From the comparison, it is evident that the ac-split instance of Aho–Corasick has the highest packet processing capacity in inline mode as it depicts the lowest packet loss among all the options while utilizing a maximum of 27% memory and 55% of CPU resources. All the remaining instances of Aho–Corasick indicate similar lower performance in all traffic regimes.

After finding the best instance (ac-split) of Aho–Corasick, the comparison among different Pattern Matching Algorithms was conducted. In our work, we considered three Pattern Matching Algorithms: namely Boyer Moore, Wu-Manber, and Aho–Corasick. After fixing AF_Packet as the packet capturing module, these pattern matching algorithms were compared in real-time (IPS mode) on Snort 2.9.16.

Comparison among Boyer Moore, Wu-Manber, and Aho–Corasick (ac-split) Pattern Matching Algorithms in IPS mode, as given in Fig. 9,

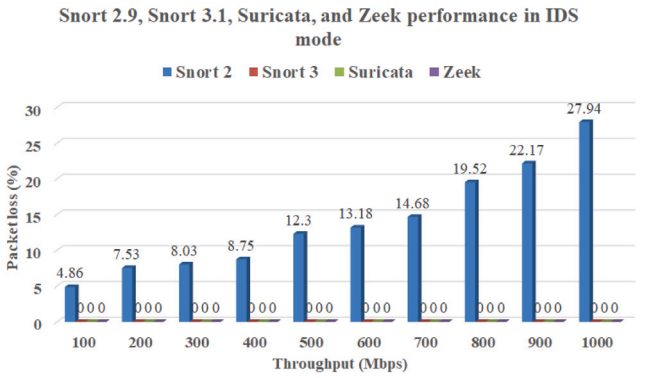


Fig. 10. Comparison of IDS solutions.

indicates that Aho–Corasick (ac-split) surpassed the other two in performance. It has much better packet processing capability as compared to Boyer Moore and Wu-Manber; while, the former two depict nearly similar packet processing capability.

It is worth mentioning, as already highlighted in Section 4, that Zeek does not support AF_Packet, and hence the impact of modules evaluation (Packet Capturing Module) did not include Zeek performance with AF_Packet. The next subsection presents SUTs performance comparison in IDS mode in which Zeek was kept in default configurations for the evaluation. However, both variants of Snort were configured with AF_Packet as Packet Capturing Module and Aho–Corasick as Pattern Matching Algorithm in Detection Engine. Similarly, Suricata was configured with AF_Packet and Aho–Corasick for evaluation purposes. Results for Performance Benchmarking of SUTs carried out as per the evaluation strategy discussed in Section 4, are presented next.

5.2. Performance benchmarking

5.2.1. IDS mode comparison

Fig. 10 depicts the comparison among Snort 2.9.16, Snort 3.1.7, Suricata, and Zeek in IDS mode with a total of 42k rules in the ruleset. Snort 2.9.16 has the highest packet loss among all the SUTs which is primarily due to its single-threaded architecture whereby it fails to process packets efficiently in the detection engine. Thus as the traffic rate increases from 100 Mbps to 1000 Mbps, packet loss sees a corresponding rise from 4.86% to 27.94%. On the contrary, Snort 3.1.7 and Suricata do not show any packet drop due to the multi-threaded architecture support.

Zeek also does not indicate any packet drop, the reason being that rules utilize limited fields; thus, requiring less processing as compared to the other two SUTs. Peak memory usage of Snort 2.9.16, Snort 3.1.7, Suricata, and Zeek was observed to be 37%, 40%, 42%, and 40% respectively, while the peak CPU consumption figures for the three SUTs were 60%, 64%, 67%, and 64% respectively.

5.2.2. Latency-based comparison

The Latency offered by SUT plays a vital role in the end-user experience and is considered an important parameter for the commercial NIDPS evaluation [13]. As already highlighted in Section 4.2, the SUTs were evaluated in IDS mode by generating latency using the window size in TCP protocol.

