On Network Monitoring for Intrusion Detection

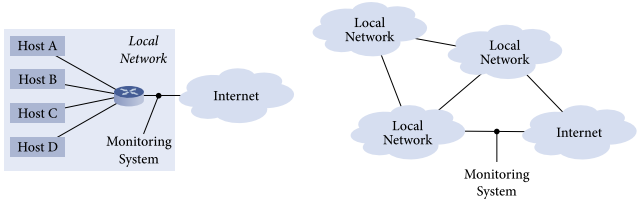
Tobias Limmer* and Falko Dressler†
*Siemens CERT, Munich, Germany
†Institute of Computer Science, University of Innsbruck, Austria
tobias.limmer@siemens.com, falko.dressler@uibk.ac.at

***Abstract*—Network-based intrusion detection approaches are frequently in use to track down malicious activities. However, all approaches in the field have always been struggling to cope with increasing network speeds. This article investigates the reasons behind this fact by providing a detailed analysis of methodologies that are commonly used by Network-based Intrusion Detection Systems (NIDSs), thereby focusing on the layer of packet capturing and data filtering. Recently, the performance of NIDSs has been greatly improved by new software-based and hardware-based techniques. On the one hand, improvements and optimizations within the processing chain allow faster packet capturing and analysis, and on the other hand, the advance of multi-core processors encourages the parallelization of NIDSs, thus achieving further speedups. Especially in the area of passive data analysis with only limited control on the incoming data rate, Amdahl's law poses strict limits on parallelization and achievable processing speeds for NIDSs.**

I. Introduction

Many different techniques are available for the detection of malicious software that is being executed on computers. The most popular methods are host-based concepts like virus scanners. On the other hand, network-based detection methods, even though known already for a long time, are used with varying success depending on the requirements and the operational environment of the deployed systems [1]. Nevertheless, network-based intrusion detection has become an integral part of many defense systems. This article intends to explore one of the current challenges in Network-based Intrusion Detection Systems (NIDSs): performance issues, which have always been a concern for algorithms developed in the computing world, and, of course, also for NIDSs with today's network link speeds of 10 GBit/s and more [2]–[6].

The amount of processed data heavily depends on the placement of monitoring sensors, i.e., on the amount of sensors and the locations within a network where data is captured. Therefore, we will investigate optimal placement of sensors in more detail and analyze their effects on the performance of NIDSs. In general, we can differentiate between two types of monitoring sensors: dedicated and integrated sensors. Dedicated systems usually receive read-only data from network taps. These network taps can either be separate devices that only mirror one specific link, or managed network switches or routers, where specific ports or subnets can be selected for mirroring. In contrast, integrated sensors are directly embedded in switches or routers and typically aggregate monitored packets to flows. These flows are then exported using protocols like Netflow or Internet Protocol Flow Information Export (IPFIX) [7], [8]. When routing devices directly monitor and aggregate data, no extra dedicated device needs to be set up to monitor the link.



(a) Small local network (b) Large network with multiple uplinks

Fig. 1: Placement options of monitoring sensor

The location where monitoring systems and Intrusion Detection Systems (IDSs) are to be placed depends on the size of the network: small networks usually only have a single Internet uplink, an example is shown in Figure 1a. If these links are monitored, the vast majority of attacks will detected, as they are transferred over these links coming from an external network, usually the Internet. The bigger a network becomes, the more difficult the placement of monitoring points will be. Networks will not be at a central location any more, but distributed over multiple branches, each probably having multiple uplinks to communicate with each other and to external networks (Figure 1b). To detect attacks from the Internet, all uplinks have to be monitored.

Another problem arises with bigger network sizes: the larger the network becomes, the more people have access to it and so-called insider attacks are made more probable. Insiders, either voluntarily or involuntarily, bypass Internet gateways, e.g., using infected mobile devices. A recent example is the worm Stuxnet that was designed to attack from *inside* a network [9]: the malware spread primarily via USB stick. Once attached to a computer, it exploited a vulnerability in the Windows operating system and infected the host. From this point on, the malware spread within the local network using multiple vulnerable services on Windows hosts. It was not designed to spread using external networks or the Internet, so sensors placed at the Internet uplink would not necessarily have detected the malware. Only sensors placed within the local network would have been able to detect the virus's spreading – in effect, the general principle of data mining, the more input data is available, the better the results will be, also applies to network monitoring and intrusion detection.

