IEEE TRANSACTIONS ON
EMERGING TOPICS
IN COMPUTING

Received 2 September 2014; revised 7 November 2014; accepted 8 November 2014. Date of publication 4 December 2014;
date of current version 26 February 2016.

Digital Object Identifier 10.1109/TETC.2014.2369958

LEoNIDS: A Low-Latency and Energy-Efficient Network-Level Intrusion Detection System

NIKOS TSIKOUDIS[1], ANTONIS PAPADOGIANNAKIS[2], AND EVANGELOS P. MARKATOS[2]

[1]Brandeis University, Waltham, MA 02453, USA
[2]Institute of Computer Science, Foundation for Research and Technology—Hellas, Heraklion 700 13, Greece
CORRESPONDING AUTHOR: N. TSIKOUDIS (tsikudis@brandeis.edu)

This work was supported in part by the project GCC funded by the Prevention of and Fight against Crime Programme of the European Commission—Directorate-General Home Affairs under Grant Agreement HOME/2011/ISEC/AG/INT/4000002166 and in part by the FP7 Project SysSec through the European Commission under Grant 257007. This work was conducted while N. Tsikoudis was with FORTH-ICS. This publication reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

ABSTRACT Over the past decade, design and implementation of low-power systems has received significant attention. While it started with data centers and battery-operated mobile devices, it has recently branched to core network devices such as routers. However, this emerging need for low-power system design has not been studied for security systems, which are becoming increasingly important today. Toward this direction, we aim to reduce the power consumption of network-level intrusion detection systems (NIDS), which are used to improve the secure operation of modern computer networks. Unfortunately, traditional approaches to low-power system design, such as frequency scaling, lead to a disproportionate increase in packet processing and queuing times. In this paper, we show that this increase has a negative impact on the detection latency and impedes a timely reaction. To address this issue, we present a low-latency and energy-efficient NIDS (LEoNIDS): an architecture that resolves the energy-latency tradeoff by providing both low power consumption and low detection latency at the same time. The key idea is to identify the packets that are more likely to carry an attack and give them higher priority so as to achieve low attack detection latency. Our results indicate that LEoNIDS consumes power comparable to a state-of-the-art low-power design, while, at the same time, achieving up to an order of magnitude faster attack detection.

INDEX TERMS Intrusion detection systems, network security, energy-efficient systems, low-power design, low latency, performance, multi-core packet processing.

I. INTRODUCTION

Low power consumption has emerged as one of the main design goals in today's computer systems. Recently, much effort has been put into improving the energy efficiency in a variety of areas like data centers [1], high performance computing [2], mobile devices [3], and networks [4]. Towards this direction, we aim to build an energy-efficient Network-level Intrusion Detection System (NIDS). NIDS are commonly deployed to detect security violations, enhancing the secure operation of modern computer networks. They perform computationally heavy operations like pattern matching, regular expression matching, and other types of complex analysis to detect at real time malicious activities in the monitored network. Thus, NIDS usually utilize multi-core systems [5] or cluster of servers [6], [7] to cope with increased link speeds and complicated analysis. However, the energy efficiency of security systems like NIDS has not received significant attention and has not been studied before. In this work, we study this emerging need for low-power system design and improved energy efficiency focusing on NIDS, which are among the most commonly deployed systems for cyber security.

Although NIDS are usually provisioned to operate at link rate, in order to be able to handle a fully utilized network, most networks are typically much less utilized. This results in increased power consumption at low traffic load. To reduce the energy spent under low traffic we aim at building a power-proportional NIDS using Dynamic Voltage and Frequency Scaling (DVFS) and sleep states (C-states), which can be found in modern processors. The system should consume the less power needed to sustain the incoming traffic load. We found that a NIDS consumes less power when it uses

the smallest number of cores that can operate at the lowest possible frequency to process the network traffic, by keeping these cores nearly fully utilized. This energy-efficient NIDS can process all packets with up to 23% lower power consumption than the original system at low rates. However, we observe a significant increase on the detection latency due to higher processing times when reducing the frequency, and mostly due to increased queuing delays imposed by the high utilization.

A low detection latency is very important to ensure a timely reaction to the attack. Upon the detection of a packet that carries an attack, the NIDS can actively terminate the offending connection or install a new firewall rule. This reaction should be immediate, before the attack packets reach the victim's machine and the attack succeeds. Therefore, our results indicate a new tradeoff for NIDS: the *energy-latency tradeoff*. Our key idea to resolve this tradeoff is to identify the most important packets for attack detection and process them with higher priority, resulting in low latency and fast detection. The rest packets are processed with lower priority to achieve an overall low power consumption.

We explore two alternative approaches to reduce the latency of high-priority packets: *time sharing* and *space sharing*. In time sharing we use a typical priority queue scheduling in each core. In space sharing the high-priority packets follow a different path, using dedicated cores with much lower utilization to achieve low latency. To implement space sharing we use features of modern network interface cards (NIC) to move efficiently the processing of least-significant packets to cores with higher utilization, a technique we call as *flow migration*. We experimentally compare the two approaches and we find that space sharing has a better power-latency ratio.

Based on these approaches we propose LEoNIDS: a NIDS architecture that resolves the energy-latency tradeoff. The implementation of LEoNIDS uses NIC features, a specialized kernel module, a modified user-level library, and it is based on the popular Snort NIDS [8]. LEoNIDS consumes less power, proportionally to the traffic load, while its detection latency remains low and almost constant at any traffic load.

The main contributions of this work are:

- We identify a new tradeoff for NIDS: the *energy-latency tradeoff*. As we reduce power consumption, the detection latency is significantly increased, which impedes a timely reaction to incoming attacks. We found that the main cause of this increase is the queuing delays imposed by the high core utilization.
- We resolve the energy-latency tradeoff by identifying the packets that have a higher probability to contain an attack and processing them with higher priority.
- We introduce *space sharing*: a new technique based on flow migration that processes high-priority packets in dedicated cores with low utilization, and moves the low-priority packets to cores with higher utilization.
- We experimentally compare two alternative approaches for low latency in a power-proportional NIDS. We show that space sharing results in lower detection latency when power consumption is reduced.
- We present the design, implementation, and evaluation of LEoNIDS, a NIDS architecture that achieves both low latency and reduced power consumption.

II. MOTIVATION

We first explain why a low response time is crucial for a NIDS, and then we argue for the usefulness of an energy-efficient NIDS.

