Adaptive Load Balancing for Parallel IDS on Multi-Core Systems using Prioritized Flows

Tobias Limmer* and Falko Dressler†
*Dept. of Computer Science, University of Erlangen, Germany
†Institute of Computer Science, University of Innsbruck, Austria
limmer@cs.fau.de, dressler@ieee.org

***Abstract*—We describe a load balancing system for parallel intrusion detection on multi-core systems using a novel model allowing fine-grained selection of the network traffic to be analyzed. The system receives data from a network and distributes it to multiple Intrusion Detection Systems (IDSs) running on individual CPU cores. In contrast to related approaches, we do not assume a static association of flows to IDS processes but adaptively determine the load of each IDS process to allocate network flows for a limited time window. We developed a priority model for the selection of network data and the assignment process. Special emphasis is given to environments with highly dynamic network traffic, where only a fraction of all data can be analyzed due to system constraints. We show that IDSs analyzing packet payload data disproportionately suffer from random packet drops due to overload. Our proposed system ensures loss-free analysis for selected data streams in a specified time interval. Our primary focus lies on the treatment of dynamic network behavior: neither data should be lost unintentionally, nor analysis processes should be needlessly idle. To evaluate the priority model and assignment systems, we implemented a prototype and evaluated it with real network traffic.**

I. Introduction

The analysis of network traffic has become common practice in current networks. Depending on the use case of the analysis, methods require different levels of aggregation for input data. Typical examples include simple metering for accounting and billing purposes, network flow analysis for the estimation of traffic properties, and Deep Packet Inspection (DPI) or anomaly detection in the field of network security [1]. One of the computationally most expensive methods currently in use is payload-based detection, in use in Intrusion Detection Systems (IDSs) such as Snort. Even though network link speeds of 10 GBit/s are becoming common in modern networks, such payload-based IDSs already struggle to cope with a tenth of that speed [2]. More complex analyses like virus and spam detection are simply not possible at those data rates. In the scope of this paper, we study the applicability of payload-based intrusion detection in high-speed networks, which, according to the mentioned performance limitations, is not possible for each individual packet.

In previous years, Moore's law ensured steadily increasing processing speeds of CPU cores, so that single IDS instances were able to process increasing amounts of data. With the spread of multi-core CPUs, processing capabilities have been further advanced and overall performance of multi-core systems shows excellent performance increases [3]. To exploit the power of multi-core systems, IDSs have been parallelized and the incoming network traffic is distributed by a load balancer to individual instances for further analysis [3]–[6]. 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.

Assuming a realistic data processing performance of about 200 MBit/s for a single IDS instance running Snort with a common ruleset, at least 100(!) instances would be needed to process a fully loaded 10 GBit/s duplex link without data loss. To distribute incoming data to parallel IDS instances, static hashing is commonly used [4]. This approach heavily relies on the hash function to distribute traffic evenly to all available instances. In dynamic environments, such a static assignment will not keep a balanced configuration. So, several approaches have been introduced in the past to dynamically adjust to changes [7]. This concept works very well in scenarios where incoming data rates can easily be processed by all available IDS instances. However, if network data rates become too high, packets will be randomly dropped by the system.

The challenging problem is to operate an IDS in a way that as many events can be detected as possible, even though it may be impossible to analyze all the received packets. We will show that distributed processing alone is not sufficient to handle security-related events: signatures may have to match multiple packets in a flow, thus, the assignment needs to be fixed for a minimal time before re-allocating the flow to another IDS instance. One way to solve this problem is to perform intelligent data filtering within individual connections [8]–[10], or data filtering of whole connections [11].

We propose a novel load balancing approach that is specifically suited for environments where available computational power cannot cope with incoming data rates. The selection method used for incoming network packets is based on a priority model. To the best of our knowledge, this is the first time that a host-based priority model is used in the context of adaptive load balancing of flows to IDS instances. We allow to assign a weight value to each host within the local subnetwork, which defines a preference model where, for example, security-critical servers can be preferred over non-critical systems in the selection process. This way, it is possible to increase the monitoring time per host. We show that random packet drops cause disproportionately high false-

978-1-4577-0636-3 /11/$26.00 ©2011 IEEE

negative event rates for payload-based IDSs like Snort. Fighting this problem, the load balancer ensures a minimal monitoring time for each monitored host. During the monitoring time, our adaptive load balancer interacts with the IDS by measuring its performance and by observing the fill level of queues in the system. This prevents packet drops as well as possible and ensures uninterrupted data analysis at the respective IDS instance. Our load balancing algorithm uses two levels with a per-packet and interval-based selection mechanism. The per-packet selection process is extremely lightweight and is designed to be directly implemented on hardware-based filters in the network interface for higher speeds [11].

