Work Balancing vs. Load Balancing for Network IDS Parallelization

Hossein Doroud*, Tobias Wiese†, Felix Erlacher‡, and Falko Dressler*
* School of Electrical Engineering and Computer Science, TU Berlin, Germany
† CQSE GmbH, Germany
‡ Cancom a+d, Innsbruck, Austria
{doroud, erlacher, dressler}@ccs-labs.org, wiese@cqse.eu

***Abstract*—Signature-based Network Intrusion Detection Systems (NIDS) is state-of-the-art for precise attack detection. Using multiple instances of NIDS in parallel is considered the most promising solution for improving its processing speed in the scale of high speed network. This can be realized by (1) distributing the network traffic between multiple NIDS to reduce the network load per system or (2) distributing the signatures (rules) between multliple NIDS to reduce the work load per packet. In this paper, we study distribution strategies targeting application and transport layer for both traffic and rule distribution approaches. In addition, we investigate the importance of considering the processing speed optimization in the rule development phase. Our experiments show that in general traffic distribution performs slightly better in terms of packet drop and alert detection compared to rule distribution. The Transport layer distribution strategy shows traffic distribution parallelization detecting 1.6% more alerts and dropping 6% less packets. We also show that optimizing the rules sets further improves the processing speed significantly.**

I. Introduction

Attacks on IT infrastructure are ubiquitous, both targeted as well as untargeted. Despite the diversity of Network Intrusion Detection Systems (NIDS) available in the market, signature-based methods offer the highest detection rate compared to the other concepts [1]. A signature-based NIDS works with a set of "signatures" or "rules" characterizing a known attack. NIDS sequentially check the incoming network traffic against a database of rules until either a rule matches, or all rules have been checked [2]. According to Cisco, *Snort*[1] is the most popular IDS on the market with over 4 million downloads.[2] *Snort* performs deep packet inspection such that the rule options are matched against the payload of the packets, which is computationally very expensive. Thus, *Snort* suffers from packet loss in high-speed networks. This may lead to missing malicious activities [3].

The lack of performance in high-speed networks can be traced back to a combination of two factors: Firstly, in high-speed scenarios the precessing load is to high due to the high number of packets [4]. Secondly, the very high number of known attacks leads to a very large number of *Snort* rules that the traffic has to be compared to. A common solution is extending the processing capability of *Snort* via utilization of multiple machines in parallel [5–8]. Here, one way is to distribute network traffic among multiple *Snort* instances. This approach reduces the incoming traffic of each *Snort* instance. The other way is to divide the original ruleset into multiple subsets and distribute them among multiple *Snort* instances, mirroring the whole network traffic to each instance. Consequently, each *Snort* instance processes network traffic faster as less rules need to be processed for each received packet.

Several research works illustrate the positive effects of *Snort* parallelization on the overall network traffic throughput [9–11]. However, they only focus either on rule distribution or traffic distribution. A direct comparison of their performance is still missing. We aim at filling this gap by highlight the strengths and weaknesses of each approach in an extensive experimental evaluation. In short, we propose and evaluate two strategies to distribute traffic and rules between two instances of *Snort*. Each strategy splits the rules and traffic according to one layer of protocol stack. Furthermore, we compare the performance of rule distribution strategies with the performance of traffic distribution strategies by considering packet drop and detection rate. Finally, we also explore the 'fast pattern' option to speed up *Snort*.

Our main contributions can be summarized as follows:

- We propose two strategies for distributing rules and traffic in the scope of NIDS parallelization;
- we also study the processing speed of rules using the new 'fast pattern' option.

II. Background & Related Work

A. The Signature-based NIDS Snort

Snort [12–14] is the most well-known signature-based NIDS. Cabrera et al. [15] analyzed the distribution of processing times on the different subsystems of *Snort*. The results show that content matching which conducted by detection engine module, has the biggest impact on the overall processing time of a packet.