A. WHY DETECTION LATENCY MATTERS

Although a NIDS operating in passive mode does not affect the actual latency of the monitored packets, a fast attack detection is necessary. This is because a NIDS is able to react and protect the potential victims upon a timely attack detection, without the need of human advisory. One way to achieve this is to actively terminate an offending TCP connection by sending TCP reset packets with the correct sequence numbers and spoofed IP addresses of victim and attacker hosts, *e.g.*, using Snort's active response [9]. Since a reset packet may reach the client or server after the other host has already responded, the NIDS tries to close the connection by sending multiple reset packets and guessing the next TCP sequence and acknowledgment numbers. However, such an active response is not guaranteed to successfully terminate an offending connection: it is a race between the NIDS and the endpoints of the network communication. Depending on the detection latency and the network latency, NIDS may or may not win this race. Thus, a NIDS should be able to detect the incoming attacks very fast, especially in order to stop fast TCP connections. For instance, in case of an 100 Mbit/sec connection, assuming an average packet size of 500 bytes, an attack packet should be detected in less than 40 microseconds upon its arrival for an effective active response by the NIDS.

Another possible reaction of a NIDS is to automatically add a firewall rule to block the next incoming attack packets in order to prevent a full system compromise. To be effective, a low detection latency is again crucial. Moreover, DNS and URL blacklists may also be updated upon the detection of a malicious domain or malicious URL to protect the other hosts from accessing it. Since a malicious website may be accessed within short time periods by many users, *e.g.*, due to massive spam messages, it is important to automatically update these blacklists in a timely fashion.

B. WHY POWER CONSUMPTION MATTERS

NIDS are usually overprovisioned to handle a fully utilized line and tolerate overloads without missed attacks [10]. Thus, they use all the available resources: all cores [5], and the maximum CPU frequency. In high speed networks, the traffic load may also be split among multiple machines [6], [7]. However, the monitored networks are rarely fully utilized at their maximum capacity and a NIDS machine is not often overloaded. This results in increased energy and increased cost for running multiple NIDS to protect a large infrastructure.

Power consumption is a significant concern in data center environments with limited power capacity. Moreover, it is important in devices with limited resources, such as small routers or wireless access points. The power consumption is even more important when NIDS run on devices with limited battery life, such as sensor nodes or mobile devices. For instance, mobile devices may run a host-based NIDS to protect their users. We believe that our work can be applied in NIDS that run in such devices as well.

Reducing the power consumption of network security solutions like NIDS is important to reduce the operational costs of these security products, making them more attractive to use than other alternative approaches. Reducing the power cost of a NIDS may not be very important for a data center compared to its total power consumption, but it is quite important for NIDS vendors to provide more cost-effective solutions. Network security products with reduced power consumption will give one more benefit in the market, while network security services based on NIDS, which are rapidly developed today using cloud infrastructures, will offer lower prices when consuming less energy. Given the recent advances in hardware and computer architecture, with more powerful hardware components and increasing number of CPU cores, building energy-efficient systems and applications becomes an important performance indicator.

III. TOWARDS A POWER PROPORTIONAL NIDS

In this section we explore the design space to build a power-proportional NIDS.

A. EXPERIMENTAL ENVIRONMENT

Our testbed consists of two machines interconnected with a 10 GbE switch. Both machines are equipped with two six-core Intel Xeon E5-2620 processors with 15 MB L2 cache, 8 GB RAM, and an Intel 82599EB 10 GbE network interface. The clock frequency can be scaled from 1.2 GHz to 2.0 GHz using DVFS, which results in 9 available frequency steps. They also support Intel Turbo Boost technology to increase their frequency up to 2.5 GHz. To reduce power consumption, each idle core can be put into one of the 3 available sleep states: C1, C3 or C6, where the CPU reduces or stops the performance of internal units. We measure the power consumption in the NIDS machine using the Watts up? PRO ES device.

The first machine is used for traffic generation. The generated traffic reaches the second machine, which runs Snort IDS [8] v2.8.3.2 with official rule set [11] containing 8308 rules. We use PF_RING [12] v5.3.0 and ixgbe driver v3.7.17 to split the incoming traffic to active cores using the Receive Side Scaling (RSS) [13] feature of Intel 82599 NIC [14]. We set the size of the ring buffer that stores packets at each core to 4096 slots. To change the frequency we use the *cpufrequtils* package. Both machines run 64-bit Linux (kernel version 3.5.0).

We generate real traffic by replaying an one-hour long anonymized trace captured at the access link of an operational network. The trace contains 58,714,906 packets and 1,493,032 flows, totaling more than 40GB, 95.4% of which is TCP traffic. For this trace Snort triggers 1851 alerts from 76 different rules. Most of the matching rules are related to common threats and protocol violations. In order to strengthen our evaluation, we augmented the trace with 120 anonymized traces of real attacks captured in the wild [15], adding 233 more alerts from 14 different rules. In this work we present our findings using this trace as workload, which we believe is representative for a typical network. We found quite similar results when using few different workloads based on anonymized packet traces from other sources.

B. POWER CONSUMPTION

The system's idle power consumption is 85.1 W, and when Snort fully utilizes all cores it consumes 145.7 W. As NIDS perform heavy computational operations, the CPU consumes the larger portion of energy in the system. Table 1 shows the contribution of the CPU in the total power consumption when running Snort. We measure the CPU power consumption by accessing the RAPL (Running Average Power Limit) registers provided by each Intel Xeon E5-2620 CPU. In all NIDS utilizations, 58-62% of the total power is consumed by CPUs.

TABLE 1. The CPU consumes the larger portion of energy in a NIDS. More than 50% of the power is consumed by the CPU.

NIDS utilization	Total power consumption	CPU power consumption
Low	109.9 W	63.5 W
Moderate	129.6 W	78.0 W
High	145.7 W	90.6 W

Modern processors offer two ways to reduce power consumption: frequency scaling (DVFS), and sleep states (C-states). Intel processors have a single voltage and frequency regulator, so the frequency changes uniformly at all cores of a processor and the transition between the frequencies requires small time. However, each core can operate in a different C-state to save energy. The power consumption of each core consists of (i) power consumed when the core processes packets, (ii) power consumed to enter a C-state, and (iii) power consumed during the idle state. We see that idle cores consume less power in C6 state, so we put inactive cores in this state.

Based on the packet arrival rate, we aim to find the most energy-efficient strategy for a NIDS by properly adapting the frequency and the number of active cores (not in C-states). The two main questions are: (i) is it better to operate at lower frequency or utilize sleep states? (ii) is it better to use more cores on lower frequency or fewer cores at higher frequency?