II. Related Work

Performance deficiencies in the field of payload-based IDSs have been tackled by various means: speed improvements of algorithms within the IDS [12], offloading computation to specialized hardware [13], intelligent data filtering [8]–[10], and, of course, parallel execution of multiple IDS instances.

A central component, commonly called a load balancer, distributes traffic to parallel IDS instances. In early approaches, each of these instances was running on a separate host, and a central splitter was used for distributing the network packets. Static hashing methods operating on network and transport layer header fields determined the IDS instance the packet was forwarded to [14]. These static techniques were further improved with hashing algorithms that dynamically adapted to the current load [4], [15]. As employed IDS instances were usually disjoint and did not share state, a protocol for exchanging state information has been introduced [6]. This technique enabled pattern matching and anomaly detection across multiple connections. However, due to the high amount of shared states, synchronization proved to be difficult. The trend to multi-core environments has been exploited in [3], where a model for an intrusion prevention system based on the IDS Bro was introduced. This approach exploits multi-core and multi-thread CPU architectures by splitting data processing in multiple stages and optimizes them for high-performance.

In common practice, there are currently two open-source systems in use for performing parallelized payload-based network intrusion detection: Suricata[1], a parallelized version of the IDS Snort, and Snort in combination with the packet capturing framework PF_RING for load balancing [11].

III. Adaptive Load Balancing for IDS

This section covers the methodology used by our proposed traffic selection and load balancing system. We use the following terms in the description: the *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 a separate CPU core.

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

A. Considerations and assumptions about the environment

In the following, we will discuss important aspects of the overall system and how these aspects differ from a typical load balancing scenario used for actively serving requests.

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 [16]. 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 our solution, we use a two-stage selection process 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 also called *DPI*), do not pose any requirements on the load balancing process. However, most techniques keep state between packets, either for unidirectional flows like 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.

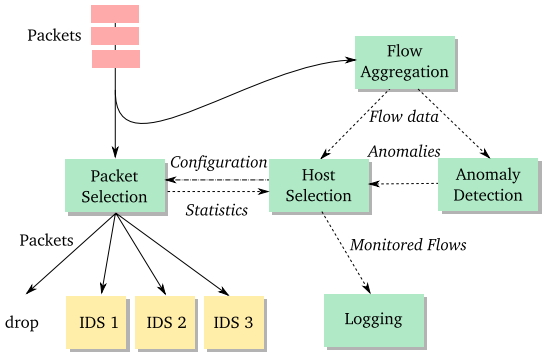

Figure 1. Overview of our proposed load balancing system

In our load balancing scenario, we assume that analyzers require whole packets for analysis, as this is the standard case for payload-based IDSs like Snort. Our view is to separate analysis techniques that perform payload matching from algorithms which are able to operate on aggregated data. Algorithms operating on aggregates do not need to be served by a packet-level load balancing system but may be integrated in a separate flow aggregation system.

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. Our approach additionally assumes that the analyzer keeps state for local hosts, as this is the case in some malware detection systems [17]. 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 our selection mechanism is based on the packets' relationship to local hosts, we may assign weights to these hosts and use them 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 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 1 shows an overview of the proposed 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 IP Flow Information Export (IPFIX) records. This module can easily be integrated into hardware implementations of routers or switches that also support IPFIX – in that case, our system would receive flow data directly from such a device. The flow data is used by the host selection module to predict traffic. 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 our monitoring framework Vermont [18][2].

C. Selection process

Our 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. We assume that the configured local network is only on one side of the monitored link, i.e. we do not observe traffic exchanged between local hosts. This ensures that traffic can be grouped into disjoint sets by the local hosts that will receive or sent the packets.

1) Per-packet level selection: 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. We only use the IP address fields 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 will be dropped from this list. We will call these *forcibly dropped hosts*. The algorithm will be described in detail in Section III-G. In our implementation, we are using a hash table for the lookup. Recent off-the-shelf network interfaces provide programmable hardware-based filtering and load balancing techniques [19] that may be used to implement this simple selection algorithm. Further software-based speed improvements are possible using Bloom filters.

