2018 IEEE 36th International Conference on Computer Design

General IDS Acceleration for High-Speed Networks

Jan Kučera, Lukáš Kekely
CESNET, a. l. e.
Prague, Czech Republic
jan.kucera@cesnet.cz
kekely@cesnet.cz

Adam Piecek
Faculty of Information Technology
Brno University of Technology
Brno, Czech Republic
xpiece00@stud.fit.vutbr.cz

Jan Kořenek
IT4Innovations Centre of Excellence
Faculty of Information Technology
Brno University of Technology
Brno, Czech Republic
korenek@fit.vutbr.cz

***Abstract*—Network Intrusion Detection Systems have gained popularity as one of the key technologies to secure communication infrastructures. However, their high computational complexity poses performance challenges for practical deployment in modern high-speed networks. To achieve the highest quality of detection, IDS should process as much relevant data as it can without becoming the bottleneck of a network connection. At the same time, IDS implementation should be flexible enough to accommodate detection methods of ever emerging new security threats.**

This paper aims at an acceleration of IDS by means of informed packet discarding, effectively focusing the available resources of overloaded IDS to the most relevant parts of analyzed traffic. Unlike previous works, the proposed scheme does not move the IDS nor any specific portion of it into the hardware accelerator. Rather it uses smart software based or hardware accelerated offload (bypass) of the traffic parts that are not likely to represent a security threat. The flexible nature of software-based IDS is therefore fully maintained, while the quality of threat detection remains sufficiently high even when processing high-speed traffic. We show that controlled (informed) discarding of well-defined portions of input traffic yields better detection rates, compared to the default uncontrolled (blind) buffer overflow discarding in high throughput scenarios. Our results show that it is entirely possible to run an IDS on a high-speed network link using single CPU with an FPGA accelerated packet pre-filtering.

I. Introduction

Intrusion Detection Systems significantly contribute to network security by providing a deeper insight into transferred packets and their payloads. These systems often use some form of deep packet inspection, such as pattern matching or other methods, to detect characteristic signatures of malicious activity present in the network data. A common property of these inspection methods is their overwhelming computational complexity, leading to challenges in meeting the performance requirements of modern high-speed networks. Running a practical pattern matching algorithm at 100 Gbps or even tens of Gbps is an unreachable goal for current CPUs.

On the other hand, the ever-changing nature of security landscape requires the IDS to be able to react quite rapidly to newly emerging threats and attack vectors. The flexibility of software implementation is therefore highly desirable and the use of considerably less flexible hardware processing offload that accelerates only a specific IDS application directly is therefore not so feasible.

This research has been supported by the project Reg. No. CZ.02.1.01/0.0/0.0/16_013/0001797 by the MEYS of the Czech Republic; the IT4Innovations excellence in science project IT4I XS – LQ1602; and by the Ministry of the Interior of the Czech Republic project VI20172020064.

To aid the performance of a general software-based IDS without hindering its flexibility, we propose and explore a different approach to IDS acceleration. The key idea of our concept is not to directly accelerate the speed of data processing in the IDS, but rather to intelligently reduce the amount of input traffic that the IDS must process. Based on a few basic characteristics some packets are deemed interesting and selected for processing, while others are discarded. Also, this selection of packets for processing/discarding is done in such a controlled way so that negative impacts on IDS detection abilities are minimized. We expect and experimentally prove that only a considerably small percentage of all threats on the network is overlooked this way, while the IDS can operate at much higher speeds as originally possible.

We present an IDS acceleration based on these assumptions:

1) The packet rate performance of software-based IDS is limited and insufficient. The IDS is not fast enough to sufficiently process all of the packets that are transmitted in a monitored high-speed network, such as 40 Gbps or 100 Gbps Ethernet line.
2) The default packet discarding mechanism is a blind input buffer overflow that behaves in an effectively random manner. In an overloaded IDS, incoming packets are discarded right after they arrive without any chance of further examination. As a result, a number of threats are overlooked and remain unreported.
3) The most relevant information regarding security is present in several packets at the beginning of each network connection (flow). These packets should be, therefore, preferred for processing over others, when IDS becomes overloaded.
4) Basic packet examination can be performed and the obtained information subsequently used to selectively discard some of the following packets. Furthermore, such packet discarding mechanism exists, that lets the IDS yield a better quality of detection than the default (blind) discarding. This holds, even though the raw packet rate of the IDS itself stays unchanged.
5) Majority of the network traffic is carried by a quite small number of relatively large flows. So, selection of only a few flows for discarding (bypass) can significantly reduce input packet rate of IDS.

2576-6996/18/$31.00 ©2018 IEEE
DOI 10.1109/ICCD.2018.00062

In the following text, we basically assume that points (1) and (2) hold true based on our previous experiences with IDS deployment (more in section III). The validity of point (5) has been already shown in several other papers, so we simply check if this feature is also present in our network data. The main focus of this paper is placed on thoroughly exploring and proving points (3) and (4).

The main contribution of this paper is three-fold:

- Design of an input acceleration concept (heuristic) that considerably reduces the amount of data sent to any general IDS in a controlled and beneficial manner. The concept idea enables that both software and hardware-based implementations are possible.
- Examination of real network traffic traces to show that the overall quality of threat detection remains sufficiently high even when only short flows and several initial packets of large flows are observed. In other words, proving that only a minute fraction of network threats is present in the latter parts of the connection contents.
- Implementation and experimental evaluation of the proposed informed discarding concept to demonstrate its effectivity under real network deployment conditions.