This principle also applies on a larger scale: Rajab et al. [10] analyzed how detection effectiveness of self-propagating worms improves when the monitoring sensors are not clustered at

geographically close locations, but distributed in different parts of the Internet. They assume a non-uniform distribution of worms in the network and build a simulation model based on data from the DShield project.[1] Their results show that detection times are 4 to 100 times shorter with distributed sensors compared to a single sensor monitoring the same total network size, where strategic placement of the sensors close to vulnerable networks plays an important role. It is disputable whether the authors' assumption for the malware distribution is realistic, but the problem of insider threats should not be underestimated [11]. Thus, sensors placed only at the Internet uplinks of large networks may not provide enough network coverage for adequate detection qualities. Furthermore, in local networks, packets for bidirectional communication are usually not broadcast to all stations. So, to capture all transferred data, all network links or interconnecting devices within a network need to be monitored. This is not feasible even within small networks due to the very high data rates.

As can be seen, performance is still a very important issue for network-based intrusion detection. This article describes the basics of NIDS providing a basis discussions of the challenges in NIDS. Although specialized hardware-based solutions for very fast intrusion detection have been available for years, often off-the-shelf hardware is preferred for NIDS. For these systems, performance is a concern for current network speeds, so we describe optimized software-based packet capturing and detection techniques that focus on performance aspects. In the second part, we present a specific example relying on parallelization in multi-core systems.

II. Network-based Intrusion Detection

In general, NIDSs can be differentiated in *active* and *passive* approaches. Passive approaches just monitor (observe) and analyze the network traffic without influencing it, e.g., by duplicating the data transferred over the network under observation. It is not possible to alter the transferred traffic. In contrast, active approaches directly influence the transferred traffic to gain more information about hosts in the network (e.g., network scanners), or to remove suspected malicious data from the network stream. Systems in this category are frequently called Intrusion Prevention Systems (IPSs). Active approaches that modify the network streams may influence delays and adjust contents of the packets. Especially for timing-critical network applications like video conferencing, this effect may be problematic.

Depending on the type of input data, NIDS techniques can be differentiated in *payload-based* and *header-based* classes. Payload-based methods may analyze all data contained within network packets. On the other hand, header-based approaches only analyze data contained in network packet headers, and payload is dropped beforehand. The amount of data that needs to be processed by those header-based approaches is only a fraction of the data, so in general, header-based approaches show a better performance compared to payload-based methods. Assuming an average packet size of 700 B and a header size (network and transport layer) of 60 B per packet, this results in a fraction of roughly $\frac{1}{10}$th of the original amount of data. Furthermore, the data processing of header-based and payload-based methods differs significantly: header-based algorithms need to look up data within the packet headers, which have a predefined structure. Often, packets may be aggregated using preprocessing stages like flow aggregation, further reducing the data to be processed. In contrast, payload-based methods need to recreate the original data that was transmitted by the sending application. In the case of stream-based protocols, separate packets need to be reassembled to the original data stream. This process is called IP defragmentation for the network layer and TCP reassembly for the transport layer. This preprocessing stage further slows down the operation of payload-based detection approaches.[2]

A. Performance of Packet Capturing

The performance of NIDSs is influenced by multiple factors and on different levels: the *packet capturing* layer attempts to capture packets from the networking hardware as fast as possible and provides this information to the *detection algorithm* layer. As the packet capturing system handles all incoming packets, it needs to be as efficient as possible to prevent packet drops. Packets are described as *dropped* or *lost* when they were not forwarded by the packet capturing module. The goal for all monitoring systems is to prevent (random) packet drops to enable succeeding analysis systems to gather as much information as possible.

In general, two approaches can be differentiated: *software-based* capturing solutions that have no special requirements for the networking interface, and *hardware-based* solutions specialized on custom hardware that are designed to offload computational costs to hardware for higher performance. If packet capturing solutions need to be cost-effective, end users often favor software-based solutions using off-the-shelf hardware. Linux is currently a very good choice for a packet capturing system, because there are high-performance libraries available optimizing the Linux kernel specifically for this task.