Results, as depicted in Fig. 11, again indicate Snort 2.9.16 to be the worst performer due to the incapability of pattern matching algorithm (Detection Engine) to efficiently compare packets against the higher 42k ruleset. The single-threaded architecture of Snort 2.9.16 is the major bottleneck in the performance and hence the higher packet loss as the latency decreases. Snort 3.1.7 and Suricata do not drop any packets because of the multi-threading architecture support whereas,

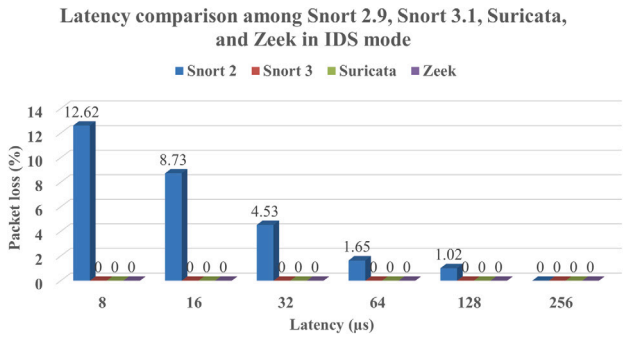


Fig. 11. Latency Comparison of IDS solutions.

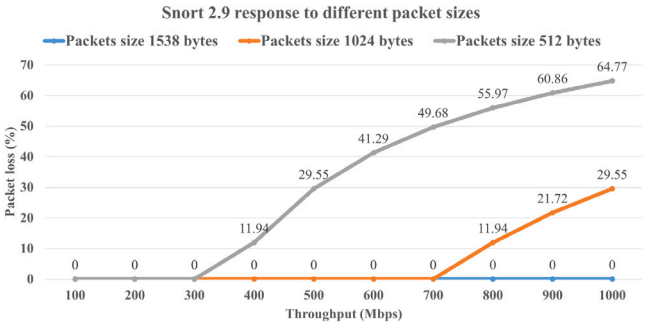


Fig. 12. Snort 2.9 performance with different packet sizes.

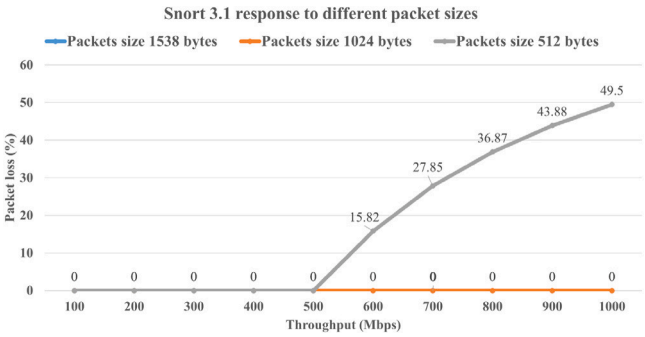


Fig. 13. Snort 3.1 performance with different packet sizes.

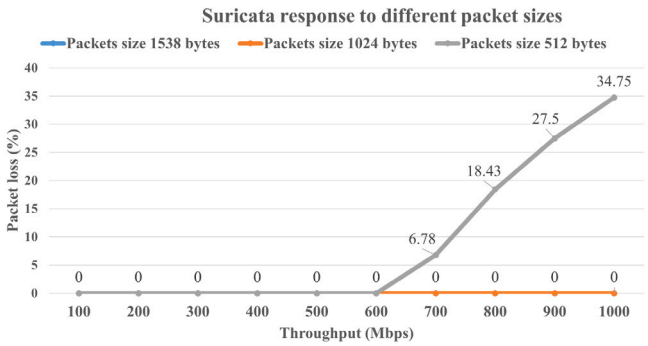


Fig. 14. Suricata performance with different packet sizes.