The detection engine is the core of *Snort*. In the initialization stage, the detection engine parsed and stored the rules in memory. During this process, *Snort* builds up several rule trees based on the similarity between the rules header information (e.g., network protocol, IP addresses). Later, an arriving packet

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

[2] In Cisco's security blog "Exploring Snort" (June. 2019), accessible at https://meraki.cisco.com/blog/2019/06/exploring-snort/

will be delivered along the rule tree until it ends up at the leaf that matches the packet content. This rule tree helps *Snort* to compare packets to all rules in an efficient way, by, e.g., skipping unrelated rules. Since *Snort* 2.0, rules are divided in two groups according to whether they use the *content* field or not. Rules without this field will be processed entirely at once.

Rules with content field are processed in two phases and follow the 'fast pattern' matching procedure: In the first phase, the fast, but memory consuming, Aho-Corasick [16] string matching algorithm processes the content, which is defined as 'fast pattern'. By default, the longest content of a rule is considered as 'fast pattern'. If the match failed, the detection engine will skip to check the rest of the rule in the next phase. In the second phase, the rule options are evaluated sequentially using the Boyer-Moore [17] string matching algorithm. If an option does not match, the evaluation of the rule will be aborted. Although, Aho-Corasick [16] is faster, it consumes considerably more memory and resources than Boyer-Moore [17].

B. Related Work

Utilizing hardware with a high processing capability is among the preliminary solutions. Armstrong et al. [18] proposed an FPGA-based architecture to process different *Snort* rules in a separate thread. The core of their proposal consists of a logical-based comparator that matches packets and the rules. The implementation on a medium-scale Virtex-FPGA achieves a throughput of 208 Gbit/s.

Although hardware accelerators increase the *Snort* processing speed significantly, they are relatively expensive and have limited flexibility. Changazi et al. [19] investigated the influence of configurable parameters of the Linux kernel networking subsystem. They demonstrated that changing the default value of budget B in the Linux NAPI packet reception mechanism from 300 to 14 improves the packet drop rate both at the kernel and application levels. As string matching is the most resource-demanding part of *Snort* packet process, Trivedi [20] optimized Aho-Corasick string matching algorithm which improves the matching process up 40-60%.

Shuai and Li [21] showed that the default capturing and packet processing modules of *Snort* are outdated. They designed a new data acquisition (DAQ) module based on data plane development kit (DPDK) framework [22] to capture the packet. They also integrated Hyperscan, a high-performance regular expression engine developed by Intel [23], to *Snort* detection engine. The improved *Snort* processed all the traffic and preserved the maximum detection rate at a speed as high as 1 Gbit/s.

As a considerable amount of today's network traffic consists of HTTP traffic, Erlacher and Dressler [3] proposed HTTP-based Payload Aggregation (HPA) for extracting the valuable part of HTTP traffic and filtering the rest. They show that a detection rate of 97% can be achieved by considering only the first N bytes of the flow. In this way, the packet throughput of NIDS can be increased up to 44 times.

On a macroscopic scale, running multiple *Snort* instances in parallel can improve the performance. Fulp and Farley [24] showed that the division of the NIDS rule set does have the same detection capability as the basic rule set when all the subsets together cover all rules of the original ruleset. Haugerud et al. [25] proposed dynamic rule distribution algorithms to preserve the even load distribution on each *Snort* instance facing with different traffic patterns. Limmer and Dressler [6] proposed a load balancer for realising *Snort* parallelization technique. The experimental results show that they could improve the detection performance of the NIDS by 44%.

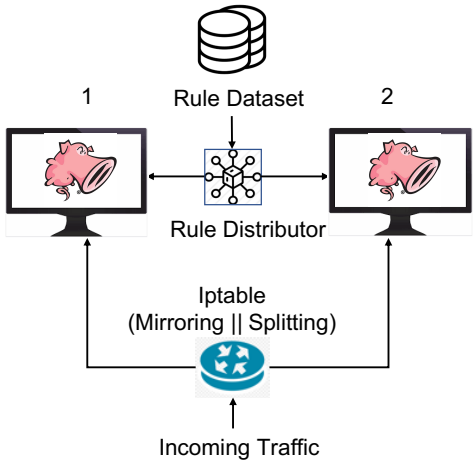


Figure 1. Parallelization approaches supported by either a traffic splitter (load distribution) or a rule distributor (rule distribution)

However, a comprehensive comparison of the distribution approaches is missing. To address this gap, we propose two distribution strategies and compare traffic distribution with rule distribution.