To find the optimal strategy we measure Snort's power consumption as a function of frequency and number of active cores, when sending traffic at a constant rate of 0.6 Gbit/sec. Figure 1(a) shows that the lowest power consumption is achieved when using 4 cores at 1.2 GHz, which is the

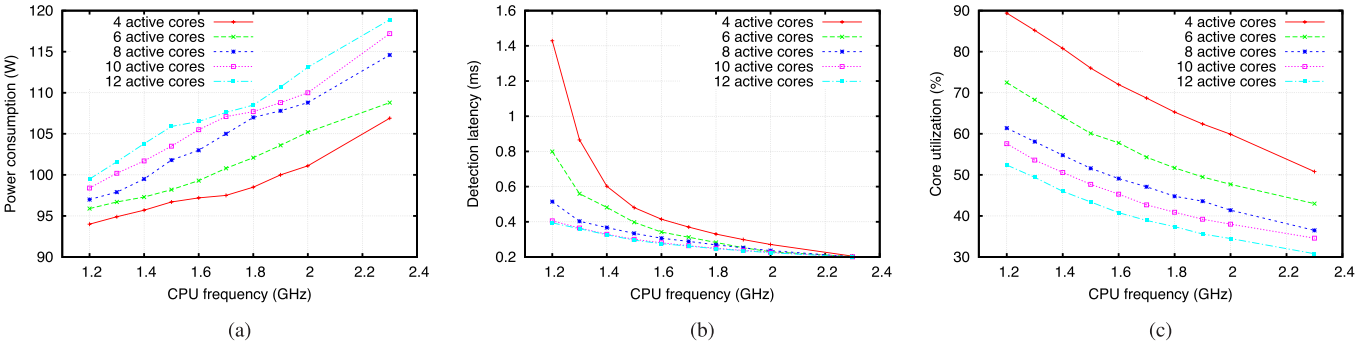

FIGURE 1. Fewer cores and lower frequency reduce the power consumption but increase the detection latency. Power consumption, detection latency, and core utilization as a function of frequency and number of active cores when running Snort and sending 0.6 Gbit/sec. We see that power consumption decreases as the utilization of active cores approaches 100%. However, this results in increased detection latency. (a) Power consumption. (b) Detection latency. (c) Utilization per active core.

minimum setup able to handle the 0.6 Gbit/sec traffic with no packet loss. In this setup we see up to 21% reduced power consumption compared to 12 cores at the maximum frequency.

We observe that the less power is consumed when the system operates at the lowest possible frequency with no idle time, instead of running at higher frequencies and entering C-states during idle periods. Moreover, we see that using more cores at lower frequency is more energy efficient than using fewer cores at higher frequencies. For instance, Table 2 shows three alternative setups that can be used to process 1.5 Gbit/sec, as they offer approximately the same computing power. We see that 10 cores at 1.2 GHz consume the less power. Figure 1(c) shows the average utilization of active cores. We see that power consumption decreases as the core utilization increases and approaches 100%. This is because being idle is not sufficiently efficient, *i.e.*, the power consumed to enter and leave C-states and during these idle periods is quite significant.

TABLE 2. Using more cores at lower frequency consumes less power but results in higher detection latency (when processing 1.5 Gbit/sec).

Active cores	CPU Frequency	Power consumption	Detection latency
6	2.0 GHz	107.0 W	0.371 ms
8	1.5 GHz	104.2 W	0.856 ms
10	1.2 GHz	100.2 W	1.228 ms

C. ADAPT TO THE TRAFFIC LOAD

Our results indicate that a power-proportional NIDS should utilize the smallest number of cores that are able to process all the incoming traffic when they operate at the lowest possible frequency. Therefore, the system should dynamically adapt to the load by changing the frequency and activating/deactivating cores. We observe that it is preferable to first activate cores of the same CPU, which explains the larger distance between 6 and 8 cores in Figure 1(a). The NIDS should be also able to handle short-term overloads by spending more energy during these periods to avoid dropping packets, which results in undetected attacks [16].

A NIDS is based on the underlying packet capturing system to receive packets for processing. To tolerate processing spikes or short-term overloads, the packet capturing system is able to store a limited number of packets in memory queues. Modern NICs [14] offer multiple receive queues and distribute the packets among them to allow for efficient multi-core processing [12]. When queues are getting full, the system has a strong indication of higher load than it can handle with the current setup, so it needs to employ more cores or increase the frequency. A straight-forward power-proportional NIDS uses the following strategy:

1. It starts with a single active core at the minimum frequency.
2. It continuously monitors the queues' usage.
   - 2.1. If queues are filled by more than a *high threshold*:
     - 2.1.1. If there are inactive cores, it wakes up one core.
     - 2.1.2. Else, it increases the frequency of all cores.
   - 2.2. If queues are filled by less than a *low threshold*:
     - 2.2.2. If the lowest frequency is used, it deactivates one core.
     - 2.2.2. Else, it decreases the frequency.

We implemented this online adaptation algorithm within the packet capturing subsystem and we ran Snort over this system while varying the load. We set *high threshold* to 90% and *low threshold* to 70%. Figure 2 (bottom part) shows the power consumption of this straight-forward energy-efficient NIDS as a function of the traffic rate, compared to the original system. We see that the vanilla system consumes 24% less power when processing 0.2 Gbit/sec, compared with the power consumption at 3 Gbit/sec, which is the maximum rate without packet loss. Contrary, the power-proportional NIDS adapts much better to the load reducing the power consumption by 39% when processing 0.2 Gbit/sec. This is a 23% improvement on the power consumption compared to the original system.

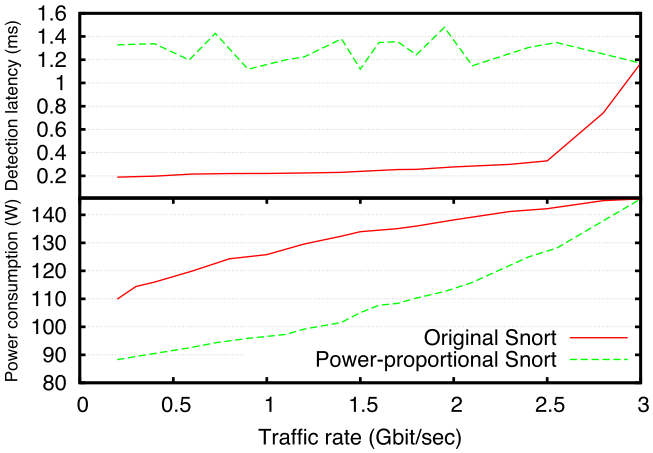

FIGURE 2. Power consumption and detection latency of a straight-forward power-proportional NIDS versus the original NIDS as a function of traffic rate. The straight-forward power-proportional NIDS consumes less power with higher detection latency.