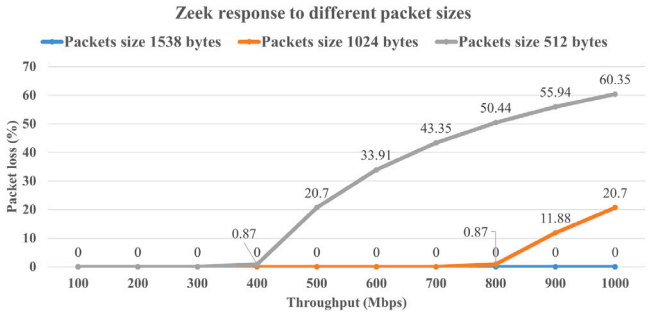


Fig. 15. Zeek performance with different packet sizes.

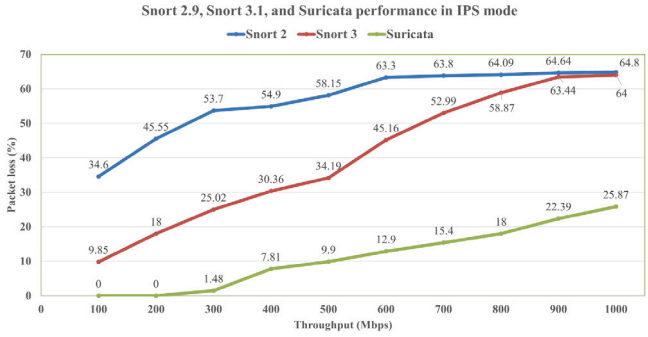


Fig. 16. Snort 2.9, Snort 3.1, and Suricata comparison in IPS mode.

Zeek also exhibits similar performance as it utilizes limited fields inside rules. Peak memory usage of Snort 2.9.16, Snort 3.1.7, Suricata, and Zeek was observed to be 37%, 40%, 42%, and 40% respectively while the peak CPU consumption figures for the SUTs were 52%, 54%, 57%, and 55% respectively.

5.2.3. *Effect of different packet sizes*

The SUTs were tested with different network packet sizes to assess their performance in diverse networks. The performance was analyzed with three different packet sizes; 512 bytes, 1024 bytes, and 1538 bytes for the evaluation. The objective of performing such a test was to study the bottlenecks (if any) of three NIDPS solutions when processing smaller packets.

By the virtue of the intrinsic relation between packet size, the number of packets, and throughput, as the packet size is reduced, the number of packets per second needs to be increased to maintain the same throughput. Thus, keeping the same throughput and using different packet sizes benchmarking acts as stress testing in validating SUT performance under varying packet loads. Figs. 12–15 represent the Snort 2.9.16, Snort 3.1.7, Suricata, and Zeek performance, respectively under different packet sizes.

Snort 2.9.16 and Zeek cannot process smaller packets size efficiently due to their single-threaded architecture; and hence, the degraded performance of Snort 2.9.16 and Zeek as the packet size decreases. On the contrary, Snort 3.1.7 and Suricata can process even smaller packets due to their multi-threading capability, but they also start to drop packets while processing 512-byte packets at a high traffic rate. However, Suricata did demonstrate better performance in comparison with Snort 3.1.7 despite both being based on multi-threaded architectures. Snort 2.9.16 utilized a peak memory of 25% and CPU consumption of 52%, Snort 3.1.7 utilized a peak memory of 28.7% and CPU consumption of 55%, Suricata utilized a peak memory of 30% and CPU consumption of 57%, and Zeek indicated peak memory usage of 27% and CPU consumption of 55%.

5.2.4. *IPS mode comparison*

As already mentioned in Section 4, Zeek does not support IPS mode; hence, in the IPS mode evaluation, only Snort 2.9.16, Snort 3.1.7, and Suricata was considered. As IPS mode entails significant additional processing on the incoming traffic; therefore, an inefficient implementation can be easily overwhelmed by high traffic loads.