III. Work and Load Balancing Concepts

Traffic distribution and rule distribution are the main two approaches to parallelize NIDS operation for improving the performance of NIDS. Figure 1 depicts the required components and their collaboration. In the case of rule distribution, a rule distributor distributes the rules among the machines, which all work on identical copies of the network traffic provided, e.g., by `iptable`. In the traffic distribution case, traffic is split and distributed to different machines, all working on a full set of rules. We follow two strategies to study the pros and cons of rule and traffic distribution. We distribute the traffic based on the available label provided by our dataset. In a real life scenario, network traffic classifiers like Chain [26] can provide the label. In addition, we designed an additional experiment to study the importance of the new 'fast-pattern' option of Snort.

a) HTTP vs. Non-HTTP Traffic: The HTTP protocol represents a big part of the Internet usage. Also about 63 % of the main *Snort* rule set are HTTP related rules. However, most of the them are simple and fast to be processed [3].

b) TCP vs. Non-TCP Flows: The majority of attacks are targeted to TCP-based applications, corresponding to more than 91 % of all Snort rules. On the one hand TCP is more complex to analyze due to its connection-oriented (i.e., stateful)

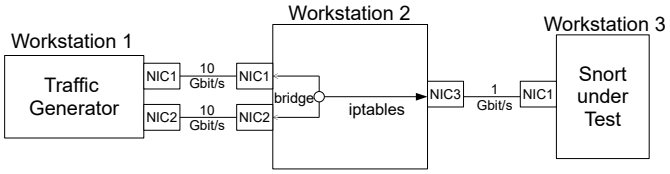


Figure 2. Test bench for performance measurements

Table I
SPECIFICATION OF THE MAIN HARDWARES OF THE TEST BENCH

	Workstation 1/2	Workstation 3
CPU	i7-3930K @ 3.2 GHz	i7-2600 @ 3.4 GHz
RAM	32 GB	16 GB
NIC	2x 82599ES 10 Gbit/s	
	1x 82541PI 1 Gbit/s	1x 82541PI 1 Gbit/s

approach. On the other hand all other protocols represent a significant amount of traffic (mostly UDP-based multimedia traffic).

c) Snort Fast Pattern Approach: Snort introduced the 'fast pattern' approach in release 2.8. Consequently, the user can define very short, but significant patterns of malicious packets that can be processed very fast with the Aho-Corasick algorithm. Our strategy is to deploy these rules on one machine (around 15 000 rules, i.e., more than 50 %), while the other processes the remaining rules.

IV. Experiment Setup and Configuration

In order to analyze the performance of *Snort* in all the load and work distribution approaches, we prepared an experimental setup as shown in Figure 2. The test bench simulates a high-speed traffic in a network that is connected to the NIDS under test. In all experiments, we first measured the performance of the first *Snort* instance with the respective traffic, followed by the second instance and the corresponding traffic to avoid artifacts of switches and online traffic splitters. As we use deterministic traffic generation, the result will not be affected.

We use three workstations running the Linux Ubuntu 16.04 operating system Table I. The traffic generator on workstation 1 sends realistic network traffic by simulating client and server side on its two NICs. Workstation 2 acts as the traffic distribution device and forwards the traffic to workstation 3 following different distribution strategies.

We use *Snort* version 2.9.11.1 with a rule set of 30039 rules. This large rule set is the combination of two smaller ones. The first one is the basic rule set available at *Snort* webpage from snapshot 29120.[3] It consist of 10740 registered rules. The second rule set is publicly published by *Emerging Threats*[4] for Snort version 2.9.0.

We rely on the packet drop rate and detection rate of *Snort* as the main metrics to evaluate its performance under different distribution strategies. For the drop rate calculation, we extract the rate of analyzed packets from the *Snort* output and compare it with the numbers of packets that *Snort* receives on the wire. In addition, we measure the alerts that *Snort* raises in an experiment and compare it with the number of alerts that *Snort* should raise for that specific traffic.