Packet capturing performance is mainly dependent on the incoming packet rate, not on the packet size (i.e., the resulting incoming data rate). This is because of the overhead that is induced by every received packet: all structures in software and hardware processing systems are designed to handle individual packets, so there is a specific overhead that is caused by each processed packet. For example, individual packet-specific checks have to be performed for implemented structures, pointers need to be updated, critical sections need to be adhered, and so on. Because of this, packet capturing systems are generally able to process higher data rates when the average size of captured packets is high. But on the other hand, large packets also influence the performance due to copying operations of packet contents in the RAM. Additionally, CPU caches tend to provide fewer cache hits due to larger packet buffer sizes, thus leading to cache thrashing and a higher number of accesses to slower memory types.

[1] http://www.dshield.org/

[2] Compared to the performance of the actual detection algorithm, this preprocessing stage may cause only marginal overhead, as is the case for many popular payload-based NIDSs like Snort.

These problems can be partly alleviated if a snap length is configured for monitoring. This configuration parameter specifies the data buffer size that is used for capturing network packets. If the size of a network packet exceeds the buffer size, it is truncated. Using snap lengths, the real packet length does not matter, as only the data buffer size is important for the succeeding processing steps.

The de-facto standard of receiving network packets from the operating system is Packet Capturing (PCAP). It is an open-source library for capturing network traffic and is available for various operating systems. It offers capturing packets from network interfaces and provides packet filtering capabilities. For use in high-speed networks, PF_RING by Luca Deri [12] offers an improvement to the memory-mapped version of the PCAP library. In contrast to PCAP, PF_RING does not forward received packets to upper layers of the network stack by default. Network capturing usually uses dedicated network interfaces and all packets arriving on this interface need not to be processed by the network stack. The ring buffer is memory-mapped to user-space and can be directly accessed by the monitoring application. As a result, PF_RING shows higher performance than the memory-mapped PCAP library with small packets. Deri also introduced DNA (Direct NIC Access) for further performance improvements [13], where he modified network card drivers so that they directly write packet data into PF_RING's buffer. This technique circumvents Network API (NAPI) and achieves even higher packet capturing rates.

In contrast to monitoring in networks with data rates below 1 GBit/s, monitoring of higher data rates requires hardware support for processing the incoming data. Hardware-based systems have commonly been based on network monitoring adapters using FPGA technology for fast data preprocessing and a large packet buffer for data transfer to the monitoring system. The FPGA is geared to offering flexible packet filtering and aggregation capabilities like flow aggregation. Using this method, a significant part of the processing time can be offloaded from the CPU to the monitoring hardware. Additionally, packet bursts can be remediated by the buffer with no influence on the CPU caches.

In order to improve monitoring in high-speed networks, off-the-shelf network interfaces for 1 GBit/s and 10 GBit/s networks already support multiple techniques to offload computations to the networking hardware. Examples are *checksum offloading* that calculates all packets' checksums on the networking interface, or *TCP Segmentation Offload* that splits big data chunks into multiple TCP packets according to the current Maximum Transmission Unit (MTU). Recently released 10 GBit/s networking interfaces additionally offer hardware support for optimized network monitoring techniques and multi-core CPUs.[3]

B. NIDS Optimization Techniques

After capturing packets from the network, they need to be analyzed by detection algorithms. The performance of these algorithms is mainly influenced, of course, by the amount of input data that actually has to be processed by the algorithm, and the performance of the algorithm itself. As the field of proposed detection techniques is very diverse, we only focus on the most popular type of algorithms, signature-based techniques that process packet payload and explain methods for preprocessing their input data to improve performance.

Payload-based detection algorithms use multiple methods to reduce input data before payload is processed, and try to speed up the processing of the payload itself. The following list describes other techniques for popular payload-based systems which are currently in use:

- *Filtering based on header data* is based on flow aggregation [7], [14]. Header data (e.g., source and destination addresses, ports, packet size, protocol information and flags) is checked by rules to filter data streams before the payload is analyzed [4] – this way, much irrelevant data may be filtered out before expensive pattern matching algorithms process the payload.
- *Use of efficient rules* is important, as, besides simple string matching, some signature-based IDSs support regular expressions for pattern matching. In theory, finding a match for regular expressions belongs to the class of NP-complete problems [15], [16]. This property can be exploited in signature-based IDSs if regular expressions are supported by the system. Snort also supports regular expressions, and evaluations have showed that specially crafted network data may cause the system to take up to one second per packet for pattern matching [17]. To counter performance problems in normal network traffic, regular expressions are often only evaluated if a simple content match was successful.
- *Optimization of pattern matching algorithms* has been a goal within computer science for a long time. Many new algorithms have been proposed in recent years that deal with controlling state-space explosion and reducing per-flow state [18]–[20], but only offer limited improvements in performance.
- *Use of specialized hardware* is another option. IDSs are often software-based and do not require specialized hardware. There have been several efforts that use hardware acceleration for speeding up the payload matching process, like graphic cards or FPGAs [5], [21], [22]. Chips optimized for pattern matching reach speeds of up to 10 GBit/s [18].