IV. THE ENERGY-LATENCY TRADEOFF IN NIDS

Although a power-proportional NIDS is able to handle the same traffic as the original system with lower energy, we would like to explore the impact of this approach on the detection latency.

A. DETECTION LATENCY

We instrumented Snort to measure the attack detection latency, by subtracting from the time that an alert is triggered the timestamp of the packet that contains the attack. The packet's timestamp is set within the packet capturing module before the packet is queued. Figure 1(b) shows the detection latency as a function of frequency and number of active cores for 0.6 Gbit/sec. We see a linear increase when frequency is reduced until 1.6 GHz, and when up to 8 cores are used, but we see a significant increase to the detection latency when core utilization exceeds 70%. To better see the relation between power consumption and detection latency we replot these data in Figure 3. We see a clear tradeoff: to achieve power consumption lower than 100 W, the detection latency has to be increased 2–7 times. Our experimental evaluation, at the end of Section VII, shows that such an increase in the detection latency can significantly impede the effectiveness of an automated reaction of a NIDS to an incoming attack.

Table 2 leads us to the same outcome: although using 10 cores at 1.2 GHz consumes the less power, it comes at a price of significantly increased latency. Figure 2 (upper part) shows the detection latency of a power-proportional NIDS, compared to the original system. We see that although it consumes less power, it has a significantly higher detection latency at all rates. This is because it always selects the frequency and number of cores that lead to high utilization, close to 100%, in order to save energy.

B. DECONSTRUCTING DETECTION LATENCY

We define detection latency as the time passed from the arrival of the last packet that contains the attack until the alert generation in the NIDS. Thus, detection latency is equal to the latency imposed per each attack packet, from capturing time until it finishes processing. The packet latency can be divided in three parts: (i) *interrupt handling time*, *i.e.*, the time spent for packet handling in OS kernel, (ii) *queuing delay*, *i.e.*, the time that packet waits in a queue to be delivered for processing, and (iii) *processing time* by the NIDS at user level. We see that the interrupt handling time is negligible compared to queuing delay and NIDS processing time. Thus, the increased detection latency may occur due to higher processing times when reducing the frequency or due to higher queuing delays imposed by the increased utilization.

To explore why detection latency is increased, we measure how much each part contribute to the detection latency as we vary the offered traffic rate for different frequencies. We instrumented Snort to measure the queuing delay per packet, by subtracting the packet's timestamp from the time that packet is received in Snort for processing, and the packet's processing time in Snort. Figure 4 shows the average processing time and queuing delay per each attack packet for traffic rates ranging from 0.5 Gbit/sec to 1.5 Gbit/sec, when using 12 cores in 1.2 GHz, 1.8 GHz, and 2.3 GHz.

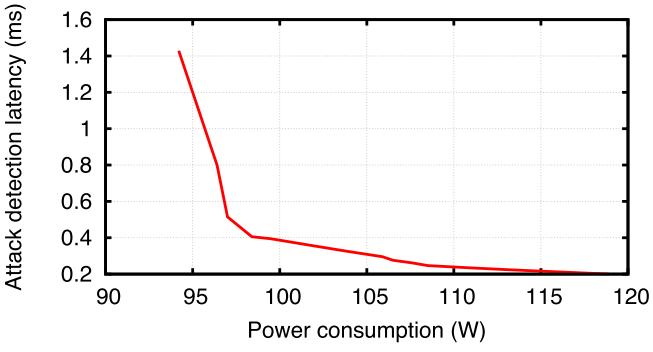

FIGURE 3. The energy-latency tradeoff. Detection latency as a function of power consumption when sending 0.6 Gbit/sec traffic. We see that detection latency increases as power consumption is reduced.

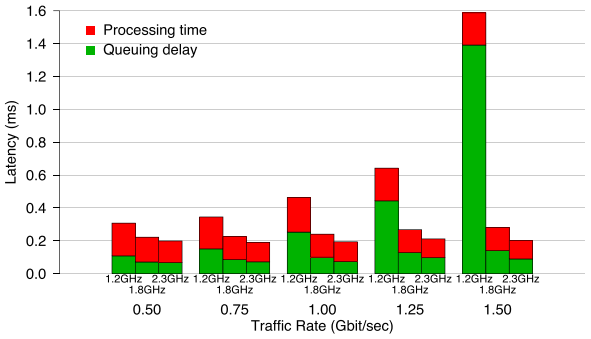

FIGURE 4. The main cause of increased detection latency is higher queuing delay. We see that for low frequency and high rates, queuing delay is much higher than processing time.

In low frequency and high rates, when the system is more utilized, queuing delay is the main factor of the increased detection latency. For instance, when processing 1.5 Gbit/sec at 1.2 GHz, queuing delay is 7 times higher than processing time. This is because the higher utilization results in a large number of packets waiting at each queue and thus in a higher queuing delay. In contrast, processing time increases linearly as we decrease frequency.

C. DELAY ANALYSIS

To better understand how queuing delay is affected by reduced frequency and number of cores, we present a theoretical formulation of the problem using basic concepts from queuing theory. We assume that packet arrivals follow a Poisson distribution with an average rate of λ packets per second, and that the queued packets are processed with an exponential service rate of μ packets per second. The maximum service rate is μ_{max} when all the c_{max} cores are used at the maximum frequency f_{max}. Packets arrive at each core with rate $\lambda_c = \lambda/c$, where c is the number of active cores, and they are served from each core with rate $\mu_c = \mu/c_{max}$. Each core can be modeled as a $M/M/1$ queue with a finite queue size of N packets. The core's utilization is $\rho_c = \lambda_c/\mu_c$. Reducing the number of cores c that process incoming packets results in an increased arrival rate λ_c per core. Reducing the frequency f results in reduced service rate per core:

$$\mu_c(f) = \frac{\mu_{max}}{c_{max}} \frac{f}{f_{max}} \tag{1}$$

The total delay T of a packet, which is queuing delay plus processing time, when using c cores and frequency f is:

$$T = \frac{1}{\mu_c(f) - \lambda_c} = \frac{c_{max} \cdot f_{max} \cdot c}{\mu_{max} \cdot f \cdot c - \lambda \cdot c_{max} \cdot f_{max}} \tag{2}$$

Respectively, the average queuing delay W is:

$$W = \frac{\rho_c}{\mu_c(f) - \lambda_c} = \frac{\lambda \cdot c_{max}^2 \cdot f_{max}^2}{\mu_{max} \cdot f \cdot (\mu_{max} \cdot f \cdot c - \lambda \cdot f_{max} \cdot c_{max})} \tag{3}$$

The packets loss probability is:

$$P_{loss} = \sum_{k=N}^{\infty} \rho_C^k (1 - \rho_c) \tag{4}$$

Based on equation 2, Figure 5 plots the total delay T as a function of frequency f and number of cores c. We set $f_{max} = 2.5$ GHz, $c_{max} = 12$ cores, $\lambda = 75,000$ packets per second, and $\mu_{max} = 500,000$ packets per second. We see that the total delay approximates the behavior of detection latency: when using 4 cores to process 75,000 packets per second, the delay increases exponentially as we decrease frequency. When using more cores, we see a linear increase on packet's total delay.