We are using `TRex`[5] in combination with `GENESIDS` [27] for traffic generation because of its ability to create definable and consistent high-speed traffic at low costs. TRex is based on the Data Plane Development Kit (DPDK)[6] and creates stateful (and stateless) layer 4 to layer 7 traffic based on traffic templates. To simulate appropriate test traffic, given traffic templates are mixed with the generated attacks of `GENESIDS`. For this purpose, we use the `sfr_delay_10_1g.yaml` traffic template that comes with the `TRex` package, which represents a realistic traffic mix normalised to 1 Gbit/s that was defined by SFR France. The template combines a bunch of typical network traffic and represents a good average of many internet applications, such as HTTP, mailing services, video calls, streaming, and pure TCP. The number of connections per second (cps) is configurable for each traffic flow to attain a desirable distribution of different application traffic flows. We include 100 generated attacks to the templates. Each attack is simulated with 1 cps. The traffic consists of roughly 60 % TCP and 40 % UDP-based traffic.

Every work and load distribution strategy was tested twice at seven ascending traffic speeds. As a metric for the traffic speed we use the packets per second (pps)-rate instead of bit rate because the pps-rate is more relevant from a systems perspective (one operation per packet). To this end, we tested the *Snort* default configuration with the traffic simulation at different pps-rates until a point that *Snort* cannot cope with the traffic anymore.

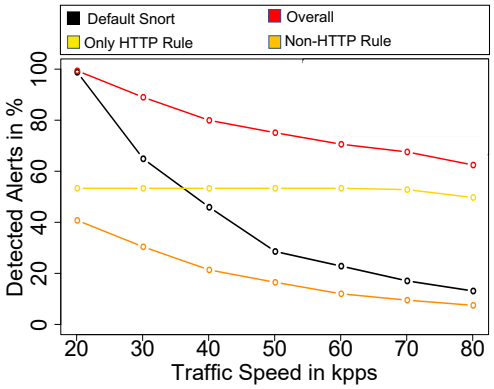
V. Evaluation

In the following, we present and discuss the performance results we obtained for the comparison of traffic and rule distribution, respectively.

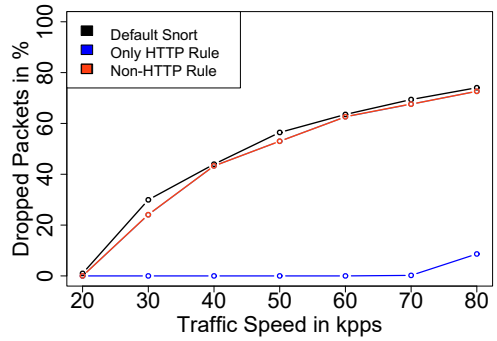
A. HTTP vs. Non-HTTP Traffic

In the first experiment, we split traffic and rules according to whether they belong to HTTP traffic. According to Figure 3, the drop rate is very low for all traffic speeds when only HTTP rules are considered. The reason behind this is that the HTTP rules can be processed very fast. On the contrary, the drop rates in the experiment with the complement ruleset is as high as *Snort* with its default configuration. The detection rate is very stable at 56% in only HTTP rule experiment, while *Snort* has to deal with high drop rates when it checks the rest of the rules. Here, the detection rate is falling monotonously from the beginning on. As a first conclusion, we can say that this strategy overly outperforms the default *Snort* configuration regarding the detection rate but it failed to distribute the load among two machines evenly.

[3]snortrules-snapshot-29120.tar.gz downloaded from https://snort.org/
[4]https://rules.emergingthreats.net
[5]https://trex-tgn.cisco.com/
[6]https://www.dpdk.org/