The results for the comparison among Snort 2.9.16, Snort 3.1.7, and Suricata, in IPS mode, have been plotted in Fig. 16. Snort 2.9.16 performance declines in the IPS mode of operation even at a 100 Mbps traffic rate where it drops 34.6% of the received packets. Snort 3.1.7 comparatively performs better by dropping only 9.85% of the received packets at a 100 Mbps traffic rate. Suricata indicates no drop up to 200 Mbps, and the packet drop becomes evident right after the 300 Mbps traffic rate. The main reason behind packet drops is by virtue of the detection engine limitation in; efficiently comparing packets against the rules. Snort 2.9.16 utilized a peak memory of 27% and CPU consumption of 55%, Snort 3.1.7 utilized a peak memory of 30.7% and CPU consumption of 58%, and Suricata utilized a peak memory of 32% and CPU consumption of 60%.

6. Discussion

Based on the results (Section 5), we next analyze the limitations of the SUTs as per our assessment of the three systems.

The **Packet Capturing Module** plays a vital role in the performance of SUT since the throughput that a system supports depends on its packet capturing module, which if selected without adequate assessment, could result in a large packet drop. Libpcap, the default packet capturing module in Snort, has packet dropping issues that lead to degraded performance of the system. As in the case of Snort, AF_Packet outclasses Libpcap; thus, it is recommended that the default packet capturing module should be changed from Libpcap to the AF_Packet. Although CPU consumption and memory utilization of AF_Packet is comparatively a little higher than that of Libpcap, the performance improvement in response is quite significant compared to the incremental increase in resource consumption (Section 5.1.1).

While similar results are observed for Suricata; yet, by default the system is configured with AF_Packet, thus providing better performance than both variants of Snort in the default configuration. Zeek does not support AF_Packet, but has an option for PF_Ring, though not considered by us as it requires specific hardware platforms [35].

Our review also highlights that selection of optimal **Pattern Matching Algorithm** has a strong bearing on the overall performance of the NIDPS. While, in general, Aho–Corasick outperforms the other two — Boyer Moore and Wu-Manber; yet, specifically ac-split instance provides the best traffic handling (Section 5.1.2). It has been observed that the ac-split instance of Aho–Corasick outperforms Boyer Moore and Wu-Manber; hence, Snort and Suricata are pre-built configured with the ac-split instance of Aho–Corasick.

Results from the **IDS Mode Comparison** of the three SUTs indicate that, while a higher number of rules are required for improved misuse-based detection; yet, the same has a negative bearing on the system performance due to increased load on pattern matching (Section 5.2.1). Pattern matching algorithm requires more memory utilization at high traffic rates with the higher number of the rules, thus leading to more traffic drop.

While comparing the three SUTs with the higher number of rules (42k in comparison with default 10k), Snort 2.9.16 dropped a large number of packets because of the single-threaded architecture; whereas, Snort 3.1.7, Suricata, and Zeek did not drop any packets. The multi-threading architecture in Snort 3.1.7 and Suricata can compare a higher number of rules without dropping any packets. Zeek did not drop any packets because it uses limited fields in its ruleset as compared to Snort and Suricata.

Latency-based comparison, which has been prior used to evaluate commercial NIDPS [13], was carried out for the open-source SUTs to assess the responsiveness of the systems. Snort 2.9.16 again turned out to be the worst performer due to the incapability of the pattern matching algorithm (Detection Engine) to efficiently compare packets against the higher (42k) ruleset (Section 5.2.2). The single-threaded architecture of Snort 2.9.16 is the major bottleneck in the performance; hence, the higher packet loss as the latency decreases. On the contrary, Snort 3.1.7 and Suricata did not drop any packets because of the multi-threading architecture support; whereas, Zeek also exhibited similar performance as it utilizes limited fields in its rules.