One of the most promising approaches to improve the performance of the detection algorithms is to make use of the current trend to *parallelization* in multi-core systems. In general, two different approaches can be considered: incoming data can be forwarded within a line of data processing modules, where each module receives the output of the preceding module. Parallelization is accomplished by executing tasks in the modules within separate CPU cores. This approach is very efficient when modules require a high amount of processing power, so that CPU cores can operate at a high utilization ratio. As an alternative, the whole processing pipeline may also be placed on a single CPU core. Then all incoming data has to be split up in equal parts and fed into disjunct parallel processing pipelines. This solution is only feasible if a single

[3] An example is the Intel X520 networking interface.

pipeline receives all data that is needed for processing. As an example, a pipeline performing 5-tuple flow aggregation must see all network packets belonging to the processed 5-tuple flows. If the pipeline did not receive all packets relevant for the aggregated flows, the data contained in the flow records would be incomplete. To guarantee that all packets belonging to the same flow are assigned to the same processing pipeline, hash functions can be used for the assignment decision. Xinidis et al. [23] proposed one solution that computes a hash value from all flow key fields, and hardware-based acceleration cards, e.g., Intel X520, also uses hashing for distributing incoming packets to succeeding queues [24].

To exploit the power of multi-core systems, NIDSs have also been parallelized and the incoming network traffic is distributed by a load balancer to individual instances for further analysis [25]. According to Amdahl's law, high performance advantages from parallel processing are unlikely if fine-grained coordination between the parallel tasks is required. Therefore, one of those instances typically represents a fully self-contained IDS. A central component, commonly called a load balancer, distributes traffic to parallel IDS instances. The trend to multi-core environments has been exploited in [26], where a model for an intrusion prevention system based on the IDS Bro has been introduced. This approach exploits multi-core and multi-thread CPU architectures by splitting data processing in multiple stages and optimizes them for high-performance.

III. Case Study: Adaptive Load Balancing for NIDS

In this section, we describe a state-of-the art adaptive filtering and load balancing system for NIDSs [3] to provide more insights into the performance problems and propose possible solutions. In this system, a *load balancing component* decides about incoming packets whether to drop or forward them, packets are forwarded to *analyzers* or *IDSs*, whereas single disjunct instances of these systems are called *analyzer instances* or *IDS instances*, each usually running on separate CPU cores.

A. Considerations and assumptions about the environment

In the following, we discuss important aspects of a load balancer for NIDSs and how these aspects differ from a typical load balancing scenario used for actively serving requests, such as a load balancer that distributes incoming HTTP requests to multiple web servers.

Very high data rates in combination with traffic bursts: With current network link speeds of up to 10 GBit/s, hardware provided for network monitoring cannot cope with observed data rates. Buffers are commonly used to temporarily alleviate spikes in the traffic, but, due to the high data rates, buffering can only be used in a limited way: In a web server scenario, buffered requests only require a few KiB's. A high number of requests can be easily cached and processed at a later time [27]. In a passive monitoring system, all incoming data needs to be stored in a buffer. Assuming a data rate of 10 GBit/s, a cache holding only 10 s of data would require a buffer of 12.5 GiB.

Hard real-time requirements for analysis: In contrast to other load balancing scenarios, passive network monitoring systems cannot influence incoming data rates. If the incoming data rate is too high and internal buffers are full, the system will have to drop incoming packets and lose information.

Serving all requests is not compulsory: It is beneficial when all network traffic can be analyzed by the system, but not compulsory. Normal network operation is not affected by an overloaded passive monitoring system.