II. Related Work

There are several commonly used software implementations of IDS, which we list here with a brief description. Snort [1] is an open source software network intrusion detection and prevention system. It relies heavily on regular expression matching. Similarly to Snort, L7-filter [2] also operates with regular expressions. It is a Linux based packet classification software aiming primarily at application layer (L7) processing. A software library for application layer traffic processing called nDPI [3] can serve as an example showing that regular expression matching alone is not enough. It should be only one component of a more complex set to form a robust IDS. Bro [4] is a flexible framework that allows specification of custom detection rules using its own scripting language. This feature makes it very powerful, but also rather complex. Suricata [5] is functionally quite similar to Snort, but supports multithreaded processing and is, overall, built to achieve higher performance.

Apart from software implementations, there is a large number of papers proposing partial or full IDS functionality offload to a hardware accelerator, for example [6], [7], [8]. The most common way of doing that is by converting regular expressions to an FPGA firmware structure, thus offloading the time-consuming pattern matching from the CPU. Such approach poses a disadvantage of lowered flexibility. Most proposed methods require recompiling of the FPGA firmware when the regular expression set changes. That may take hours to complete and meeting FPGA timing constraints is never guaranteed. Also, advanced techniques like TCP stream re-assembling are often missing in FPGA-based IDS accelerators, leaving open back doors for covert attacks. Finally, for IDS that use more complex threat detection methods than just pattern matching, such acceleration is of little use.

An example of a more flexible acceleration is provided by The Shunt system [9]. It is a hardware accelerator that cab divert a suspicious (interesting) traffic to the software for further analysis. To this end, it somehow resembles our work. Of course, our work uses a more powerful accelerator, resulting in throughput of tens of Gbps, while the Shunt was demonstrated at only 1 Gbps links. The main differentiation of our work is that it is much more complete. While the Shunt paper describes almost exclusively the hardware architecture and its implementation, we primarily aim to provide analysis of real network traces together with extensive experimental results of achieved IDS acceleration. Moreover, we provide results showing the benefits of a pure software implementation of our proposed controlled packet discarding method.

Another more sophisticated hardware acceleration concept is our previous work – Software Defined Monitoring (SDM) system [10]. It is a hardware accelerator aimed primarily at flow-based network monitoring. It supports offloading of NetFlow statistics computation of uninteresting and large flows into the hardware, while sending packets of short, interesting and unknown flows to the CPU for a more detailed analysis. SDM also includes a software controller with easy to use API, which accepts offload requests from the monitoring (security) applications and commands the hardware accelerator accordingly. It has been shown to effectively accelerate network flow measurement up to application layer processing, but no clear benefits have been demonstrated when used to offload traffic from more computationally complex IDS.

A research published in [11] proposes a Time Machine concept. This concept exploits the heavy-tailed nature of network traffic and also assumes that the most relevant information is present at the beginning of each network flow. But, their approach aims at a packet capture system which enables storage of suspicious traffic for later offline (i.e. retrospective) forensics. Also, the Time Machine system is only controlled by an IDS and do not in any way accelerates its operation. That is distinctly different from our proposed concept. We deal with an online analysis of suspicious traffic and employ an accelerated pre-filter to directly aid the IDS performance by reducing the amount of data on its input.

An attempt at creating a high-speed IDS was performed at Berkeley Lab [12] using Bro. In this approach, the traffic is distributed to multiple IDS servers by a pair of switches. To achieve high throughput, five servers with 10 Gbps input lines each are running Bro in parallel. If some Bro instance detects a bulk transfer, the switches are set to discard the subsequent packets of that flow. Therefore, the amount of input traffic to Bro servers is reduced. Our work is similar to [12] in that it uses software IDS and a controlled packet discarding. However, our work aims at achieving similar IDS performance in a single box deployment. Furthermore, our paper also focuses on a detailed analysis of the controlled packet discarding influence on the achieved quality of detection, which is completely missing in [12]. Finally, our discarding algorithm is much finer and IDS agnostic as it does not rely solely on the information from the IDS.

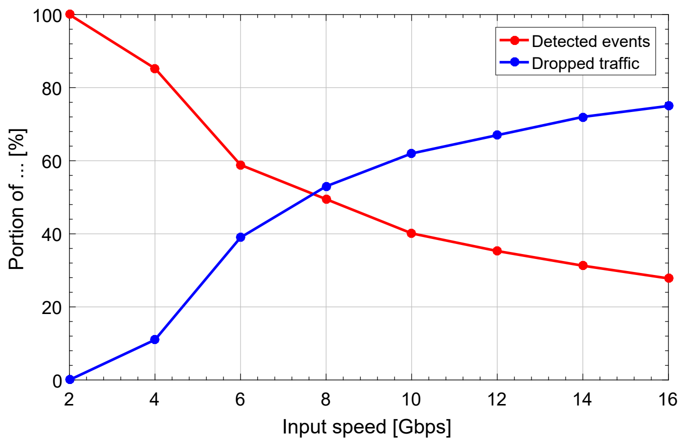

Fig. 1. Percentage of dropped packets and detected events at different speeds.

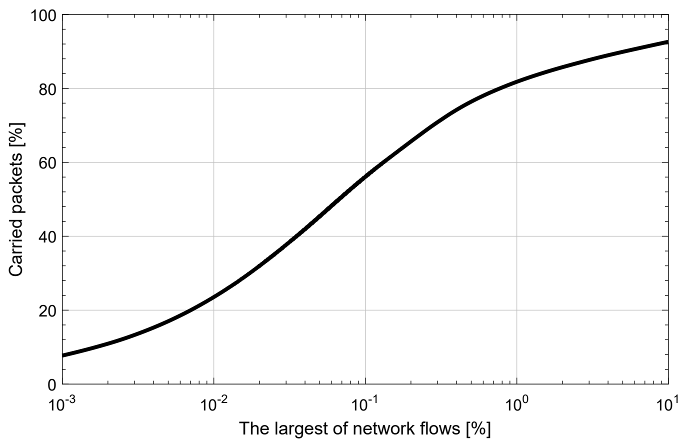

Fig. 2. Percentage of packets carried by the largest flows on the network.