The effect of the **different packet sizes** on the performance of SUT is a salient feature considered for the evaluation of Commercial NIDPS solutions. While our evaluation (Section 5.2.3) reinforced the fact that for the same throughput, smaller packet size would lead to more stress on the SUTs; yet, Snort 2.9.16 again turned out to be the worst performer among the three systems. When SUT processes smaller size packets with the same traffic rate, the packets per second increase due to which computational resources like CPU and Memory utilization also increase that becomes overbearing for single-threaded architectures.

Thereby, the limitation of both Snort 2.9.16 and Zeek in processing smaller size packets is by virtue of their single-threaded architecture. For instance, the results of considered SUTs for the packet size of 512 bytes portray the dominance of multi-threaded architecture over single-threaded architecture, as indicated in Figs. 12–15. Considering the results given in Fig. 12, Snort 2.9.16 starts dropping packets after 300 Mbps, whereas Zeek, on the other hand, starts dropping packets just before 400 Mbps, as depicted in Fig. 15. While Snort 3.1.7 performed better than Snort 2.9.16 and Zeek as in this case packets drop is observed only after 500 Mbps (Fig. 13); yet, Suricata outperformed even the Snort multi-threaded variant by not dropping any packets up to 600 Mbps (Fig. 14). Suricata surpasses others due to its efficient multi-threading capability, as in the case of Snort 3.1.7 inefficient load balancing on multiple cores was exhibited in comparison with Suricata for which all the cores were nearly equally loaded.

In **IPS mode**, SUTs operate in inline where the detection and protection are conducted in real-time, and then the legitimate packets are copied to the other interface. Such computation in real-time entails high processing that Snort 2.9.16 fails to handle (Section 5.2.4). Despite its multi-threading architecture, Snort 3.1.7 indicated poor performance (packet drop from 100 Mbps onwards) in comparison with Suricata which starts dropping packets in IPS mode from 300 Mbps onwards. As discussed above, the reason for Snort 3.1.7 weaker performance is its inefficient cores load balancing. It is highlighted that performance of any solution in IPS mode is heavily dependent on the efficiency of the underlying Pattern Matching Algorithm (detection engine). Thereby, the performance of Suricata is expected to be further enhanced if better Pattern Matching Algorithms are developed and integrated with the solution. As Zeek does not support IPS mode; thus, it was not compared with others.

Although the study focuses on the evaluation of the considered open-source NIDPS solutions with the same resources dedicated to every system and accordingly analyze the performance under various stress conditions; yet, there are some avenues for improving the performance of considered SUTs which does not come under the scope of our study but are worth mentioning. These various avenues include improvement in detection engine, packet capturing modules, network interface cards, and hardware dependency.

Packets spend most of the time inside the detection engine component of NIDPS, so a faster pattern matching algorithm will be able to process the packets faster. Hence, selecting an efficient pattern matching algorithm will enhance the performance of the NIDPS solution. Similarly, Packet Capturing Module is also a vital component of the NIDPS solution in determining its capability. Improper selection of Packet Capturing Module will lead to loss of packets that will severely affect the end-user experience. Thus, the detection engine and Packet Capturing Module are considered the most vital components of NIDPS in determining its performance.

Appropriate selection of network interface cards for capturing complete traffic is also a salient factor. The selection of inadequate network interface cards for higher traffic rates will result in packet loss that will affect the end-user experience. The performance of considered NIDPS solutions is also dependent on the underlying hardware. A system with

a lower CPU cycle will process the packets faster, thus enhancing the performance of the NIDPS solution. Our focus had been on comparative analysis of the three SUTs under similar environments, thus we have ensured similar hardware resources for all the SUTs. Moreover, we ensured that the selected hardware configuration meet the minimum requirements for all the three SUTs.