2) Interval-based level selection: In a configurable interval t, we build the configuration needed for the per-packet decision. Essentially, a set of hosts is prepared for each analyzer instance defining the packets to be forwarded to this IDS. So, our algorithm differentiates single hosts from all hosts $H = \{h_0, h_1, \ldots, h_{n-1}\}, |H| = n$ within the local network.

[2]Released under the GPLv2 on http://vermont.berlios.de

Algorithm 1 Regular priority updates

1: $H_M \leftarrow$ hosts h_i with highest priorities that were contiguously monitored for mt_{min}, $|H_M| \leq m_{max}$
2: **for all** $p_i \in P$ **do** {Adjust priorities of monitored hosts}
3: **if** $h_i \in H_M$ **then**
4: $p'_i \leftarrow p_i(r-1) - \frac{1}{w_i}$
5: **else**
6: $p'_i \leftarrow p_i(r-1)$
7: **end if**
8: **end for**
9: **for all** $p_i \in P$ **do** {adjust all priorities to ensure that $p_{total} = \sum_{p_i \in P} p_i(r)$}
10: $p_i(r) \leftarrow p'_i + \frac{p_{total} - \sum_{j=0}^{n-1} p'_j(r)}{n}$
11: **end for**

a) Priority system: The selection process is primarily based on the priority system. A priority value is assigned to each host and is updated in each round. Only hosts with the highest priorities are selected for analysis. So, each host h_i is assigned a dynamic priority p_i, and we get a set of priorities $P = \{p_0, p_1, \ldots, p_{n-1}\}$. Variables like p_i change over time, so we describe a variable's value v at a specific round r as $v(r)$. At the starting round $r = 0$, we set all priorities to a uniformly distributed random value r_i in the range $[0,1)$, so $p_i(0) = r_i \ \forall \ 0 \leq i < n$. We require that the sum of all priorities p_{total} remains constant for each round r:

$$p_{total} = \sum_{i=0}^{n-1} p_i(r) = \sum_{i=0}^{n-1} r_i \tag{1}$$

$m_i(r)$ describes the number of rounds that a host h_i was analyzed from round 0 to round $r-1$. To specify the relative importance of hosts and to control the total amount of time a specific host is selected, each host h_i is assigned a weight w_i. Between each round, a selection algorithm ensures that hosts with highest priorities are selected and assigned to the analyzers. As will be discussed in Section IV-A, the time a host is contiguously monitored should be as high as possible. Therefore, we introduce a minimal time mt_{min} as configurable parameter that specifies how long a host needs to be monitored without interruption. When a host has been monitored for mt_{min} without interruption, its priority p_i is decreased according to its weight. If the host was removed from IDS assignment ahead ot time, its priority will not be decreased. It is possible to formally prove that for $r \to \infty$ a host's weight factor is proportional to the total time the host is monitored (proof omitted due to space limitations).

Algorithm 1 outlines the procedure to update each hosts' priority. Every round r, all hosts are sorted according to their priority p_i. At maximum, the top m_{max} hosts that were monitored for a time of at least mt_{min} are included in set H_M (line 1). Then, the priorities of all hosts $h \in H_M$ are decreased, because they were successfully analyzed (lines 2–8). To ensure that the sum of all hosts' priorities stays at p_{total}, each $p_i \in P$ is increased by a constant value that is equal for all hosts.

b) Preventing unstable models: Unfortunately, there are cases possible in which priorities may rise or fall indefinitely. In the following, we show how to avoid such cases.

Assuming a scenario, where all hosts are monitored by the system and have different weights $W = w_0, w_1, \ldots, w_{n-1}$. It is trivial to show that the priority of all hosts with a weight w_i higher than the average weight $\overline{w}$ will rise indefinitely. Without loss of generality, we additionally assume that the minimal monitoring time is set to $mt_{min} \leq t$, and there exists a maximum weight w_{max} of at least one host h_{max}. To prevent an infinite rise of h_{max}'s priority, the following inequality must be fulfilled:

$$\frac{1}{w_{max}} \geq \frac{M(r)^T \hat{W}}{n} \left[= \frac{p_{total} - \sum_{j=0}^{n-1} p'_j(r)}{n} \right], \tag{2}$$

whereby $\hat{W} = (1/w_0, 1/w_1, \ldots, 1/w_{n-1})^T$, and $M(r)$ is a vector of length n whose element at index i is set to 1 if h_i was monitored for at least mt_{min} time, else it is set to 0. As we assume that all hosts are monitored, $M(r) = \mathbf{1}$. In this case, Inequality 2 can only be fulfilled if $w_i = w_j, \forall w_i, w_j \in W$. This contradicts our assumption, thus, some priorities may rise and others may fall indefinitely.