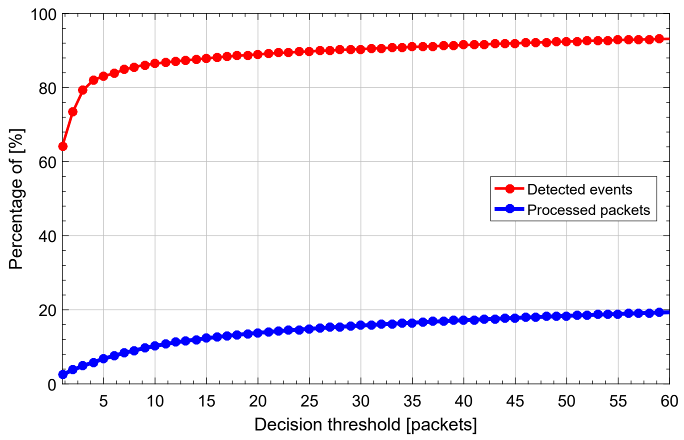

Fig. 3. Processed packets and detected events in them for different N.

III. Proof of Concept

In this section, we analytically support our claim that *controlled* packet discarding results in sufficiently high IDS detection quality, while IDS can handle traffic at much higher speeds. For this evaluation, we use unsampled packet data from one of the lines in our nation-wide network. The captured PCAP file contains 285 833 947 packets of 8 642 744 flows in over 200 GB of data. It was captured at around 2 Gbps link utilization over a duration of 826 seconds. The traffic is replayed on its original capture speed. To perform measurements with higher bandwidths, we still replay the PCAP at the original speed (to maintain flow timing characteristics), but replicate each packet several times. To avoid changing flow data, we deterministically modify each packet's IP address when replicating, which effectively results in new flows being created. As an IDS, we choose Suricata [5] because of its affinity to high-performance multi-threaded implementation. We use 13 642 detection rules from the public EmergingThreats database [13].

To support our claims of insufficient IDS performance or detection quality for high-speed deployment, we provide some basic results in Fig. 1. An out-of-the-box version of Suricata is used. A packet drop rate on its input (blue) and detection accuracy (red) are measured for different traffic speeds. We can see that a packet loss of around 10 % is present even at 4 Gbps. The loss is consistently rising with input speed and at 10 Gbps it already reaches more than 60 %. For the overall detection quality of tested IDS at various input speeds and related drop rates, the 100 % baseline is given by offline (non-discarding) analysis of the acquired network data. All of the expected events are reliably detected only at the speed of 2 Gbps. The presence and increase of *uncontrolled* packet discarding for higher speeds causes the percentage of successfully detected events to drops rapidly. Even at 6 Gbps (40 % drop) only around 60 % of all events are detected and at the speeds of 10 Gbps and more (above 60 % drop) only fewer than 40 % of events is detected. Therefore, there is definitely a reason to utilize some kind of IDS acceleration.

Findings already presented in various papers like [10] suggest that high-speed network traffic has a heavy-tailed character of flow (communication) size distribution. The heavy-tailed character of flow size distribution derived from the captured PCAP file is shown in Fig. 2. The graph shows the percentage of all packets carried by the specific portion of the largest (heaviest) flows from the file. It can be seen that even 0.1 % of the largest flows carry as many as nearly 60 % of all packets and 1 % carry more than 80 %. The observed heavy-tailed character of flow sizes has a potentially positive consequence for an achievable efficiency of the proposed controlled discarding concept. Even if only a small percentage of all flows is selected for controlled discarding, processing of majority of the packets by IDS is still avoided. In other words, this enables to focus the IDS's effort primarily to short flows and several initial packets of larger flows with potential for considerable benefits in achievable performance.

Fig. 3 shows (in blue) the percentage of packets that have to be processed by IDS when only the first N packets of each network flow are analyzed and the rest is discarded (flows shorter than N are analyzed entirely). The discarding decision threshold N is shown on the horizontal axis, while the percentage of packets is drawn in blue, one dot for each considered value of N. We can see that the first 50 packets of all flows carry less than 20 % of all packets, therefore the remaining 80 % of packets are found in later packets of flows larger than 50 packets. These results only confirm the expected implications of the heavy-tailed character of flow size

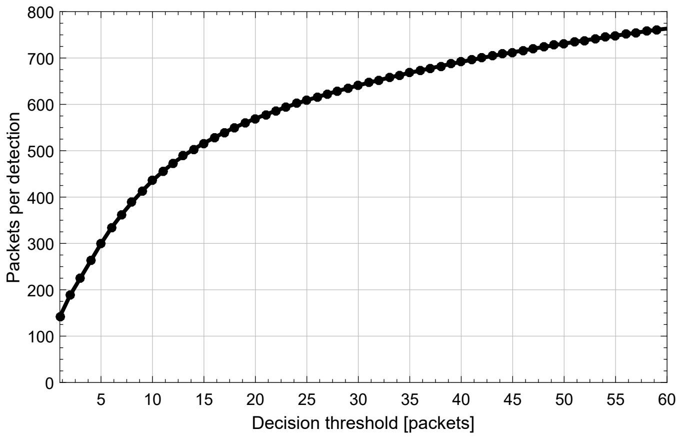

Fig. 4. A number of processed packets per detected event for different N.