Our results in section III-B indicate that we have the lower power consumption when $\rho_c \approx 1$. Figure 6 shows the total delay T and queuing delay W as a function of utilization ρ. We keep $\mu_{max} = 500,000$ packets per second and we vary

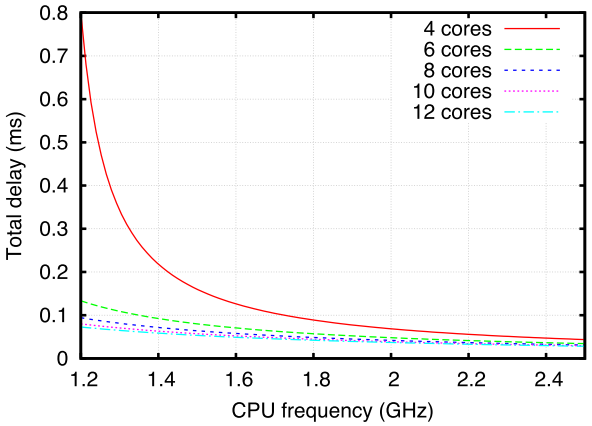

FIGURE 5. The total delay approximates the behavior of detection latency. When using 4 cores to process 75,000 packets per second the delay increases exponentially as we decrease frequency.

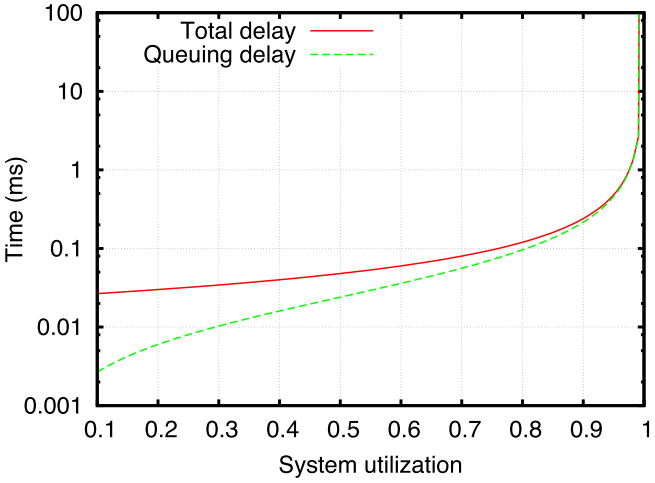

FIGURE 6. The queuing delay significantly increases with high utilization. We see that when utilization exceeds 80%, the queuing delay approaches the total delay and they both increase significantly.

λ from 50, 000 till 500, 000, which results in system utilization from 10% to 100% respectively. We use $c = 12$ and $f = 2.5$ GHz in equations 2 and 3. We see that when utilization is increased above 80%, the queuing delay approaches the total delay and both increase significantly. It is clear that under high system utilization, the queuing delay becomes the main reason for the increased total delay, as we also observe in section IV-B. To achieve a total delay lower than 0.1 ms we need to sustain a system utilization lower than 75%. If we want to reduce the total delay even more, *e.g.*, less than 50 μs in this case, the utilization should be kept lower than 50%. However, this system underutilization reduces its energy-efficiency.

V. SOLVING THE ENERGY-LATENCY TRADEOFF

We now explain our approach to resolve the energy-latency tradeoff in NIDS.

A. IDENTIFY THE MOST IMPORTANT PACKETS FOR DETECTION LATENCY

One way to address the energy-latency tradeoff in NIDS would be to keep the utilization of active cores within a specific range, so to keep power consumption and detection latency lower than the respective thresholds. However, in this way a NIDS cannot achieve the lowest possible power consumption, while detection latency may also be much higher. To efficiently resolve the tradeoff, we use domain-specific knowledge about NIDS: we capitalize on the fact that not all packets have the same probability to carry an attack. By identifying the most interesting packets, a NIDS is able to process them with higher priority to achieve fast detection, while efficiently reducing the power consumption at each traffic load at the same time.

A key abstraction we use to identify the most important packets for attack detection is the network *flow*: a flow is defined as the set of packets belonging to the same one-way connection, *i.e.*, packets with same protocol, source and destination IP addresses and port numbers (5-tuple). Previous works have shown that most attacks are found among the first few bytes of each flow [17]–[19]. This is because many types of threats like port scanning, service probes and OS fingerprinting, code-injection attacks, and brute force login attempts, require a new connection for each attempt, and the attack vector is found in the first few bytes of the flow. In contrast, very large streams usually correspond to file transfers, VoIP communication, or streaming media applications, which typically are not related to security threats. Due to the heavy-tailed flow size distribution in the Internet [20], the first bytes of each flow correspond to a very small percentage of the total traffic. Thus, processing the respective packets with higher priority will result in faster detection for most attacks.

To validate and analyze our choice for high-priority packets in a NIDS, we measure the position of each attack within its flow, for attacks detected while running Snort with our anonymized trace. As we explained in Section III-A, we injected 233 real attacks into the trace (labeled attacks), while the background traffic contains 1851 more attacks. Most of these attacks are related to popular threats and protocol violations. Table 3 presents a classification of these attacks based on Snort's ruleset [11].

Figure 7 shows the CDF of the detected attacks' position within their flows. We see that 50% of the attacks are found within the first 2 KB of a flow, while 90% of the attacks are detected in the first 30 KB of their flows. Only 2% of the attacks are found beyond the first 200 KB. We observed that the labeled attacks, which we consider more important as they correspond to real attacks and have been validated as true positives, are always detected within the first 5 KB of their flows. We found that the small percentage of attacks detected beyond 100 KB of a flow correspond to less significant threats and are usually triggered by threshold-based rules. Thus, the first few bytes of each flow are more important for the detection latency, since they have a much higher probability to actually contain an attack. We can separate the respective packets by applying a cutoff value to the flow size. Then we classify as high priority the packets until this cutoff. and as low priority the rest packets.