In summary, high data rates in combination with hard real-time requirements are the main problems to tackle in a dynamic load balancing system. These problems lead to considerations about the frequency of the selection process, i.e., how often it should be decided what data will be dropped or forwarded. For load balancing systems with disjoint requests and low request rates, this can be decided for each individual request. When the decision is reached, all packets belonging to the request will automatically be forwarded with little overhead. In a network monitoring scenario with high data rates, it is not feasible to perform complex calculations for the selection decision due to the hard real-time requirements. This is also valid for decisions about groups of packets that do not need to be performed per packet. In the presented solution, a two-stage selection process is used where a lightweight algorithm performs per-packet decisions. Periodically, this algorithm is configured by a more complex decision process.

Full packet information: Depending on the techniques used in the analyzer, it is important to control which packets are sent to the same analyzer instance. Methods that operate statelessly by examining individual packets, like pattern matching on header data or payload matching for individual packets (sometimes this is also called *Deep Packet Inspection (DPI)*), do not pose any requirements on the load balancing process. However, most techniques keep state between packets, either for unidirectional flows as required for flow aggregation, or for bidirectional 5-tuple flows representing single UDP or TCP connections as required by many payload-based IDSs tracking connections on the application layer. Another aspect is the required level of detail: payload-based IDS requires the full packet payload, whereas many anomaly detection schemes only require aggregated statistics about network flows.

Host-based packet selection: Commonly used load balancing methods for parallel IDSs are based on hashes over the packets' 5-tuple, so all packets belonging to the same connection are forwarded to the same analyzer instance. The presented approach additionally assumes that the analyzer keeps state for local hosts, as this is the case in some malware detection systems [28]. To preserve important semantics inherent in the network data, we forward all packets related to the same local host, that means all packets that were sent or will be received by the host, to the same analyzer.

As the presented selection mechanism is based on the packets' relationship to local hosts, weights may be assigned to these hosts and be used as a configurable selection criterion. Local networks often include hosts with high security risks, such as authentication servers, and hosts with lower importance, such as common workstations. Depending on the security requirements, the total time a host's traffic is forwarded to an analyzer should depend on the configured weight of the host. Furthermore, hosts located behind a Network Address Translation (NAT) gateway within the local network are

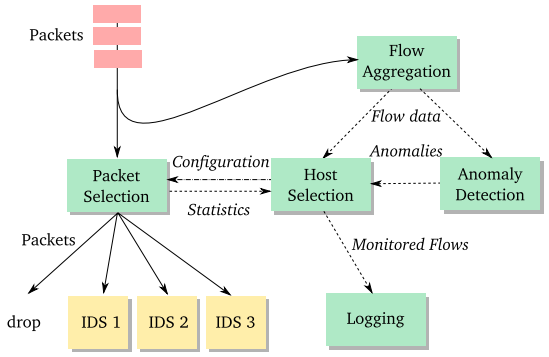


Fig. 2: Overview of the load balancing system for NIDSs

disadvantaged, as the load balancer regards the NAT gateway as a single host, thus assigning monitoring time for a single host to the gateway. Raising the NAT gateway's priority rectifies this issue.

B. System overview

Figure 2 shows an overview of the load balancing system. Packets captured from the network are processed by a packet selection module, which uses a lightweight decision algorithm that determines whether the received packet should be dropped or forwarded to a connected IDS instance. At the same time, all packets are fed into a flow aggregation module that produces standard conform IPFIX records. This module can easily be integrated into hardware implementations of routers or switches that also support IPFIX – in that case, the presented system would directly receive flow data. This flow data is used by the host selection module to predict the traffic volume. Furthermore, anomaly detection algorithms may process the flow data for detecting conspicuous traffic and report anomalies to the host selection module in order to immediately select affected hosts for monitoring.

The host selection module works interval-based. After each interval (also called a round), a set of hosts in the local network is assigned to each IDS instance and the configuration is transferred to the packet selection module. In addition, statistics, such as the number of forwarded packets or packets dropped due to overload, are reported. The host selection module performs local calculations based on these statistics and the received flow data. Logging functionality is integrated to collect information about forwarded and dropped packets for later forensic analysis, providing exact statistics about the packets that were analyzed by the IDS instances. All modules except the IDS instances have been implemented in a prototype using the monitoring framework Vermont [29].[4]

C. Selection process

The presented selection process is separated into two parts: A lightweight packet selection module that decides for each incoming packet, whether it should be dropped or forwarded to one of the IDS instances; and an interval-based host selection module that prepares selection criteria using a computationally more expensive algorithm.