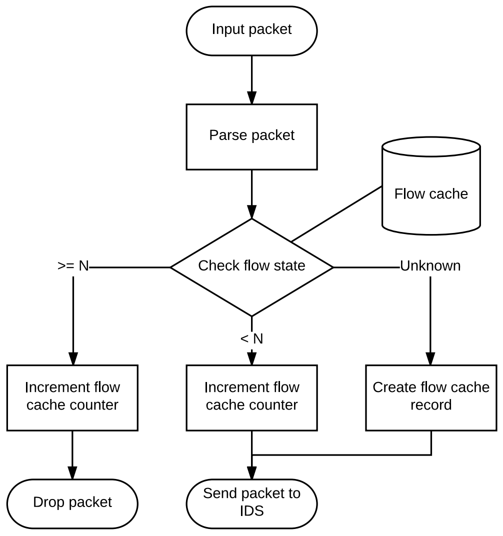

Fig. 5. Flowchart of packet processing in the proposed IDS acceleration.

distribution stated in the previous paragraph.

Fig. 3 also shows (in red) the number of events detected by Suricata when the processing of the input file is reduced to N initial packets of each flow. The results are shown in relative form (as a percentage), where the base value (100 %) was obtained similarly as before – by running Suricata offline analysis on the original 200 GB file (i. e. $N = \infty$). We can see that more than 80 % of all security threats are detected even when only the initial 5 packets of all flows are considered and more than 90 % when $N > 30$. Furthermore, with forwarding of more packets into the IDS (a further increase of N) the detection rate increases only rather slowly. Therefore, we argue that if the IDS system is able to process only part of the network traffic due to insufficient performance, it should focus primarily on the processing of the first several packets of each flow and the rest should be preferred for discarding.

A different view of the same data is provided in Fig. 4. The number of processed packets is divided by the number of detections for each analyzed value of N. We can see that by lowering the threshold N, considerably fewer and fewer packets must be, on average, analyzed to detect a security threat. Just for comparison, when the whole PCAP file is processed by Suricata (i. e. $N = \infty$) the value of packets per detection ratio rises to over 2000. These findings further prove the conclusion of the previous paragraph, that the IDS can be tuned to higher detection efficiency under heavy loads by the controlled preference of initial packets of flows for analysis.

IV. System Design

As already mentioned in the Introduction, the design of our IDS acceleration concept is mainly motivated by the need to reduce input packet rate to the IDS, while retaining as much relevant information as possible to maintain high detection accuracy. Since we assume that the most relevant information regarding security is present in several packets at the beginning of a network connection, **we design our concept to drop all packets that follow the first N packets of each network flow.** The value of threshold N can be arbitrarily altered depending on momentary network situation. Of course, because of this design choice, the system does not detect attacks which arise after the first N packets of the flow, but as the analysis results from the previous section have shown, the number of such attacks is very small (less than 10 % even for $N = 30$). Furthermore, the value of N should be set to ∞ when IDS is processing input data at sufficient rate and gradually lowered only to prevent blind buffer overflow as input traffic volume starts to overwhelm the IDS.

To perform the described decision, a system implementing the proposed concept must maintain very basic flow statistics. This requires two main additional modules: packet header parser and flow cache. Packet header parser is required to obtain packet header fields that uniquely identify a network flow. We use the standard five-tuple of IP addresses, port numbers, and L4 protocol number to identify flows. Network flow cache is necessary to store and update records of actual flow lengths. Every incoming packet either creates a new flow record or increments a size counter in an existing one.

A flowchart of the proposed IDS input system operation is shown in Fig. 5. Every input packet is firstly parsed and flow identification fields are extracted. Then the associated flow record is searched in the flow cache. A new flow record is created for unknown (not found) flows, packet counter is incremented for known flows. The packet is dropped if the counter for appropriate flow exceeds configured threshold N otherwise, the packet is forwarded to IDS for normal processing. Furthermore, there is an independent housekeeping process (not shown in the flowchart) that removes old entries from the flow cache. It operates with configurable inactive and active timeout periods of flow records.

This scheme of operation is general and independent on the particular features of IDS used. It is therefore applicable to any software IDS for which it is possible to alter the input path. Since most software IDS employ extensible modular design, their input modules or plugins are often a convenient place for the implementation of our acceleration method.

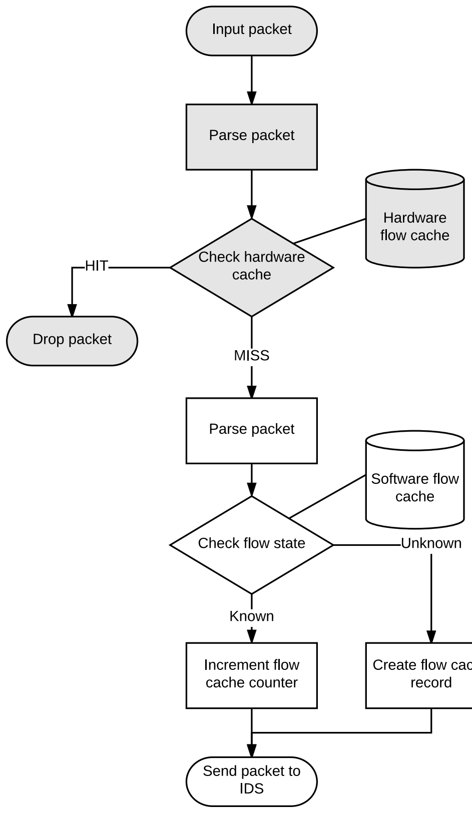

Fig. 6. Packet processing in the proposed IDS acceleration with HW offload.

A. Hardware Accelerated Offload

Utilization of the proposed input system inevitably requires some additional processing from the CPU, adding to its total load. To further explore the performance limits of our approach and achieve even better results, we employ hardware accelerator that drops unwanted packets before they reach the CPU. To enable this functionality, both the packet parser and the network flow cache must be present in the hardware accelerator, so that the decision whether to drop or pass packets can be offloaded and made directly by the accelerator.