In order to resolve this problem, we introduce a parameter m that limits the number of monitored hosts whose priority is decreased for each round, regardless of the actual number of hosts that are monitored. Again, we assume the worst case where the right side of Inequality 2 is highest, i.e. if all hosts with the lowest weight w are monitored. $M_{minw}(r,m)$ is derived from $M(r)$, whereby in M_{minw}, at maximum m elements are set to 1 for the monitored hosts with lowest weights. Now, we choose m such that

$$\frac{1}{w_{max}} \geq \frac{M_{minw}(r,m)^T \hat{W}}{n}. \tag{3}$$

NB: m should be chosen as high as possible, as the more priorities are unadjusted each round, the more information is lost about the current selection state of the system. In our current implementation, we calculate an upper bound of m at system startup.

D. Host assignment method

Our interval-based selection process allows regular selection of hosts in the system for monitoring. As we employ several IDS instances executed in parallel on multiple cores, the selected hosts need to be assigned to individual IDS instances. We assume that for each host h_i in interval r we have the priority $p_i(r)$, the traffic $te_i(r)$ expected to be transferred in this interval by h_i, and the traffic transferred in the previous interval $t_i(r-1)$. Only when the minimal monitoring time mt_{min} has elapsed, a host may be transferred to another IDS or removed from the set of monitored hosts. Therefore, $mt_i(r) = 0$ when h_i was not monitored in round $r-1$. We have q IDS instances with $IDS = \{ids_0, ids_1, \ldots\ ids_{q-1}\}$. It has been shown that the CPU load of a payload-based IDS is proportional to the rate of the processed traffic [20]. So, we use the processed traffic as a measure for the utilization of an IDS. The maximum data rate of ids_i is defined as $tmax_i$. All hosts assigned to ids_i

Algorithm 2 Host assignment to succeeding IDSs instances

```
1: Remove forcibly dropped hosts from each M_i
2: for all ids_i ∈ IDS and h_j ∈ M_i do
3:    if mt_j(r) ≥ mt_min then
4:       M_i ← M_i ∖ h_j
5:    end if
6: end for
7: for all ids_i ∈ IDS do
8:    while Σ_{j∈M_i} te_j(r) > tmax_i do
9:       h_k ← host in M_i with lowest priority p_k
10:      M_i ← M_i ∖ h_k
11:   end while
12: end for
13: Sort hosts in R with decreasing priority
14: for all ids_i ∈ IDS and h_j ∈ R do
15:   Remove hosts h_k from M_i with lowest priorities and
      p_k < p_j until te_j(r) ≤ tmax_i − Σ_{h_m∈M_i} te_m(r)
16:   add host h_j to M_i
17: end for
```

are in the set M_i. Thus, all monitored hosts are in the set $\bigcup_{i=0}^{q-1} M_i$, and the others are contained in $R = H \setminus \bigcup_{i=0}^{q-1} M_i$.

The problem of assigning hosts with priorities and expected traffic to IDSs with limited processing rate corresponds to the well known computational complex knapsack problem. However, much simpler heuristics can be used because of the large variance in the data rates of individual hosts and the usually high number of hosts monitored by a single IDS instance. Our proposed algorithm is shown in Algorithm 2. First, all hosts are removed that were forcibly dropped in the previous round by the per-packet level selection process. Then we iterate through the IDS instances and remove all monitored hosts h_k with $mt_k(r) > mt_{min}$ (lines 2–6). We ensure that no IDS instance is overloaded by removing hosts having the lowest priorities so that the maximum data rate $tmax_i$ of IDS i is not exceeded (lines 7–12). NB: we do not measure mt_k of the removed hosts. Finally, we compare the monitored hosts' priority p_k with the host's priority p_j of R. As long as hosts $h_k \in M_i$ with priorities $p_k < p_j$ can be removed from M_i, new hosts h_j will be added to M_i. To determine whether h_j will fit into M_i, the expected host data rates $te_j(r)$ are used in the calculation. This process is repeated for each IDS instance until no hosts can be added any more (lines 14–17).

This assignment process ensures that all hosts with highest priorities will be assigned to an IDS instance as soon as possible, while displacing hosts with lower priorities from assignment without regarding their monitoring time mt. The higher the priority, the higher the possibility that the host will be consecutively monitored for mt_{max}. Additionally, by only decreasing a monitored host's priority after it was removed from the assignment and was monitored for at least mt_{max}, the order of hosts regarding their priority will not change in the sets $M_i \; \forall \; i \in 0, \ldots q-1$. This is caused by the mechanism that increases all priorities to p_{total}.