TABLE 3. **Classification of the attacks detected in our trace.**

Attack category	Labeled attacks	Background traffic attacks
Web attacks	14	635
Specific threats	15	282
IMAP & SMTP	1	172
NetBIOS & RPC	200	236
Attack responses	0	213
SQL & MySQL	0	188
Spyware, backdoor, misc	3	125

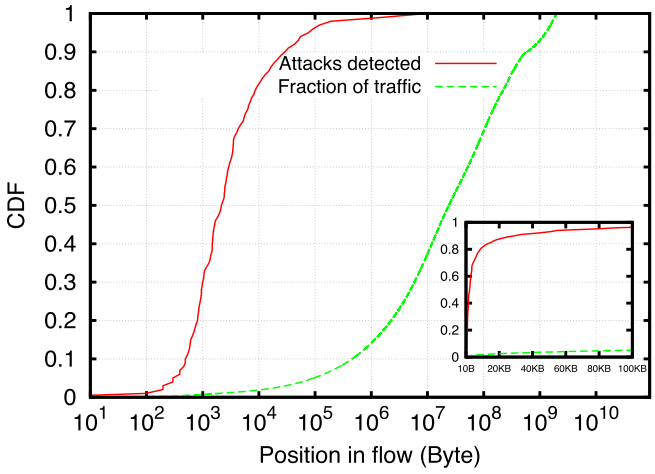

FIGURE 7. **Most attacks are detected within the first few KBs of a flow, which is a small fraction of the total traffic. CDF of attack's position at each flow and the respective fraction of traffic.**

Figure 7 also presents the CDF of the fraction of traffic that is located in a flow before the corresponding position on the x-axis. This fraction is the percentage of high-priority traffic as a function of the cutoff applied. For instance, 10% of the total traffic is found in the first 500 KB of the flows. This means that a cutoff value of 500 KB per flow will classify 10% of the total traffic as high-priority, and 99% of the attacks can be detected on this high-priority traffic.

B. TOLERATING EVASION ATTEMPTS

An attacker could try to exploit the flow cutoff mechanism used for priority assignment in order to increase detection latency and impede a timely reaction. Thus, we aim to protect LEoNIDS against such attacks. One way to exploit the cutoff mechanism would be to overburden the system with high-priority packets, *e.g.*, by sending a large number of small flows. However, as we explain in the following sections, LEoNIDS properly adapts to the traffic load by increasing frequency and active cores so that the latency of high-priority packets remains always low. In the worst case, *e.g.*, a fully utilized system only with high-priority packets,

LEoNIDS will approach the behavior of the original system: it may spend the maximum available power to keep latency of high-priority packets low.

Another way for an attacker to exploit our cutoff-based approach would be to push the attack into low-priority packets, resulting in higher detection latency. To address this attack, we take a number of countermeasures. We define flow cutoff in bytes, not in packets, so an attacker cannot exceed it by sending small packets. Also, the flow size we track should correspond to the actual size of each connection, excluding, for example, TCP retransmissions or overlapping TCP sequence numbers, which could otherwise be used to evade our technique and impose a lower priority. To handle persistent connections, like HTTP keep-alive connections, we reset flow size to zero for each new request or response. Instead of detecting each new HTTP request using DPI and HTTP protocol parsing, which is a costly operation, we use an optimization based on following TCP sequence numbers on each direction to detect a new HTTP request when we see a change in the data transfer direction, as described in [18]. Finally, we use a lower limit for the flow cutoff value. This is because most protocol implementations have a maximum protocol message (request/response) and headers size, and may close connections exceeding the size. Thus, putting the attack beyond this size is not always possible.

For instance, many of the attacks in our trace are detected at the HTTP protocol, usually based on a signature matching in URI or request headers. Although attackers can send an arbitrary large URI to exceed cutoff, *e.g.*, by adding KBs of space characters before URI, all Web servers have a *maximum URI size* configuration option. When a URI exceeds this limit, an *HTTP/1.1 414 Request-URI Too Large* error is returned, and request is not processed. Hence, the actual attack cannot succeed. Similarly, when the *maximum request size* limit is exceeded, servers respond with *HTTP/1.1 400 Bad Request (Header Field Too Long)*.

In most Web servers, the default maximum URI size is 8 KB. Thus, using a cutoff larger than 8 KB ensures the timely detection of all *successful* attacks against servers using this limit. To find out how many of the popular Web servers use this default limit, we sent a request with URI slightly larger than 8 KB to the top-100 Web sites based on the ranking of *alexa.com*. The 98 of them responded with an error, while only two of them accepted the request. When sending requests with 100 KB long URI, all the top-100 Web sites responded with error. Similarly, other protocols (*e.g.*, IMAP, SMTP, NetBIOS) have also a maximum message size. Even if it is equal to few MBs, the fraction of high-priority traffic remains low. Another reason for setting a lower limit for cutoff value is that 49% of the Snort rules use the *depth* keyword: these rules require a pattern to be detected in a specific distance from the beginning of a packet or flow.

We propose two alternative techniques to ensure low latency for the high-priority packets when the system enters into a power saving mode and active cores' utilization increases: *time sharing* and *space sharing*.

C. TIME SHARING

Time sharing uses a typical priority queue scheduling to favor the high-priority packets. It first classifies packets into flows and then uses a flow cutoff to assign them a low or high priority. Then, packets are stored into the respective priority queue. When a new packet is scheduled for processing, the NIDS choose the next packet from the high-priority queue. If this queue is empty, a low-priority packet is chosen. However, this priority queue scheduling is non preemptive: when a high-priority packet arrives and a low-priority packet is being processed, the NIDS cannot evict the low-priority to serve immediately the high-priority packet. Time sharing follows the same strategy described at section III-C to adapt frequency and number of cores.

We expect a much lower latency for high-priority packets and a higher latency for the lower priority packets. With a careful cutoff selection, the majority of the attacks will be detected faster on high-priority packets. However, an attack detection on a lower priority packet will be significantly delayed.

D. SPACE SHARING

In time sharing, the cores of the energy-efficient NIDS remain almost fully utilized. This may cause reduced performance due to the non-preemptive priority queue scheduling. In space sharing, we use separate cores for each priority. We aim to keep cores that serve high-priority packets less utilized, to ensure low latency. In contrast, cores serving low-priority packets can remain highly utilized to allow for reduced power consumption. The increased latency for low-priority packets is less likely to affect the overall detection latency. As the majority of the packets have low priority (see Figure 7), most cores can be used to serve low-priority packets with high utilization to achieve energy savings.

In order to further reduce the detection latency, we would like to increase the frequency of the dedicated cores used to serve high-priority packets. However, the single per chip regulator in our Intel processors limits significantly our ability to change the frequency of high priority cores independently of low priority cores. Fortunately, our analysis in section IV shows that core utilization is the main factor of an increased detection latency. Thus, just reducing the utilization could be enough to achieve our low latency goal even with a lower frequency, which is necessary for low priority cores to reduce their power consumption.