The updated scheme of system operation with the utilization of hardware acceleration is shown in Fig. 6. The hardware accelerator (grey blocks) parses packets, searches for appropriate flow records in its flow cache and drops all matching packets. Then the software path (white blocks) only passes packets to IDS and maintains flow records in its software flow cache. The housekeeping process (not shown) replicates (offloads) the heavy flow records that exceed configured threshold N from the software flow cache to the hardware one and also removes outdated flow records from the software and hardware cache according to the configured timeouts.

While it would be certainly possible to *move* the packet parser and the flow cache to the hardware entirely, we rather *replicate* them. The main reason for this decision is to maintain the flexibility of utilization. By maintaining a software flow cache, alternative packet discarding mechanisms can be easily used and fine-tuned. By choosing which flows will be offloaded from the software flow cache to the hardware one, the system can, for example, use different discarding thresholds N for individual traffic types or IP subnets. Also, the packet parser is typically present in IDS anyway, so that it can be shared and presents no additional computation load for CPU. Finally, due to the fact that most of the traffic is discarded in the hardware, the performance penalty of maintaining a software flow cache is significantly reduced.

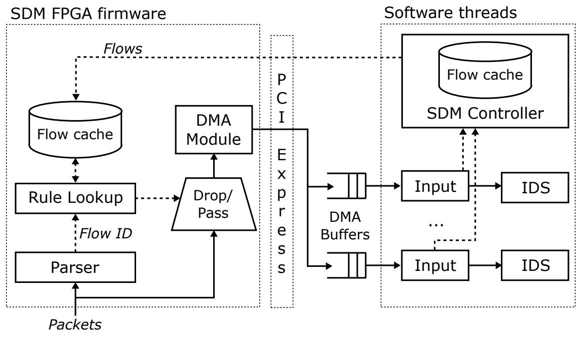

Fig. 7. Our hardware accelerated IDS input concept utilizing subset of SDM.

To implement the proposed hardware accelerated version of our concept, we utilize the Software Defined Monitoring (SDM) system [10]. Although SDM is primarily designed to accelerate flow-based network monitoring, a subset of its functionality can be easily utilized also for our IDS acceleration concept. This utilization is outlined in Fig. 7. SDM uses FPGA accelerator cards with 10, 40 or 100 Gbps Ethernet interfaces connected to a host server via PCI Express bus. The SDM FPGA firmware itself (on the left) already implements capture of input packets from Ethernet links, hardware packet parser, a cuckoo hashing based form of flow cache with packet filtering capabilities, as well as fast DMA transfers with the support for flow-based traffic distribution among CPU cores, enabling effortless multi-threaded IDS operation. The hardware flow cache is realized using on-card QDR memories and has a total capacity of over 250 000 rules (flows). Because network traffic has a heavy-tailed character of flow sizes and we want to offload only the heaviest of the flows, the cache capacity should not be a very limiting factor. However, if it proves to be an obstacle, on-chip URAMs can be used to further enlarge the cache in newer UltraScale+ FPGAs. The SDM software (on the right) includes mainly the SDM Controller, which implements software flow cache and presents a simple C language API for easy integration into existing IDS. As depicted, only input modules/plugins of accelerated IDS are communicating with software flow cache of SDM Controller maintaining its records and requesting offload of selected flows into hardware cache. The offloading process is optimized for latency and throughput, the whole hardware cache can be filled with new records in less than a second. The capacity of the software flow cache is considerably larger compared to the hardware one and it can store tens of millions of flow records.

V. Experimental Results

Evaluation of the proposed acceleration concept presented in this paper uses Suricata IDS [5] running on a commodity

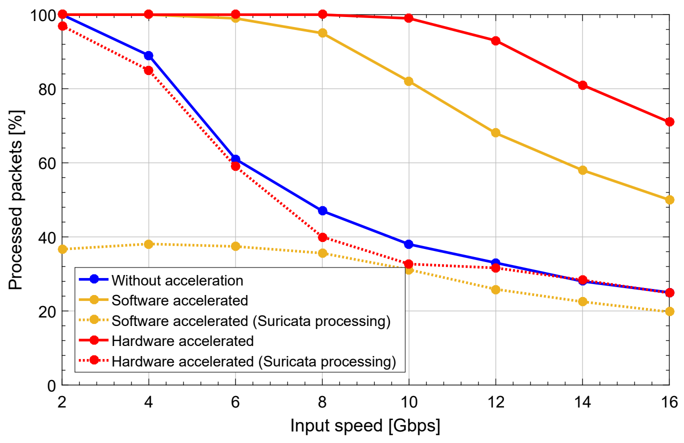

Fig. 8. Percentage of incoming packets processed for different input speeds.

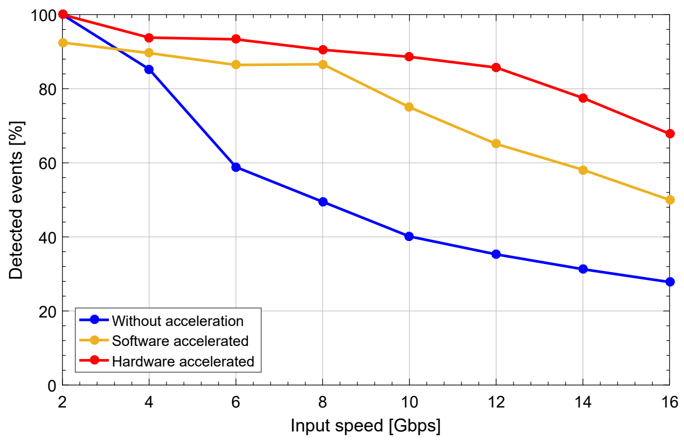

Fig. 9. Percentage of security events detected by Suricata at different speeds.