E. Data rate prediction for hosts

The prediction of each host's data rate in the next round works as follows. Assuming a standard on-off traffic distribution for each host with switching intervals higher than our round length, we use the measured data rate $t_i(r)$ of host h_i as a basis for our prediction. This works for all hosts that transmitted data in the last round, i.e. the data rate was larger than zero. In this case, we set the expected data rate $te_i(r+1) = t_i(r)$. If we set $te_i(r+1)$ for all other hosts to $te_i(r+1) = t_i(r) = 0$, there is a certain probability that these hosts will transfer data in the next interval, i.e. the overall prediction of the hosts' data rate will not be correct and the IDS will be overloaded. To avoid this problem, we calculate an average data rate for all hosts that transfered no data in the round $r-1$, i.e. $t_i(r-1) = 0$. Let $DR_0(r-1)$ be the set of hosts with $t_i(r-1) = 0$, then we can derive the expected rate for all hosts in $DR_0(r)$ as follows:

$$\overline{dr_0}(r) = \frac{1}{|DR_0(r-1)|} \sum_{h_i \in DR_0(r-1)} t_i(r).$$

Now we can set the expected data rates for these hosts: $\forall h_i \in DR_0(r) : te_i(r+1) = \overline{dr_0}(r)$.

This results in a slight overestimation of the expected traffic, but solves the problem of IDS overloading by hosts that did not transfer data in the previous round. In addition, the assignment algorithm favors hosts with lower data rates when the hosts' priority is equal, as the lower a host's data rate is, the higher the probability that it can be assigned to an IDS instance.

To balance the effects of traffic under- and overestimation, we introduced a feedback loop that dynamically adjusts a ratio for all estimated traffic data rates.

F. Maximum data rate prediction for IDS instances

The host assignment algorithm must know the maximum data rate $tmax_i$ that each IDS instance ids_i is able to process. This value may change dynamically during execution time, because of the stateful analysis of packet streams and also due to system-related aspects (e.g., the influence of external processes). We implemented an interval-based estimation algorithm that is based on the amount of dropped $d(r)$ and forwarded $f(r)$ octets at the interface to the IDS instance. The maximum IDS data rate $tmax_i(r+1)$ is calculated as follows. If $d(r) = 0$ and $f(r) > \frac{1}{2} tmax_i(r)$, we set $tmax_i(r+1) = a \times tmax_i(r)$. If $d(r) > 0$ and $f(r) < tmax_i(r)$, we set $tmax_i(r+1) = tmax_i(r)/a$. In all other cases, $tmax_i(r+1) = tmax_i(r)$. In our prototype, sensible values for $ids_i(0)$ and a were 500 KiB/round and 1.01, respectively.

G. 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. We avoided the overhead of those copy operations by implementing a lock-free ring buffer [21] of size n in shared memory. As the forwarded traffic is highly dynamic, the IDS instance is

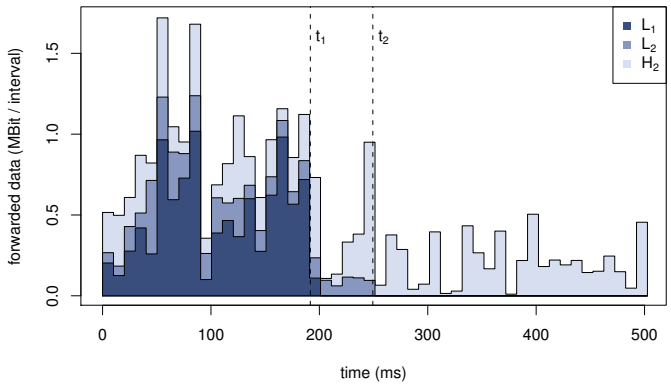

Figure 2. Runtime graph of the data rate transferred by the ring buffer to one IDS instance with a data aggregation interval of 10 ms

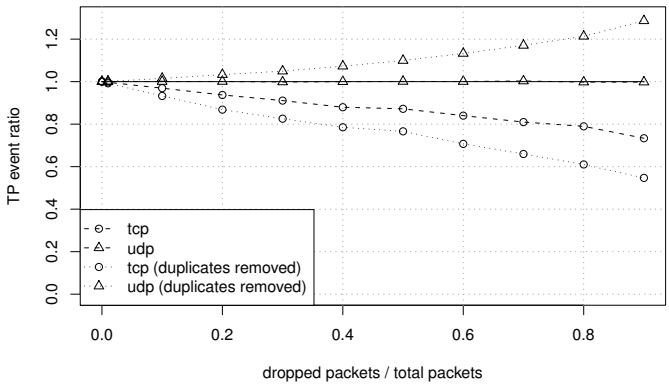

Figure 3. True-Positive event ratio of the IDS Snort working on traces with random packet loss