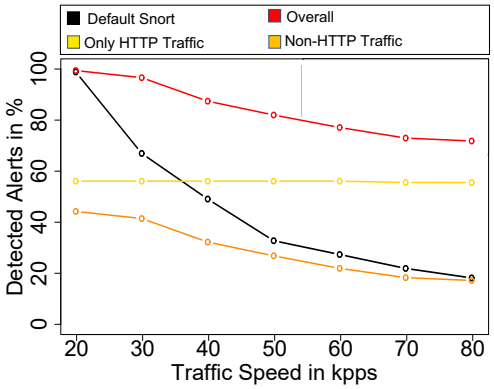


(a) Detection rates

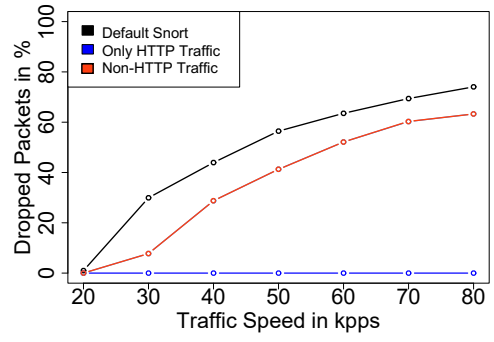


(b) Drop rates

Figure 3. Performance of the HTTP rule distribution strategy

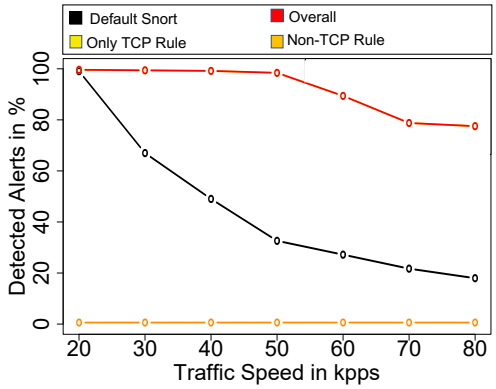


(a) Detection rates

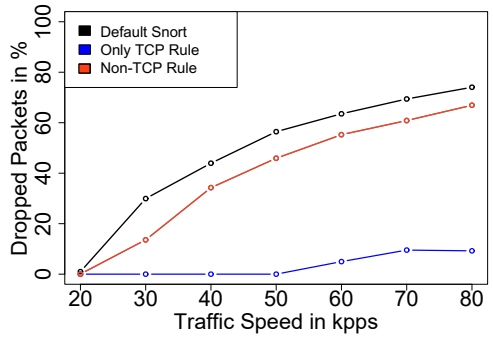


(b) Drop rates

Figure 4. Performance of the HTTP traffic distribution strategy



(a) Detection rates



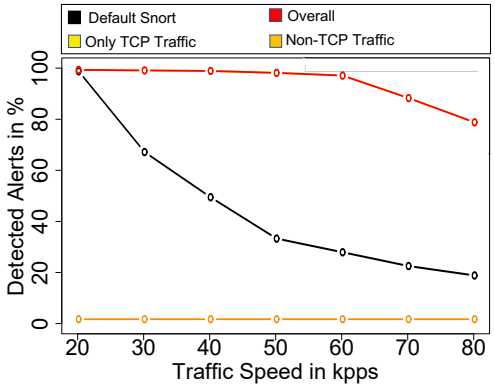
(b) Drop rates

Figure 5. Performance of the TCP rule distribution strategy

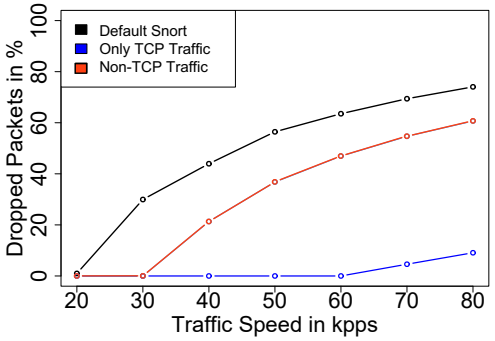
Instead of focusing on HTTP rules, we can evaluate just HTTP traffic (data on TCP ports 30, 443, and 8080) by one machine and leave the remaining traffic to be analyzed by the second machine. Figure 4 indicates that the HTTP traffic is processed extremely – the drop rate for the HTTP traffic is close to zero. *Snort* detects 56% of the attacks in the test traffic, thus, 56% of the attacks are transported via HTTP. However, the drop rate is highly increased in non-HTTP traffic processing. It is close to the drop rate of the default *Snort* setup. Consequently, the detection rate reduces gradually as the network speed is increased.