SuperMicro server with 8 core Intel Xeon E5-2670 CPU operating at 2.6 GHz and with 64 GB of RAM. The same 200 GB PCAP file with real network traffic that we used for initial analysis is also used for these experiments.

Three deployment scenarios are evaluated and compared:

- *Without acceleration*, when Suricata is executed as it is, without any modifications whatsoever. This creates a baseline to evaluate acceleration benefits against.
- *Software accelerated*, where Suricata input software plugins discard packets according to the scheme from Fig. 5, implementing their own software cache.
- *Hardware accelerated* scenario utilizes SDM to discard packets according to the scheme from Fig. 6. Input plugins of Suricata are altered to communicate with SDM Controller and instruct it to offload selected heavy flows (flows that exceed configured N) into the hardware cache.

In both software and hardware accelerated versions, the value of offload threshold N is hand-picked and optimized to obtain the best results. We assume that in a real deployment, N can be automatically adjusted on the fly by the SDM system to adapt to changing network traffic characteristics as already shown in [10]. Basic idea is to lower the value of N (i.e. offloading more) whenever an input buffer overflow (blind discard) is detected and raise it again (i.e. offloading less) when the input traffic rate drops. The threshold adaptation must also consider the current load of the hardware flow cache, which has a limited size. For the best offload ratio, it is advantageous to keep the flow cache nearly full of active records. On the other hand, to maximize the quality of detection we want to forward as many packets to the IDS as it can handle.

A. Complete Ruleset Detection

In the first experiment, we test Suricata with all of the 13 642 rules from the EmergingThreats database [13] (the same as in section III). This configuration enables the most detailed and precise threat detection but it is also extremely demanding on consumed CPU performance per packet.

Fig. 8 shows packet drop rates for each of the tested scenarios. For non-accelerated version (blue), Suricata starts uncontrolled packet drops at input rate between 2 and 4 Gbps. For software accelerated version (orange), the software input plugin actively performs controlled packet discarding reducing the load of the tested IDS (orange dotted line), which significantly helps in reducing uncontrolled packet drops at the input (shown as the decline of the solid orange line from the 100 % edge). The limited CPU performance forces the system to start dropping packets uncontrollably at input rate between 6 and 8 Gbps. With the hardware acceleration, most of the packets never even arrive at the CPU. This is shown as the dotted red line with circles in the graph – CPU (software IDS) needs to process only around 30 % of all packets at high speeds (maximum acceleration rate). At the input speed of around 10 Gbps, the rate of packets passed by the accelerator to the CPU becomes still too high to process, and uncontrollable discarding occurs. Further reducing the rate of packets that are sent to the CPU (increasing the controlled packet discarding) would require lowering of the threshold N to extremely low values. That, however, was not possible, because the hardware flow cache size of SDM implementation is limited to around 250 000 items (flows). Furthermore, reduction of N below a certain point itself can also considerably lower detection quality of IDS (see the left side of Fig. 3).

More important numbers are shown in Fig. 9, which plots the percentage of detected events, related to the 100 % baseline given by offline (non-discarding) analysis of the used PCAP file. This ultimately represents the detection quality of tested IDS for given input rate. Only for the speed of 2 Gbps, when no packets are dropped, all events present in the input data are detected by non-accelerated version (blue). The percentage of all events detected in all scenarios lowers rapidly as the uncontrolled packet drops shown in the previous graph rises. However, the increase in *controlled* discarding, present in both accelerated versions (orange and red), causes only a moderate decline of detection quality.

With the controlled discarding, the system maintains good detection quality for much higher input packet rate. In hardware accelerated scenario (red), the detection rate drops under 80 % only at speeds higher than 12 Gbps, while without acceleration (blue) this drop occurs right after 4 Gbps mark (3× sooner). From a different perspective, the detection quality of Suricata IDS is enhanced up to 2 or 3× by the proposed hardware acceleration at higher input speeds.

Furthermore, it is worth noting that for 2 Gbps input rate, the software accelerated version still performed some controlled discarding, which reduced detection quality. In hardware accelerated scenario, we employed on the fly accommodation of decision threshold N to higher values for lower speeds based on actual CPU load (note that red dotted line follows blue line in Fig. 8). That is why the hardware accelerated version can reach 100 % detection rate when the system is not overloaded. To further show the benefit of the threshold accommodation, the software accelerated version was left to discard packets unnecessarily even at lower data rates (steady orange dotted line in Fig. 8). This is exactly why the detection rate is lower (only 90 %) even at low input rates. In a real deployment, this configuration can easily be avoided. We also want to point out the fact that in the accelerated versions, only a small portion of packets is actually sent to Suricata for processing (dotted lines in Fig. 8). The detection quality remains still considerably high – around 90 % – due to the increased IDS efficiency, as already predicted by Fig. 4 in section III.

Now we return back to our assumption number 4) from the Introduction and use Fig. 10 to supports its claim. Each measured value from the previous experiment is drawn as one point in the (processed packets × detected events) space. For unaccelerated version (blue), we can see a strong linear correlation (blue dotted line), which supports our initial assumption that uncontrolled discarding is effectively random. This is also our baseline since it represents the default IDS deployment use case. With the hardware accelerated (controlled discarding) version (red), all our measured points are above the baseline, which means that for a given amount of packets processed by the IDS, more events were detected. Note that darker red dots are from measurements at highest speeds, which are negatively influenced by blind discarding. Theoretical data from our analysis in section III are shown as the dotted red line. The reason for the measured data being slightly worse is that there is a small latency in installing packet discarding rules to the SDM hardware. Therefore, the accelerator sometimes lets more than N packets in, which causes the difference between the model and our implementation.