Space sharing is based on two main ideas: *flow migration* and *adaptive core management*.

1) FLOW MIGRATION

The flow migration technique, assisted by advanced features of modern NICs, is used to distribute efficiently the packets into cores based on their priority. Initially, all packets arrive at the high-priority cores. Then, packets are classified into flows. When a flow size exceeds the specified cutoff value, the flow is moved into a low-priority core by instructing the

NIC to schedule all the successive packets of this flow into this core. Thus, only the high-priority packets remain for processing into the high-priority cores.

2) ADAPTIVE CORE MANAGEMENT

Space sharing dynamically partitions the active cores into high-priority and low-priority cores, based on the workload. It uses the optimum number of high-priority cores that keep their utilization within a desirable range. Using more cores than necessary may increase power consumption, while fewer cores may increase the detection latency. Therefore, we propose the following adaptive core management algorithm, which extends the core/frequency adaptive algorithm we presented in Section III-C:

1. It starts with one high-priority and one low-priority core at the minimum frequency.
2. It continuously monitors the queues' usage.
   - 2.1. If high-priority queues are filled by more than a *high-priority up threshold*:
     - 2.1.1. If exist inactive cores, activate a high-priority core.
     - 2.1.2. Else increase the frequency.
     - 2.1.3. If maximum frequency is used, reduce flow cutoff.
   - 2.2. If high-priority queues are filled by less than a *high-priority down threshold*:
     - 2.2.3. Increase cutoff up to a certain limit.
     - 2.2.1. Else reduce the frequency.
     - 2.2.2. If lowest frequency, deactivate a high-priority core.
   - 2.3. If low-priority queues are filled by more than a *low-priority up threshold*:
     - 2.3.1. If exist inactive cores, activate a low-priority core.
     - 2.3.2. Else increase the frequency.
   - 2.4. If low-priority queues are filled by less than a *low-priority down threshold*:
     - 2.4.1. Reduce the frequency.
     - 2.4.2. If lowest frequency, deactivate a low-priority core.

Because of the limitation of our single per chip regulator, we set the maximum frequency required by low-priority and high-priority cores. The *high-priority up threshold* ensures a low utilization for high-priority packets. The *low-priority up threshold* ensures that no packet will be lost. We can also control the load of high- and low-priority cores by changing the cutoff value, which divides the traffic into high- and low-priority packets. However, the cutoff always remains within a certain range.

E. DELAY ANALYSIS WITH PRIORITIES

We extend our analysis in Section IV-C for time sharing and space sharing. The high-priority packets arrive with rate λ_H, and low-priority packets with rate λ_L so that $\lambda_H + \lambda_L = \lambda$. The λ_H increases with the cutoff value based on the fraction of high-priority packets F_H given in Figure 7: $\lambda_H = \lambda \cdot F_H$. The service rate of each core at frequency f is $\mu_c(f)$.

1) TIME SHARING

The high- and low-priority packets arrive at each core with rate $\lambda_{cH} = \lambda_H/c$ and $\lambda_{cL} = \lambda_L/c$ respectively, where c the active cores. Time sharing can be modeled as a non-preemptive priority scheduling using two priority queues. The total delay of a high-priority packet T_H and of a low-priority packet T_L is:

$$T_H = W_H + \frac{1}{\mu_c(f)} = \frac{\rho_c}{\mu_c(f)\cdot(1-\rho_{cH})}$$
$$T_L = W_L + \frac{1}{\mu_c(f)} = \frac{\rho_c}{\mu_c(f)\cdot(1-\rho_{cH})\cdot(1-\rho_{cH}-\rho_{cL})} \quad (5)$$

where $\rho_c = \lambda_c/\mu_c(f)$, $\rho_{cH} = \lambda_{cH}/\mu_c(f)$, $\rho_{cL} = \lambda_{cL}/\mu_c(f)$.

2) SPACE SHARING

The high-priority packets arrive at high-priority cores with rate $\lambda_{cH} = \lambda_H/c_H$, where c_H the number of high-priority cores, and low-priority packets with rate $\lambda_{cL} = \lambda_L/c_L$, where c_L the number of low-priority cores. In space sharing, each core can be modeled as a M/M/1 queue with finite queue size of N packets. Thus, the total delay of a high-priority packet T_H and of a low-priority packet T_L are:

$$T_H = \frac{1}{\mu_c(f) - \lambda_{cH}} \qquad T_L = \frac{1}{\mu_c(f) - \lambda_{cL}} \quad (6)$$

The detection latency D in both time and space sharing is:

$$D = T_H \cdot P_H + T_L \cdot (1 - P_H) \quad (7)$$

where P_H the probability that an attack exists in a high-priority packet. This probability is given in Figure 7.

VI. IMPLEMENTATION

Based on the two alternative approaches we implemented LEoNIDS: a NIDS architecture that offers both low power consumption, proportionally to the load, and low detection latency. Figure 8 illustrates the architecture of LEoNIDS with time sharing and space sharing. Our implementation utilizes advanced features of modern NICs, and it is based on a specialized kernel module that modifies the packet capturing subsystem. Moreover, it includes a modified user-level packet capturing library and slight modifications to Snort NIDS [8].

We implemented the online frequency adaptation and core management algorithm within the packet capture subsystem as a Linux kernel loadable module. The module runs as a protocol handler and processes all captured packets. It is also responsible to store packets in the proper queues and impose a scheduling or load balancing policy. The packets are distributed among the available cores either with the RSS hash-based load balancing scheme [13] or with a dynamic load balancing scheme using the flow director filters (FDIR), which are used to define the core that will serve