B. TCP vs. Non-TCP Flows

In the second experiment, we split traffic and rules according to whether they belong to TCP traffic. In this way, the majority of the rules (91%) are labeled as TCP rules. The results in Figure 5 show a significant improvement regarding the detection rate in TCP rule distribution. Until 50 kpps, the machine with the only TCP rules succeeds to detect all the attacks. Even at a speed as high as 80 kpp still, it can reach to 80% detection rate. Also, the drop rate is much lower than for *Snort* with the default configuration. As our dataset delivered attacks only over TCP, the detection rate of the machine with only TCP rules represents the overall detection rate as well. However, the drop rate for the remaining rules on the second IDS is almost as high as *Snort* with the default configuration. We can

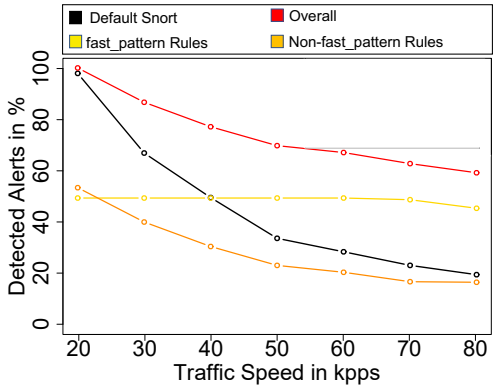


(a) Detection rates

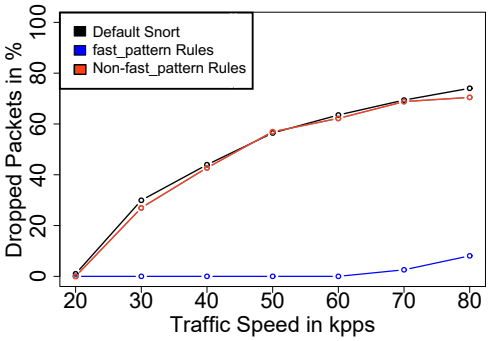


(b) Drop rates

Figure 6. Performance of the TCP traffic distribution strategy



(a) Detection rates



(b) Drop rates

Figure 7. Performance of the 'fast pattern' rule distribution strategy

conclude that the high number of TCP rules have a minimum impact on the *Snort* performance.

Another approach is to distribute TCP and non-TCP traffic, and to run *Snort* with its default configuration. The traffic splitting is done by looking at the protocol field in the IP packet headers. This way, also the state of the TCP connection is preserved. The performance shown in Figure 6 follows the same trend as with the TCP rule distribution approach. The drop rate of *Snort* analyzing only TCP traffic is very low. However, the drop rate begins to climb at 30 kpps for the non-TCP traffic. Nevertheless, it stays below the drop rate of the machine TCP rules in TCP rule distribution (cf. Figure 5b). Due to the dataset structure, the overall detection rate is equal to the detection rate of the machine that processes only TCP traffic.

We can conclude that both traffic and rule distribution based on TCP flows can maintain a 100% detection rate for higher packet rates than *Snort* with the default setup. We can manage up to 50 kpps with the rule distribution approach before *Snort* begins to miss a noticeable amount of attacks. Even 60 kpps can be reached with the traffic distribution approach.

C. Snort Fast Pattern Approach

In the third experiment, we explore the significance of considering the processing speed as a factor in the rule development phase. To this end, we use all the rules that make use of the 'fast pattern' keyword on one machine and the rest of the rules on the other one. Although more than 50% of the main ruleset makes use of the 'fast pattern' rule, the drop rate stays very low for all used packet rates. Consequently, the detection rate stays at 48.7% until 70 kpps Figure 7. For all other rules, *Snort* process one content of each rule using the Aho-Corasick string matching algorithm. The results show that the drop rates are very close to the results of *Snort* with the default configuration, even though less than half of the main ruleset was processed.

However, the trend for the detection rate is falling nearly linearly (cf. Figure 7a). This is a result of the very stable detection rate in the 'fast pattern' ruleset, and the fact, that *Snort* generates more alerts in the other ruleset.

VI. Discussion

In this paper, we illustrated the effectiveness of parallelization to improve the online performance of *Snort*. We split either the ruleset or the traffic according to two different protocol layers.