B. Malware Detection

In real network deployments of IDS, it is common to select only a specific subset of available detection rules to focus only on the most critical threats or relevant threats [14], [15]. In the wake of recent massive outbreaks of malware infections, especially ransomware like WannaCry or Petya [16], we focused on malware detection rules here. This way, we select a total of 967 malware oriented rules from the original 13 642. As a result, virtually all reported events are only of ET MALWARE type while other (previously more prevalent) types like ET SCAN, ET DOS or ET POLICY are not detected. With smaller ruleset used in Suricata, a reduction in CPU performance demands per packet is expected to lead to higher achievable speeds.

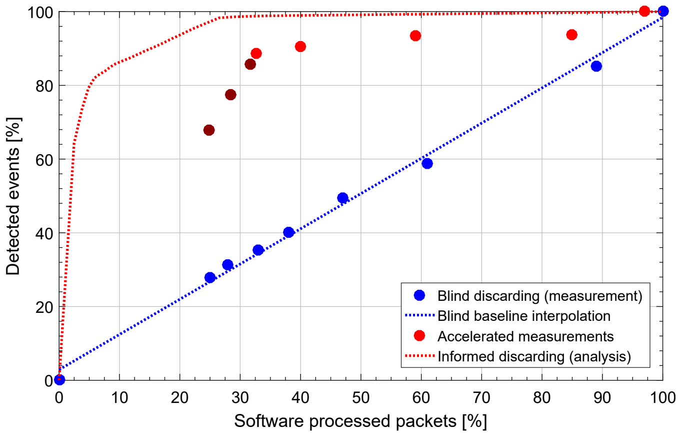

Fig. 10. The relation between processed packets and detected events.

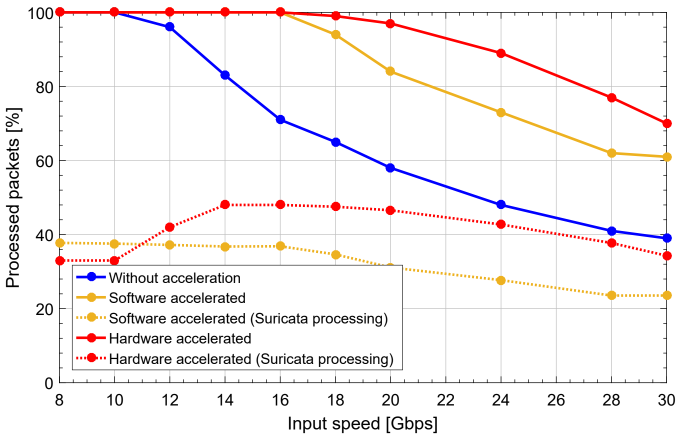

Fig. 11. Percentage of packets processed for different speeds (small ruleset).

Fig. 11 shows packet drop rates for each of the tested scenarios with the smaller ruleset. For non-accelerated version (blue), Suricata starts uncontrolled packet drops at input rate between 10 and 12 Gbps. For accelerated versions the speeds are higher, the software version starts dropping packets uncontrollably at input rate between 16 and 18 Gbps and hardware version starts only at around 20 Gbps. These numbers show that reduced ruleset enables Suricata to reach speeds about 10 Gbps higher as in the previous experiment, while other basic characteristics of these graphs remain the same.

More important numbers are shown in Fig. 12, which again plots the percentage of detected events, related to the 100 % baseline obtained from the offline analysis. For the speeds of up to 10 Gbps, when no packets are dropped, all expected events are detected by non-accelerated version (blue). The percentage of all events detected in all scenarios lowers even more rapidly as before with the rise of uncontrolled packet drops shown in the previous graph. And again, even the high rate of *controlled* discarding, causes only a rather slow drop in detection quality – detection quality drops under 80 % 2× later. At the highest input rates, detection capabilities are enhanced up to 2 or 3× compared to non-accelerated version.

Interesting information can be seen from Fig. 13. Again each measured value from this experiment is drawn as a point in the (processed packets × detected events) space. For

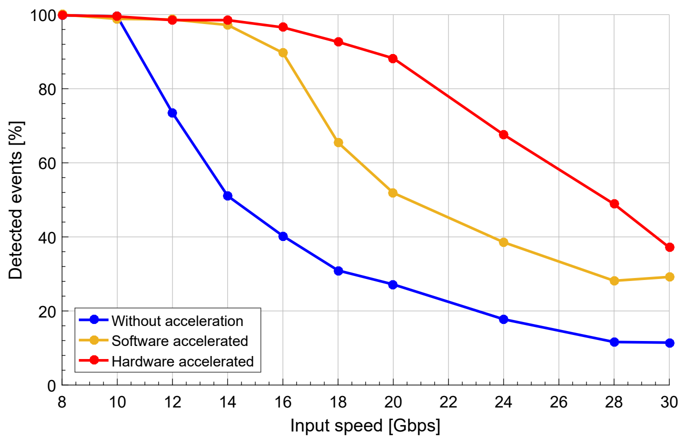

Fig. 12. Percentage of security events detected by Suricata (small ruleset).