The main focus of our work was to conduct a comprehensive assessment of the open-source NIDPS in both IDS and IPS modes. For performance benchmarking, we compared SUTs based on features previously utilized by NSS labs for comparison of Commercial NIDPS. Our review unequivocally declares Suricata to be the best performer when compared against both variants of Snort and Zeek due to its efficient multi-threading architecture, which both variants of Snort and Zeek lack. Thus, deployment of Suricata is recommended in a high traffic environment; whereas, Snort could be utilized in low traffic environments by organizations constrained with the availability of only limited computational capability platforms. Zeek is only recommended for deployment in the Network Security Monitoring (NSM) mode due to its lack of IPS support and also due to the limited size of the ruleset with no active community support for rules writing.

7. Conclusion

The selection of an open-source NIDPS solution to protect an organization's network requires a thorough analysis of the solution's performance under different circumstances. To the best of our knowledge, there is no prior work that has comprehensively evaluated the performance of Snort (both variants), Suricata, and Zeek, based on the industrial standard followed for the evaluation of Commercial products. In this work, we have considered the Impact of the Packet Capturing Module and Pattern Matching Algorithm on the performance of the NIDPS solution, followed by the performance benchmarking through IDS mode comparison with a higher number of ruleset (42k), Latency-based comparison, Effect of different packet sizes on the performance of considered NIDPS solutions, and IPS mode comparison. For the evaluation, we deployed the considered open-source solutions in a testbed in which the traffic rate was varied from 100 Mbps to 1000 Mbps to mimic a typical SME network.

The results of our evaluation indicate that Suricata outperforms both variants of Snort and Zeek in all aspects. Such dominant performance of Suricata distinguishes the solution to be the first choice of deployment for the practitioners and the standard open-source solution that could further be researched and enhanced furthermore by the researchers. Although Suricata stands tall among considered open-source solutions; yet, it has limitations when operating in IPS mode and working under stress conditions, as indicated by our results. Hence, the need to integrate a more efficient pattern matching algorithm which we plan to explore next. As part of the future work, we aim to further analyze Suricata after integration of the Hyperscan Pattern Matching algorithm, which is expected to lead to improved performance of the system under stress conditions.

CRediT authorship contribution statement

Abdul Waleed: Conceptualization, Methodology, Investigation, Resources, Data curation, Validation, Writing – original draft. **Abdul Fareed Jamali:** Methodology, Data curation, Validation, Writing – original draft, Visualization. **Ammar Masood:** Supervision, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This research work was funded by the Higher Education Commission (HEC) Pakistan and the Ministry of Planning, Development, and Special Initiatives under the National Center for Cyber Security.

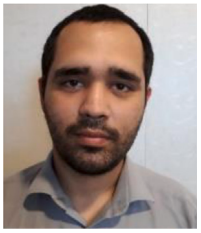
Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.comnet.2022.109116.