Our results show that any prior knowledge regarding the nature of expected attacks is highly valuable and can help to speed-up the processing. As an example, the two HTTP distributions (Figures 3b and 4b) show that we can easily configure *Snort* to operate in a high-speed network if a majority of the attacks is using HTTP. However, a small deviation in the attackers' behaviour can reduce the detection rate (cf. Figures 3a and 4a). As Figures 5 and 6 illustrates generalizing HTTP to

TCP has a better performance / detection rate trade-off, this research work recommends considering some margins to the available information

Our measurements also show (Figure 7) that careful preparation of the rules can further speed-up *Snort* significantly. However, it is not clear whether *Snort* can select the most appropriate content of a rule for Aho-Corasick [16] string matching by default. To this end, we plan to extend this research work and consider only the rules that use 'fast pattern' option.

VII. Conclusion

This research work aims to shade light on improving the performance of Network Intrusion Detection Systems (NIDS) in a high-speed network environment. To this end, we focus on the two parallelization strategies targeting two layers of protocol stack, namely Application and Transport. We developed a test bench to implement different parallelization strategies following either rule distribution or traffic distribution approaches. Our experimental results show that traffic distribution always outperforms rule distribution with respect to the performance / detection rate trade-off. We also investigated the impact of rule developers on the processing speed of *Snort*. For this purpose, we only consider the usage of 'fast pattern' option in a rule. The outcome shows that *Snort* can process these rules extremely fast but the process requires significant work at the rule development stage. We plan to study the 'fast pattern' option in further detail as the future work.

Acknowledgment

This work has been supported in part by the German Research Foundation (DFG) under grant no. DR 639/20-1.

References

[1] T. Biermann, E Cloete, and L. Venter, "A comparison of Intrusion Detection systems," *Computers & Security*, vol. 20, no. 8, pp. 676–683, Dec. 2001.

[2] S. Antonatos, K. G. Anagnostakis, E. P. Markatos, and M. Polychronakis, "Performance analysis of content matching intrusion detection systems," in *IEEE Symposium on Applications and the Internet (SAINT)*, Tokyo, Japan, Jan. 2004, pp. 208–215.

[3] F. Erlacher and F. Dressler, "On High-Speed Flow-based Intrusion Detection using Snort-compatible Signatures," *IEEE Transactions on Dependable and Secure Computing (TDSC)*, Feb. 2020, to appear.

[4] W. Bulajoul, A. James, and M. Pannu, "Network Intrusion Detection Systems in High-Speed Traffic in Computer Networks," in *10th IEEE International Conference on e-Business Engineering (ICEBE 2013)*, Coventry, United Kingdom: IEEE, Sep. 2013, pp. 168–175.

[5] H. Jiang, G. Xie, and K. Salamatian, "Load Balancing by Ruleset Partition for Parallel IDS on Multi-core Processors," in *IEEE International Conference on Computer Communications and Networks (ICCCN 2013)*, Nassau, Bahamas: IEEE, Jul. 2013.

[6] T. Limmer and F. Dressler, "Adaptive Load Balancing for Parallel IDS on Multi-Core Systems using Prioritized Flows," in *IEEE International Conference on Computer Communication Networks (ICCCN 2011)*, Maui, HI: IEEE, Jul. 2011, pp. 1–8.

[7] N. T. Huy, "A Dynamic Scalable Parallel Network-based Intrusion Detection System using Intelligent Rule Ordering," Master's Thesis, informatikk, Oslo, Norway, May 2017.

[8] D. L. Schuff, Y. R. Choe, and V. S. Pai, "Conservative vs. Optimistic Parallelization of Stateful Network Intrusion Detection," in *IEEE International Symposium on Performance Analysis of Systems and software (ISPASS 2008)*, Austin, TX, Apr. 2008, pp. 32–43.

[9] S. Kornexl, V. Paxson, H. Dreger, R. Sommer, and A. Feldmann, "Building a Time Machine for Efficient Recording and Retrieval of High-Volume Network Traffic," in *5th ACM SIGCOMM Conference on Internet Measurement (IMC 2005)*, Berkeley, CA: ACM, Oct. 2005, pp. 267–272.

[10] T. Limmer and F. Dressler, "Improving the Performance of Intrusion Detection using Dialog-based Payload Aggregation," in *30th IEEE Conference on Computer Communications (INFOCOM 2011), 14th IEEE Global Internet Symposium (GI 2011)*, Shanghai, China: IEEE, Apr. 2011, pp. 833–838.