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. Our 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 t_1 (in our implementation this is set to $\lfloor \frac{n}{2} \rfloor$ at the start of each interval), packets will be dropped in a controlled way. Within the per-packet selection module, an ordered set S_1 is stored containing all hosts whose packets are forwarded to the IDS instance. All hosts in this set are ordered by their priorities. If the buffer's fill level exceeds the threshold, the ordered set is split into two halves $S_1 = H_1 \cup L_1$ whereby all hosts within H_1 have greater or equal priority compared to all hosts within L_1. Packets belonging to hosts within L_1 will be dropped (previously also called *forcibly dropped hosts*) from this moment on. Additionally, a new threshold t_{i+1} will be used with $t_{i+1} = n - \lfloor \frac{1}{2}(n - t_i) \rfloor$ for the next iteration, as well as $S_{i+1} = H_i$.

This behavior can be observed in Figure 2. We show the data rate forwarded to one IDS instance for one round of 500 ms. The two vertical lines mark the times when the ring buffers' thresholds t_1 and t_2 were exceeded. Each time, one group of hosts was removed from the forwarding table by controlled packet dropping. The set L_1 is removed at t_1, and the set L_2 is removed at t_2. This mechanism ensures that hosts with the highest priorities will remain in the set as long as possible. The decision which hosts to remove from the set is not based on the hosts' current data rates. As can be seen in Figure 2, multiple iterations of controlled dropping may be necessary, e.g. at t_2 the set L_2 is dropped although only little data was caused by those hosts.

IV. Evaluation

A. IDS detection performance in non-optimal conditions

Payload-based signature matching uses different preprocessing methodologies depending on the protocol type of the analyzed flow. For UDP and other non-stream-oriented protocols, packets have to be parsed separately. For streaming protocols like TCP, a stream is reassembled from the captured packets and pattern matching is performed on the reconstructed stream. To detect evasion and spoofing techniques, packets with payload are commonly only accepted if an acknowledgment has been observed from the peer. If acknowledgments are dropped due to overload at the IDS, this may lead to the loss of a large portion of TCP-payload related events.

We evaluated this effect in an experimental setup using the IDS Snort with the EmergingThreats ruleset[3], using a 4100 s network packet trace from our University's network with an average data rate of 430 Mbit/s. For each run, we prepared the packet trace and randomly dropped packets at a rate $p \in \{0.0, \ldots 0.9\}$ and recorded all reported events. We removed events from the trace that required multiple detections within a specified timespan and events requiring no content matching.

Figure 3 shows the results. The horizontal axis specifies the packet loss rate p, and the vertical axis shows the ratio $\frac{n_p}{n_t p}$, where n_p is the number of events reported during a run with packet loss p, and n_t is the total number of events without packet-loss. To show the loss of unique events within the trace, the dotted lines mark results for which we removed duplicate events for the same connection. The results confirm our assumptions and show that for stream-based connections, there is a higher probability of losing events than the overall packet-loss rate suggests, e.g. for the filtered TCP trace with a packet-loss rate of 50%, 62% of all events were not detected. In contrast to TCP, UDP-related signatures did not show any disproportionate impairment by the packet-loss.

If, at a packet loss rate p, a TCP segment with payload was received matching a signature, the probability that the succeeding acknowledgment packet will be received is also p. This leads to a probability of p^2 that both packets are successfully received by the IDS. Our observed probability is much higher than p^2. This is caused by longer TCP connections that contain multiple acknowledgments after the packet containing the matched payload. At least one acknowledgment packet needs to be received by the IDS from the peer as a cumulative acknowledgment. The probability to capture at least 1 packet