References

[1] Internetworldstats.com, World internet users statistics and 2021 world population stats, 2021, [online] Available at: <https://www.internetworldstats.com/stats.htm> [Accessed 2 2021].
[2] B. Wellman, J. Salaff, D. Dimitrova, L. Garton, M. Gulia, C. Haythornthwaite, Computer networks as social networks: Collaborative work, telework, and virtual community, Annu. Rev. Sociol. 22 (1) (1996) 213–238.
[3] Small Business-Chron.com, The impact of computers in small business, 2021, [online] Available at: <https://smallbusiness.chron.com/impact-computers-small-business-2139.html> [Accessed 2 2021].
[4] L.L. Peterson, B.S. Davie, Computer Networks: A Systems Approach, Elsevier, 2007.
[5] Forcepoint, What is an intrusion prevention system (IPS)?, 2021, [online] Available at: <https://www.forcepoint.com/cyber-edu/intrusion-prevention-system-ips> [Accessed 2 2021].
[6] Alert Logic, What is an IDS and why do you need it? | Alert logic, 2021, [online] Available at: <https://www.alertlogic.com/blog/what-is-a-network-ids-and-why-do-you-need-it/> [Accessed 2 2021].
[7] Trend Micro, Catch evasive threats that hide behind real network traffic, 2021, [online] Available at: <https://www.trendmicro.com/vinfo/nl/security/news/cyber-attacks/catch-evasive-threats-that-hide-behind-real-network-traffic> [Accessed 2 2021].
[8] Barracuda.com, What is an intrusion detection system? | barracuda networks, 2021, [online] Available at: <https://www.barracuda.com/glossary/intrusion-detection-system> [Accessed 2 2021].
[9] Snort.org, Snort - network intrusion detection & prevention system, 2021, [online] Available at: <https://www.snort.org/> [Accessed 2 2021].
[10] Suricata, Suricata, 2021, [online] Available at: <https://suricata-ids.org/> [Accessed 2 2021].
[11] Zeek, The zeek network security monitor, 2021, [online] Available at: <https://zeek.org/> [Accessed 2 2021].
[12] Snort Blog, [online] Available at: <https://blog.snort.org/2021/01/snort-3-officially-released.html> [Accessed 13 2021].
[13] Google Docs, Document - network intrusion prevention - test methodology v7_2.pdf, 2019, [online] Available at: <https://bit.ly/3Nzf83E> [Accessed 30 2022].
[14] Logz.io, [online] Available at: <https://logz.io/blog/5-open-source-nids/> [Accessed 22 2021].
[15] GitHub, Snorby/snorby, 2021, [online] Available at: <https://github.com/Snorby/snorby> [Accessed 22 2021].
[16] SourceForge, Base, 2021, [online] Available at: <https://sourceforge.net/projects/secureideas/> [Accessed 22 2021].
[17] Docs.zeek.org, About zeek — Book of zeek (v4.0.0), 2021, [online] Available at: <https://docs.zeek.org/en/current/about.html> [Accessed 5 2021].
[18] WLCG Security Operations Centers Working Group documentation, [online] Available at: <https://wlcg-soc-wg-doc.web.cern.ch/zeek/hardware_requirements.html> [Accessed 19 2021].
[19] C. Huang, J. Xiong, Z. Peng, Applied research on snort intrusion detection model in the campus network, in: 2012 IEEE Symposium on Robotics and Applications (ISRA), IEEE, 2012, pp. 596–599.
[20] G. Raj, M. Katoch, Security implementation through pcre signature over cloud network, Adv. Comput. 3 (3) (2012) 121.
[21] E. Albin, A Comparative Analysis of the Snort and Suricata Intrusion-Detection Systems, Naval Postgraduate School Monterey, CA, 2011.
[22] W. Bulajoul, A. James, M. Pannu, Network intrusion detection systems in high-speed traffic in computer networks, in: 2013 IEEE 10th International Conference on E-Business Engineering, IEEE, 2013, pp. 168–175.
[23] D. Day, B. Burns, A performance analysis of snort and suricata network intrusion detection and prevention engines, in: Fifth International Conference on Digital Society, Gosier, Guadeloupe, 2011, pp. 187–192.
[24] R. Padmashani, S. Sathyadevan, D. Dath, Bsnort IPS better snort intrusion detection/prevention system, in: 2012 12th International Conference on Intelligent Systems Design and Applications (ISDA), IEEE, 2012, pp. 46–51.