High-performance selection processes for network packets are usually based on simple table lookups using key elements at fixed positions in observed packets (in most cases from the header). Examples include IP addresses, ports, protocols, or other transport layer-specific fields. Only IP address fields are used for the per-packet selection process. If either the source or destination IP address is in a predefined list of addresses, the packet will be forwarded to the corresponding IDS instance. If this instance is not able to process the forwarded traffic, hosts may be dropped from this list. Recent off-the-shelf network interfaces provide programmable hardware-based filtering and load balancing techniques [6] that may be used to implement this simple selection algorithm.

In a configurable interval t, the configuration needed for the per-packet decision is built. Essentially, a set of hosts is prepared for each analyzer instance defining the packets to be forwarded to this IDS. This set of hosts is determined by a priority model that assigns a priority to each host. Each interval, the priorities are adjusted according to whether hosts were selected for analysis in the previous interval. Hosts with the highest priorities are chosen to be processed by the analyzers. The number of hosts is determined by predicting the data rates transferred and received by the hosts and comparing these rates to the maximum possible data rate that the analyzer instances are able to process. The administrator is able to adjust these priorities.

D. Data forwarding to IDS instances

Efficient data transfer to the asynchronously executed IDS instances is a challenging issue at the envisioned data rates. In UNIX operating systems, commonly POSIX pipes are used for unidirectional data transfer between processes. In these pipes, data is copied between user and kernel-space. The overhead of those copy operations is avoied by implementing a lock-free ring buffer [30] of size n in shared memory. As the forwarded traffic is highly dynamic, the IDS instance is not always able to process the incoming packets in time and the connecting ring buffer will fill up and lead to random, uncontrolled packet dropping. The system's implementation alleviates this problem and introduces *controlled packet dropping*. This method works by continuously monitoring the fill level of the buffer. If it exceeds a threshold, packets will be dropped in a controlled way.

E. Performance of the load-balancing system

To compare the load balanced parallel IDS with a normal IDS in conditions with packet-loss, an experiment was conducted using a 4100 s network packet trace from a University's network with an average data rate of 430 Mbit/s. The packet trace was prepared by randomly dropping packets at a rate $p \in \{0.0, \ldots 0.9\}$, then all reported events were recorded. In Figure 3, the number of TCP and UDP related events reported by Snort are shown for processing the trace with different packet drop ratios, where duplicate events for the same connection were omitted. Additionally, multiple runs were

[4] Vermont has been released under the GPLv2, http://vermont.berlios.de

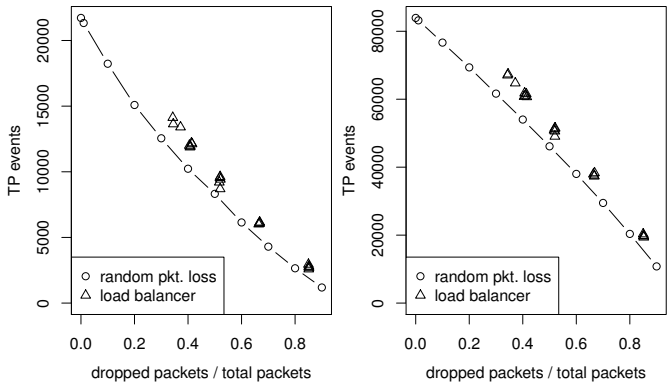


Fig. 3: Number of detected events with random packet losses compared to the load balancing system. Left/right graph: TCP/UDP-related events

performed with a selection interval of 10 s. To obtain realistic results, the packet trace was processed in real-time and limited the processing rate of each Snort instance to get comparable drop ratios. A lower packet loss rate than 40 % was not possible, as the three IDS instances were at maximum load and could not process the needed data rates. As reported in [3], the load balanced IDSs run detected 44 % more events compared to the IDS run with random packet drops at a packet loss rate of 85%. This is due to the host-based selection process adhering to a minimal monitoring interval that forwards semantically interdependent packets to the same analyzers. Similarly, rules matching UDP content were more often reported by the load balancing system. This is due to a high number of events matching DNS queries. The involved DNS servers did not produce much traffic at the Internet uplink, so the load balancer often temporarily assigned the servers to an IDS which led to a higher amount of reported UDP events.