[3] http://www.emergingthreats.net, downloaded on 29/9/2010

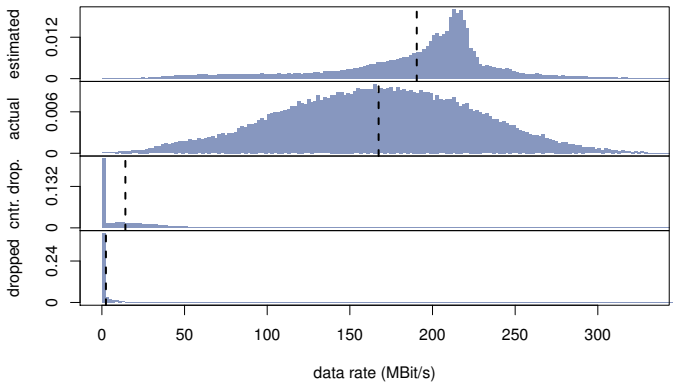

Figure 4. Multiple results as normalized histogram of the load balancer in a 24 h run. The mean is marked with a dashed vertical line

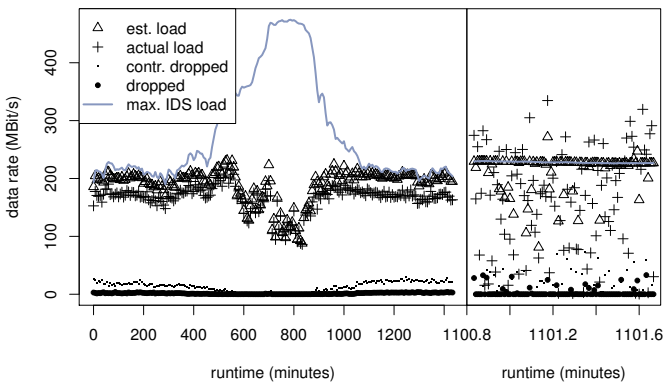

Figure 5. Runtime plot of a data transfer to one IDS instance with aggregated values over 24 h (left) and a short non-aggregated excerpt (right)

out of n is $1-(1-p)^n$. So the resulting probability of event matches is highly dependent on the connection lengths of the trace. Even for short connections with one request and one response, it can be assumed that at least two acknowledgment packets are transferred, one for the payload, and one for a successful connection shutdown using FIN flags.

B. Prototype

We implemented a prototype of the load balancing system using the monitoring framework Vermont, and customized the IDS Snort to receive packets via our shared memory-based ring buffer.

1) Runtime performance: To evaluate the performance of our load balancing system in a realistic environment, we experimented with our prototype[4] that received live data from our University's Internet uplink. Vermont was configured to capture packets from the network interface with PF_RING [22] and a 32 kpkt ring buffer. The load balancer distributed the packets to three instances of the IDS Snort and used an assignment interval of 500 ms.

We analyzed a 24 h run of the load balancing system, focusing on statistics from the first IDS instance. In this run, we observed a mean data rate of 650 MBit/s. The load balancing system dropped less than 0.1 % of packets in PF_RING's buffer. Figure 4 shows the normalized distribution of the following variables: actual load measured in one round of the IDS instance, data rate estimated in the previous round, data that was dropped by our controlled packet drop mechanism, and data that was unintentionally dropped. The estimated data rate is often directly related to the estimation of the IDS instance's maximum data rate, because the load balancing algorithm tries to use up as much available data rate as possible. The actual received load from the network shows a very high variance with a mean of 167 MBit/s. Controlled packet drop was performed at a mean data rate of 14 MBit/s, and effectively prevented most uncontrolled packet drops with a mean data rate of 2.7 MBit/s. Ensuring a lower packet drop rate would either require larger buffers, or a lower utilization of the IDS instances. Both aspects lead to compromises that need to be considered: larger buffers would cause a higher amount of cache misses within the CPU cores, resulting in slower execution, and a lower utilization of the IDS instances would reduce the amount of analyzed network traffic.

A runtime plot of the same run is shown in Figure 5. The left graph shows a timespan of 24 h, and each plotted value is a mean of 200 s, whereas the right graph shows an excerpt of non-aggregated values. The figure additionally includes the maximum data rate estimated for the IDS instance by our algorithm. During the night at runtime 450 min to 1000 min, the load balancing system was able to process almost all incoming traffic. Thus, the utilization of the IDS instances decreased and less packets were dropped, which, in turn, leads the effect that the load estimation algorithm increased the estimated maximum load. This wrong estimation has no serious consequences, as the algorithm adapts very quickly to load changes. This cannot be seen in the runtime plot, as it displays the mean value over 400 single data points, whereas the non-aggregated values oscillated. We included an excerpt on the right side of Figure 5 to visualize one of the key problems: the high variance of the received traffic's data rate. Obviously, the estimated load of the IDS instance varies with time.