[11] F. Erlacher and F. Dressler, "High Performance Intrusion Detection Using HTTP-based Payload Aggregation," in *42nd IEEE Conference on Local Computer Networks (LCN 2017)*, Singapore, Singapore: IEEE, Oct. 2017, pp. 418–425.

[12] M. Roesch, "Snort - Lightweight Intrusion Detection for Networks," in *13th USENIX Conference on System Administration (LISA 1999)*, Seattle, WA: USENIX Association, Nov. 1999, pp. 229–238.

[13] J. Beale and B. Caswell, *Snort 2.1 Intrusion Detection*, 2nd ed. Syngress, 2004.

[14] B. Caswell and J. Hewlett, "Snort Users Manual," The Snort Project, Manual, May 2004. [Online]. Available: http://www.snort.org/docs/snort_manual.pdf.

[15] J. B. D. Cabrera, J. Gosar, W. Lee, and R. K. Mehra, "On the statistical distribution of processing times in network intrusion detection," in *2004 43rd IEEE Conference on Decision and Control (CDC 2004)*, Nassau, Bahamas: IEEE, Dec. 2004, pp. 75–80.

[16] A. V. Aho and M. J. Corasick, "Efficient String Matching: An Aid to Bibliographic Search," *Communications of the ACM*, vol. 18, no. 6, pp. 333–340, Jun. 1975.

[17] R. S. Boyer and J. S. Moore, "A fast string searching algorithm," *Communications of the ACM*, vol. 20, no. 10, pp. 762–772, Oct. 1977.

[18] J. Armstrong, K. Vasudevan, and C. Rene Robin, "Snort3 rules exact string matching based on FPGA," *SPAST Abstracts*, vol. 1, no. 1, Oct. 2021.

[19] S. A. Changazi, I. Shafi, K. Saleh, M. H. Islam, S. M. Hussainn, and A. Ali, "Performance Enhancement of Snort IDS through Kernel Modification," in *International Conference on Information and Communication Technologies (ICICT 2019)*, Karachi, Pakistan: IEEE, Nov. 2019, pp. 155 –161.

[20] U. Trivedi, "An Optimized Aho-Corasick Multi-Pattern Matching Algorithm for Fast Pattern Matching," in *India Council International Conference (INDICON)*, Delhi, India: IEEE, Dec. 2020.

[21] L. Shuai and S. Li, "Performance optimization of Snort based on DPDK and Hyperscan," *Procedia Computer Science*, pp. 837–843, 2021.

[22] D. Zhang and S. Wang, "Optimization of traditional Snort intrusion detection system," *IOP Conference Series: Materials Science and Engineering*, vol. 569, no. 4, Jul. 2019.

[23] X. Wang, Y. Hong, H. Chang, K. Park, G. Langdale, J. Hu, and H. Zhu, "Hyperscan: A Fast Multi-pattern Regex Matcher for Modern CPUs," in *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 2019)*, Boston, MA: USENIX Association, Feb. 2019, pp. 631–648.

[24] E. W. Fulp and R. Farley, "A Function-Parallel Architecture for High-Speed Firewalls," in *IEEE International Conference on Communications (ICC 2006)*, Istanbul, Turkey, Jun. 2006, pp. 2213–2218.

[25] H. Haugerud, N. T. Huy, N. Aitsaadi, and A. Yazidi, "A dynamic and scalable parallel Network Intrusion Detection System using intelligent rule ordering and Network Function Virtualization," *Elsevier Future Generation Computer Systems*, vol. 124, pp. 254–267, Nov. 2021.

[26] H. Doroud, G. Aceto, W. De Donato, E. Alizadeh Jarchlo, A. M. Lopez, C. D. Guerrero, and A. Pescape, "Speeding-Up DPI Traffic Classification with Chaining," in *IEEE Global Communications Conference (GLOBECOM 2018)*, Abu Dhabi, United Arab Emirates: IEEE, Dec. 2018.

[27] F. Erlacher and F. Dressler, "How to Test an IDS? GENESIDS: An Automated System for Generating Attack Traffic," in *ACM SIGCOMM 2018, Workshop on Traffic Measurements for Cybersecurity (WTMC 2018)*, Budapest, Hungary: ACM, Aug. 2018, pp. 46–51.