IV. Conclusion

In conclusion, monitoring for network-based intrusion detection is still a challenging task, especially in high-speed networks transmitting data at multiple gigabit per second. There are, however, several techniques available to speed-up this process. Besides filtering techniques, reducing the number of bytes to be analyzed and optimizations in signature detection, substantial improvements can be achieved by exploiting parallelization on multi-core CPUs. In form of a case study, we presented a load balancing system specialized for parallel payload-based IDSs on multi-core systems using prioritized flows. Using a prototype, we experimentally evaluated the performance of the system both in quality and quantity. The system is able to cope with highly fluctuating data rates, which is a common property of todays Internet network traffic. A key feature is the minimal monitoring interval ensuring that consecutive packets needed for pattern matches will be forwarded to the IDS with a high probability.

References

[1] C. V. Zhou, C. Leckie, and S. Karunasekera, "A Survey of Coordinated Attacks and Collaborative Intrusion Detection," *Computers & Security*, vol. 29, no. 1, pp. 124–140, 2010.

[2] H. Dreger, A. Feldmann, V. Paxson, and R. Sommer, "Predicting the resource consumption of network intrusion detection systems," *ACM SIGMETRICS Performance Evaluation Review*, vol. 36, no. 1, pp. 437–438, 2008.

[3] T. Limmer and F. Dressler, "Adaptive Load Balancing for Parallel IDS on Multi-Core Systems using Prioritized Flows," in *20th International Conference on Computer Communication Networks (ICCCN 2011)*. Maui, HI: IEEE, July/August 2011, pp. 1–8.

[4] ——, "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, April 2011, pp. 833–838.

[5] G. Vasiliadis, S. Antonatos, M. Polychronakis, E. P. Markatos, and S. Ioannidis, "Gnort: High Performance Network Intrusion Detection Using Graphics Processors," in *11th International Symposium on Recent Advances in Intrusion Detection (RAID 2008)*, vol. LNCS 5230. Cambridge, MA: Springer, September 2008, pp. 116–134.

[6] L. Deri, J. Gasparakis, P. J. Waskiewicz, and F. Fusco, "Wire-Speed Hardware-Assisted Traffic Filtering with Mainstream Network Adapters," in *1st International Workshop on Network Embedded Management and Applications (NEMA 2010)*. Niagara Falls, Canada: Springer, October 2010.

[7] B. Claise, "Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of IP Traffic Flow Information," IETF, RFC 5101, January 2008.

[8] J. Quittek, S. Bryant, B. Claise, P. Aitken, and J. Meyer, "Information Model for IP Flow Information Export," IETF, RFC 5102, January 2008.

[9] B. Schneier, "The story behind the stuxnet virus," 2010. [Online]. Available: {http://www.forbes.com/2010/10/06/iran-nuclear-computer-technology-security-stuxnet-worm.html}

[10] M. A. Rajab, F. Monrose, and A. Terzis, "On the Effectiveness of Distributed Worm Monitoring," in *14th USENIX Security Symposium*, Baltimore, MD, July 2005.

[11] S. L. Pfleeger and S. J. Stolfo, "Addressing the Insider Threat," *IEEE Security and Privacy*, vol. 7, no. 6, pp. 10–13, 2009.

[12] L. Deri, "Improving Passive Packet Capture: Beyond Device Polling," in *4th International System Administration and Network Engineering Conference (SANE 2004)*, Amsterdam, The Netherlands, September 2004.

[13] ——, "Introducing PF-RING DNA (Direct NIC Access)," 2010. [Online]. Available: {http://www.ntop.org/blog/?p=50}

[14] I. Sourdis, V. Dimopoulos, D. N. Pnevmatikatos, and S. Vassiliadis, "Packet Pre-filtering for Network Intrusion Detection," in *ACM/IEEE Symposium on Architecture for Networking and Communications Systems (ANCS 2006)*. San Jose, CA: ACM, December 2006, pp. 183–192.

[15] W. Drewry and T. Ormandy, "Insecure Context Switching: Inoculating Regular Expressions for Survivability," in *2nd USENIX Workshop on Offensive Technologies (WOOT 2008)*. San Jose, CA: USENIX, July 2008.