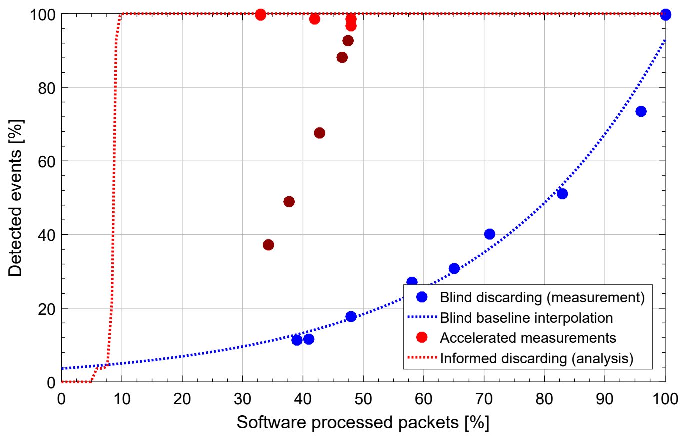

Fig. 13. The relation of processed packets and detected events (small ruleset).

unaccelerated version (blue), a strong exponential baseline correlation (blue dotted line) is now present instead of linear. This further favors the use of acceleration for IDS, as blind discarding has an even worse impact on malware detection rate as on general detections. The exponential correlation here is due to the common need to process multiple subsequent packets of a flow in order to detect a single malware threat. Loss of even one of these packets often leads to failed detection. Theoretical data from an offline PCAP analysis are shown as the dotted red line. Here, the detection rate is even more dependent on the first packets from flows as in the first experiment – all detections occurred between the 4th and 10th packets of all flows. Hardware accelerated (controlled discarding) version (red) again shows that all our measured points are well above the baseline, even the darker red dots representing measurements at highest speeds, which are significantly negatively influenced by blind discarding.

VI. Conclusion

We have designed and tested a new concept of Intrusion Detection System acceleration based on a controlled (informed) reduction of incoming traffic while retaining sufficiently good overall threat detection capabilities. Based on the analysis results showing that the first packets of network flows are the most important for security detections, in our concept, we propose to drop all packets that follow the first N packets of each network flow, where the value of N is optimized on-the-fly based on current IDS load. In other words, if an IDS is overloaded and have to skip processing of some packets, we propose a system for informed selection of which packets to skip (ends of heavy flows) instead of reliance on random buffer overflow mechanism. The proposed concept can be implemented as a pure software system, but can also take advantage of hardware acceleration to achieve even higher IDS speed up. Furthermore, the concept is general enough to be deployable with any software based IDS.

In this paper, we use Suricata IDS for experimental testing on captured data from a real high-speed network. We have tested two basic configurations of Suricata – full ruleset (13 642 rules) and smaller ruleset optimized for malware detections (967 rules). Achieved experimental results conclusively show that our proposed form of informed discarding is considerably better compared to default blind buffer overflow. Utilizing our acceleration concept, we are able to achieve **processing of 2 or 3$\times$ higher input link speeds** compared to non-accelerated IDS, while high detection quality is maintained. Or from a different point of view, our acceleration enables the IDS to **detect up to 3$\times$ more events** at given high-load compared to deployment without acceleration.

References

[1] M. Roesch et al., “Snort – Network Intrusion Detection & Prevention System,” Aug 2018. [Online]. Available: http://www.snort.org/
[2] ClearFoundation, “l7-filter,” 2018. [Online]. Available: http://l7-filter.clearos.com/
[3] ntop, “nDPI,” Aug 2018. [Online]. Available: http://www.ntop.org/products/ndpi/
[4] V. Paxson et al., “The Bro Network Security Monitor,” Aug 2018. [Online]. Available: http://www.bro.org
[5] M. Jonkman et al., “Suricata,” Aug 2018. [Online]. Available: http://suricata-ids.org
[6] H. Song, T. Sproull, M. Attig, and J. Lockwood, “Snort offloader: a reconfigurable hardware NIDS filter,” in *Field Programmable Logic and Applications (FPL)*. IEEE, 2005.
[7] A. Mitra, W. Najjar, and L. Bhuyan, “Compiling PCRE to FPGA for Accelerating SNORT IDS,” in *Architecture for Networking and Communications Systems (ANCS)*. ACM, 2007.
[8] V. Košař, M. Žádník, and J. Kořenek, “NFA reduction for regular expressions matching using FPGA,” in *Field-Programmable Technology (FPT)*. IEEE, 2013.
[9] N. Weaver, V. Paxson, and J. M. Gonzalez, “The Shunt: An FPGA-based Accelerator for Network Intrusion Prevention,” in *Field Programmable Gate Arrays (FPGA)*. ACM, 2007.
[10] L. Kekely, J. Kučera, V. Puš, J. Kořenek, and A. V. Vasilakos, “Software Defined Monitoring of Application Protocols,” *IEEE Transactions on Computers*, vol. 65, no. 2, 2016, ISSN: 0018-9340.
[11] G. Maier, R. Sommer, H. Dreger, A. Feldmann, V. Paxson, and F. Schneider, “Enriching Network Security Analysis with Time Travel,” in *Special Interest Group on Data Communications (SIGCOMM)*. ACM, 2008.
[12] V. Stoffer et al., “100G Intrusion Detection,” Aug 2018. [Online]. Available: https://commons.lbl.gov/display/cpp/100G+Intrusion+Detection
[13] Proofpoint Inc., “Emerging Threats Open Ruleset,” Aug 2018. [Online]. Available: https://rules.emergingthreats.net/
[14] M. Jonkman, Aug 2018. [Online]. Available: http://doc.emergingthreats.net/bin/view/Main/NewUserGuide
[15] ——, “What Every IDS User Should Do,” Aug 2018. [Online]. Available: http://doc.emergingthreats.net/bin/view/Main/WhatEveryIDSUserShouldDo
[16] A. Kujawa et al., “Cybercrime tactics and techniques Q2 2017,” 2017. [Online]. Available: https://www.malwarebytes.com/pdf/white-papers/CybercrimeTacticsAndTechniques-Q2-2017.pdf