2) Event detection performance: To compare our load balanced parallel IDS with a normal IDS in conditions with packet-loss, we conducted another experiment using the same packet trace described in Section IV-A. In Figure 6, we show the number of TCP and UDP related events reported by Snort for processing the trace with different packet drop ratios, whereby duplicate events for the same connection were omitted. Additionally, we performed multiple runs with our load balancing algorithm configured with $mt_{min} = 10$ s. To obtain realistic results, we processed the packet trace 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. At a packet loss rate of 85%, the load balanced IDSs run detected 44 % more events compared to the IDS run with random packet drops.

[4]Intel Core2 Quad CPU@2.83 GHz; Intel 82572EI Ethernet controller

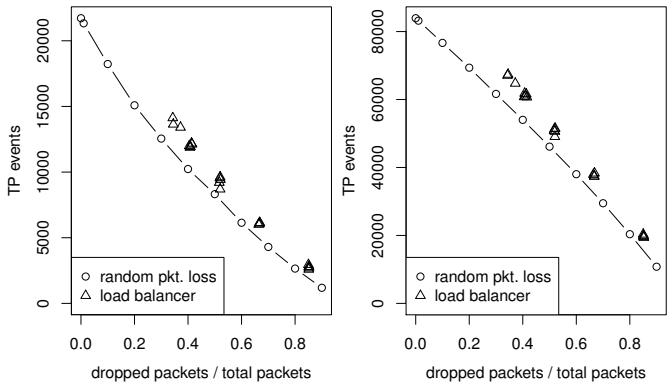

Figure 6. Number of detected events with random packet losses compared to the load balancing system. Left/right graph: TCP/UDP-related events

This is due to the minimal monitoring interval, which ensures that all consecutive packets in a certain interval are forwarded to the IDS. 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.

V. Conclusion

We introduced a novel approach for load balancing specialized for parallel payload-based IDSs on multi-core systems using prioritized flows. We primarily target the use in high-speed networks where it is only feasible to analyze a fraction of the network traffic. Using a prototype, we experimentally evaluated the performance of our system both in quality and quantity. The system is able to cope with highly fluctuating data rates, which is a common property of our 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. This capability leads to improvements of up to 44 % in the event detection rate compared to systems dealing with random packet loss. Using our prototype, we performed multiple experiments on real network traces that confirmed the expected throughput and detection ratios.

References

[1] W. John, S. Tafvelin, and T. Olovsson, "Passive internet measurement: Overview and guidelines based on experiences," *Computer Communications*, vol. 33, no. 5, pp. 533–550, March 2010.

[2] H. Dreger, A. Feldmann, V. Paxson, and R. Sommer, "Operational experiences with high-volume network intrusion detection," in *11th ACM Conference on Computer and Communications Security (ACM CCS 2004)*. Washington, DC: ACM, October 2004, pp. 2–11.

[3] 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.

[4] M. Colajanni and M. Marchetti, "A Parallel Architecture for Stateful Intrusion Detection in High Traffic Networks," in *IEEE/IST Workshop on Monitoring, attack detection and mitigation (MonAM 2006)*, Tuebingen, Germany, September 2006.

[5] P. Wheeler and E. W. Fulp, "A Taxonomy of Parallel Techniques for Intrusion Detection," in *45th Annual Southeast Regional Conference (ACMSE 2007)*, D. John and S. N. Kerr, Eds. Winston-Salem, NC: ACM, March 2007, pp. 278–282.

[6] 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.

[7] M. Andreolini, S. Casolari, M. Colajanni, and M. Marchetti, "Dynamic load balancing for network intrusion detection systems based on distributed architectures," in *6th IEEE International Symposium on Network Computing and Applications (NCA 2007)*. Cambridge, MA: IEEE, July 2007, pp. 153–160.

[8] T. Limmer and F. Dressler, "Flow-based Front Payload Aggregation," in *34th IEEE Conference on Local Computer Networks (LCN 2009): 4th IEEE LCN Workshop on Network Measurements (WNM 2009)*. Zurich, Switzerland: IEEE, October 2009, pp. 1102–1109.

[9] ——, "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, to appear.

[10] 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, October 2005, pp. 267–272.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] A. Le, R. Boutaba, and E. Al-Shaer, "Correlation-based Load Balancing for Network Intrusion Detection and Prevention Systems," in *4th International Conference on Security and Privacy in Communication Networks (SecureComm 2008)*, ser. SecureComm 2008. Istanbul, Turkey: ACM, September 2008, pp. 1–10.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.