[25] A. Alhomoud, R. Munir, J.P. Disso, I. Awan, A. Al-Dhelaan, Performance evaluation study of intrusion detection systems, Procedia Comput. Sci. 5 (2011) 173–180.
[26] S.A.R. Shah, B. Issac, Performance comparison of intrusion detection systems and application of machine learning to snort system, Future Gener. Comput. Syst. 80 (2018) 157–170.
[27] K.G. Anagnostakis, S. Antonatos, E.P. Markatos, M. Polychronakis, E 2 xb: A domain-specific string matching algorithm for intrusion detection, in: IFIP International Information Security Conference, Springer, Boston, MA, 2003, pp. 217–228.
[28] C.R. Clark, W. Lee, D.E. Schimmel, D. Contis, M. Koné, A. Thomas, A hardware platform for network intrusion detection and prevention, in: Proceedings of the 3rd Workshop on Network Processors and Applications (NP3), 2005.
[29] H. Jiang, G. Zhang, G. Xie, K. Salamatian, L. Mathy, Scalable high-performance parallel design for network intrusion detection systems on many-core processors, in: Architectures for Networking and Communications Systems, IEEE, 2013, pp. 137–146.
[30] J.S. White, T. Fitzsimmons, J.N. Matthews, Quantitative analysis of intrusion detection systems: Snort and suricata, in: Cyber Sensing 2013, Vol. 8757, International Society for Optics and Photonics, 2013, 875704.
[31] Q. Hu, M.R. Asghar, N. Brownlee, Evaluating network intrusion detection systems for high-speed networks, in: 2017 27th International Telecommunication Networks and Applications Conference (ITNAC), IEEE, 2017, pp. 1–6.
[32] Q. Hu, S.Y. Yu, M.R. Asghar, Analysing performance issues of open-source intrusion detection systems in high-speed networks, J. Inf. Secur. Appl. 51 (2020) 102426.
[33] W. Zhu, P. Li, B. Luo, H. Xu, Y. Zhang, Research and implementation of high performance traffic processing based on intel DPDK, in: 2018 9th International Symposium on Parallel Architectures, Algorithms and Programming (PAAP), IEEE, 2018, pp. 62–68, [Accessed 17 2021].
[34] Core.dpdk.org, DPDK supported hardware, 2021, [online] Available at: <https://core.dpdk.org/supported/> [Accessed 27 2021].
[35] Ntop.org, PF_RING ZC (zero copy) — PF_RING dev documentation, 2021, [online] Available at: <https://www.ntop.org/guides/pf_ring/zc.html> [Accessed 31 2021].
[36] Snort, Downloads, 2021, [online] Available at: <https://www.snort.org/downloads#rules> [Accessed 4 2021].
[37] Rules.emergingthreats.net, Proofpoint emerging threats rules, 2021, [online] Available at: <https://rules.emergingthreats.net/> [Accessed 28 2021].
[38] Ostinato Traffic Generator for Network Engineers, OSTINATO: Traffic generator for network engineers, 2021, [online] Available at: <https://ostinato.org/> [Accessed 7 2021].
[39] Nmon.sourceforge.net, nmon for linux | main / HomePage, 2021, [online] Available at: <http://nmon.sourceforge.net/pmwiki.php> [Accessed 9 2021].



Eng. Abdul Waleed is a Research Associate at the Devices and Network Security Lab, National Center for Cyber Security, Islamabad, Pakistan. He received his bachelor's degree in Electrical Engineering in 2019 from Air University, Islamabad, Pakistan, and right after it, he joined the R&D setup (Devices and Network Security Lab). He has developed his research skills towards the assessment of different Network Security solutions based on international standards. His research interest includes Information Security, Network Security, and Machine Learning techniques.

Eng. Abdul Fareed Jamali completed his MS degree in Telecommunication and Networking from National University of Sciences and Technology, Pakistan. He received his bachelor's degree in Electrical Engineering from Air University (Pakistan) back in 2016, and since then, he has been part of different R&D projects. He is currently working as Team Lead in the Devices and Network Security Lab, National Center for Cyber Security (NCCS) at Air University, Islamabad, Pakistan. His research interest includes Information Security, Network Security, GANs, Machine Learning techniques.

Dr. Ammar Masood is the Director of Devices and Network Security Lab, National Center of Cyber Security at Air University, Islamabad, Pakistan. He completed his Ph.D. in Electrical Engineering from Purdue University, USA in 2006 and since then has worked on number of R&D projects spanning from secure networks, secure software to GSM communications. His research interests include Information security, sensors and ad hoc networks, Software Engineering, software security testing and verification, smart devices security, network security and autonomous systems.