[16] K. Ellul, B. Krawetz, J. Shallit, and M.-w. Wang, "Regular Expressions: New Results and Open Problems," *Journal of Automata, Languages and Combinatorics*, vol. 10, no. 4, pp. 407–437, 2005.

[17] K. Namjoshi and G. Narlikar, "Robust and Fast Pattern Matching for Intrusion Detection," in *29th IEEE Conference on Computer Communications (INFOCOM 2010)*. San Diego, CA: IEEE, March 2010.

[18] R. Smith, C. Estan, S. Jha, and S. Kong, "Deflating the Big Bang: Fast and Scalable Deep Packet Inspection with Extended Finite Automata," in *ACM SIGCOMM 2008*, Seattle, WA, August 2008, pp. 207–218.

[19] D. Ficara, S. Giordano, G. Procissi, F. Vitucci, G. Antichi, and A. D. Pietro, "An improved DFA for Fast Regular Expression Matching," *Computer Communication Review*, vol. 38, no. 5, pp. 29–40, 2008.

[20] M. Becchi, C. Wiseman, and P. Crowley, "Evaluating Regular Expression Matching Engines on Network and General Purpose Processors," in *ACM/IEEE Symposium on Architecture for Networking and Communications Systems (ANCS 2009)*, Princeton, NJ, October 2009, pp. 30–39.

[21] G. Vasiliadis, M. Polychronakis, S. Antonatos, E. P. Markatos, and S. Ioannidis, "Regular Expression Matching on Graphics Hardware for Intrusion Detection," in *12th International Symposium on Recent Advances in Intrusion Detection (RAID 2009)*, vol. LNCS 5758. Saint-Malo, France: Springer, September 2009, pp. 265–283.

[22] S. Yusuf and W. Luk, "Bitwise Optimised CAM for Network Intrusion Detection Systems," in *15th IEEE International Conference on Field Programmable Logic and Applications (FPL 2005)*, Tampere, Finnland, August 2005, pp. 444–449.

[23] K. Xinidis, I. Charitakis, S. Antonatos, K. G. Anagnostakis, and E. P. Markatos, "An Active Splitter Architecture for Intrusion Detection and Prevention," *IEEE Transactions on Dependable and Secure Computing*, vol. 3, no. 1, pp. 31–44, 2006.

[24] F. Fusco and L. Deri, "High Speed Network Traffic Analysis with Commodity Multi-core Systems," in *10th ACM Internet Measurement Conference (IMC 2010)*. Melbourne, Australia: ACM, November 2010, pp. 218–224.

[25] M. Vallentin, R. Sommer, J. Lee, C. L. V. Paxson, and B. Tierney, "The NIDS Cluster: Scalable, Stateful Network Intrusion Detectionon Commodity Hardware," in *10th International Symposium on Recent Advances in Intrusion Detection (RAID 2007)*, 2007, pp. 107–126.

[26] R. Sommer, V. Paxson, and N. Weaver, "An architecture for Exploiting Multi-core Processors to Parallelize Network Intrusion Prevention," *Concurrency and Computation: Practice and Experience*, vol. 21, no. 10, pp. 1255–1279, 2009.

[27] M. Andreolini, S. Casolari, and M. Colajanni, "Models and Framework for Supporting Runtime Decisions in Web-based Systems," *ACM Transactions on the Web*, vol. 2, no. 3, July 2008.

[28] G. Gu, R. Perdisci, J. Zhang, and W. Lee, "BotMiner: Clustering Analysis of Network Traffic for Protocol- and Structure-Independent Botnet Detection," in *17th USENIX Security Symposium*, ser. USENIX Security Symposium. San Jose, CA: USENIX, July 2008, pp. 139–154.

[29] R. T. Lampert, C. Sommer, G. Münz, and F. Dressler, "Vermont - A Versatile Monitoring Toolkit for IPFIX and PSAMP," in *IEEE/IST Workshop on Monitoring, Attack Detection and Mitigation (MonAM 2006)*. Tübingen, Germany: IEEE, September 2006, pp. 62–65.

[30] P. P. C. Lee, T. Bu, and G. P. Chandranmenon, "A Lock-free, Cache-efficient Multi-core Synchronization Mechanism for Line-rate Network Traffic Monitoring," in *IEEE International Symposium on Parallel and Distributed Processing (IPDPS 2010)*, Atlanta, GA, April 2010, pp. 1–12.