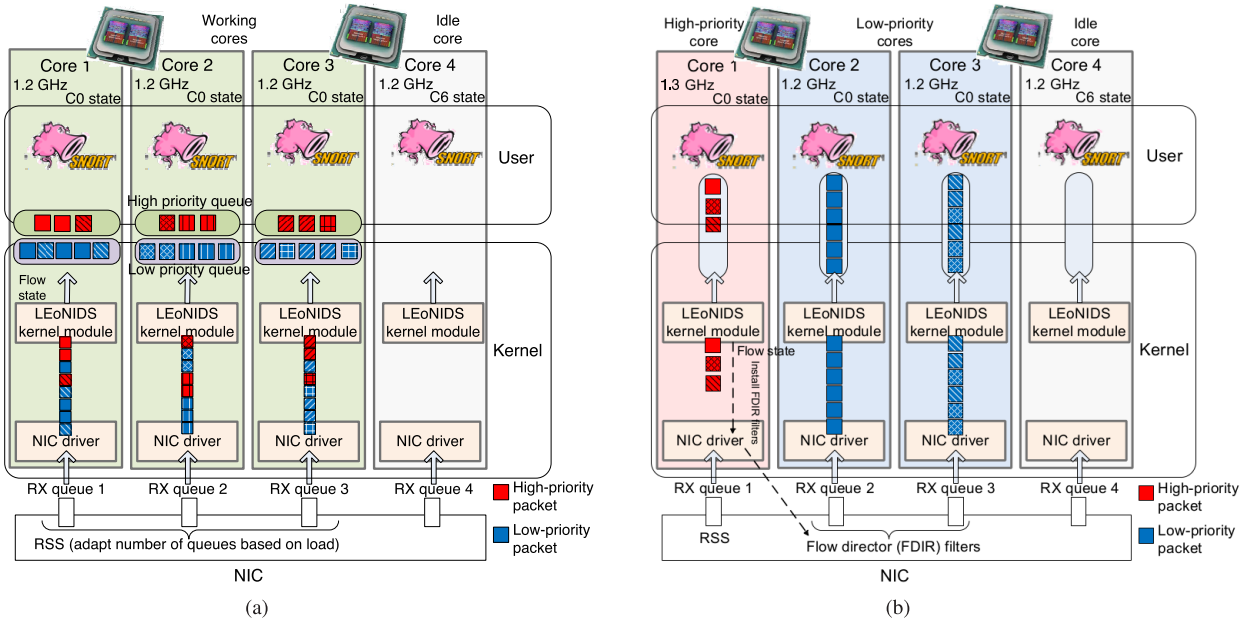

FIGURE 8. The LEoNIDS architecture with time sharing and space sharing. (a) LEoNIDS with time sharing. (b) LEoNIDS with space sharing.

each flow. We deliver packets at user-level through memory mapped buffers, and we built a libpcap [21] wrapper library. Then, we link Snort with this library, instead of the original libpcap.

3) TIME SHARING

In time sharing we extend the ring buffers of the packet capturing system using a typical priority queue scheme. The incoming packets are classified into flows and are assigned a low or high priority. Based on its priority, each packet is stored in the proper queue. The modified user level library reads the next packet from the high priority queue, and only if it is empty, from the low priority queue. This packet is then delivered to Snort for processing.

4) SPACE SHARING

In space sharing we use dedicated cores to process the high-priority packets with reduced latency. We aim to keep the utilization of these cores between 30%–50%, which results in low queuing delays as we see in Sections IV and IV-C. Based on the queue utilization we properly adapt flow cutoff, number of high-priority cores, and frequency. The RSS uses a redirection table to distribute the incoming packets to the available cores. To implement space sharing, we first modify the redirection table so that RSS splits all packets only to high-priority cores. Then, these cores classify packets into flows. When a flow exceeds the cutoff size, an FDIR filter is added to move the processing of this flow to a low-priority core. Along with the incoming packets, the flow context is also migrated to the low-priority core. This is important because (i) the new core need to be informed that there is an already established connection for the respective incoming packets, and (ii) in order to handle cases where an attack spans between both high- and low-priority packets.

Each flow that exceeds the cutoff value moves from one core to another only once. Using the FDIR filters for flow migration is highly efficient and improves cache performance, as each core accesses only its local data. We keep a list with all filters that are installed at the NIC, so when a flow expires (either explicitly by a TCP RST/FIN packet, or by an inactivity timeout) the respective FDIR filter is removed by the NIC. The intel 82599 NIC [14] offers up to 8K perfect match and 32K signature-based FDIR filters. In case all filters are used, space sharing evicts the oldest filter to accommodate a new flow.

VII. EXPERIMENTAL EVALUATION

In this section we experimentally evaluate our proposed approaches. We first compare time and space sharing while exploring the optimal cutoff values, and then we compare all approaches using real traffic.

A. COMPARING TIME AND SPACE SHARING

1) FINDING THE OPTIMAL CUTOFF

Using a small cutoff reduces the percentage of high-priority packets, and thus their queue utilization and queuing delays. However, the probability that an attack will be found in low-priority packets, which experience a higher delay, increases. To find out the optimal cutoff for time sharing and space sharing, we vary the cutoff values from 50 KB to 150 MB per flow while sending constant traffic at 1.0 Gbit/sec. Figure 9 shows that the optimal cutoff for both approaches is close to 500 KB. Using this cutoff, 99% of the attacks reside into the high-priority packets.

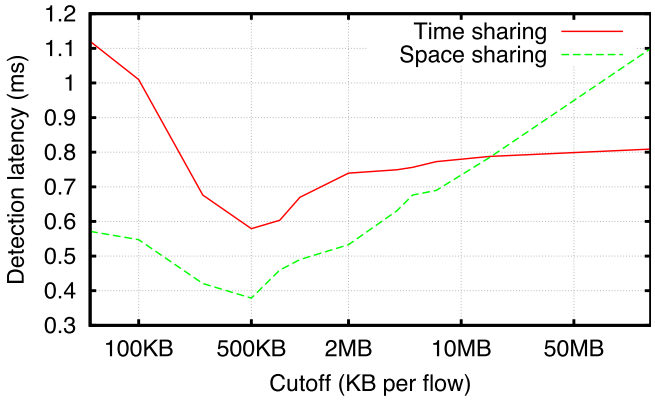

FIGURE 9. The optimal cutoff for time sharing and space sharing. Detection latency as a function of cutoff. In both time sharing and space sharing we see the lowest detection latency for 500 KB per flow.

For lower cutoff values, more attacks are found in low-priority packets with increased detection latency, while higher cutoff values increase the queuing delay of high-priority packets. We also see that space sharing achieves lower detection latency for all cutoff values below 20 MB, up to 50% lower for 50 KB cutoff and 35% lower for the optimal cutoff of 500 KB per flow.

2) THE EFFECT OF PRIORITY ON LATENCY

To better understand the detection latency we observed, we explore how the packet's latency (queuing plus processing time) changes for each priority with different cutoff values. Figure 10 shows the latency of high- and low-priority packets for time and space sharing as a function of cutoff when sending at 1.0 Gbit/sec. In time sharing, we see that low-priority packets experience up to 49.3 times higher latency than high-priority packets. As cutoff increases, we see a slight increase on the latency of high-priority packets due to the larger number of packets arriving at high-priority queues. Contrary, the latency of low-priority packets significantly decreases until cutoff reaches 750 KB, because the fraction of low-priority packets decreases, resulting in much less utilization in low-priority queues. When cutoff increases above 750 KB, the latency of low-priority packets increases fast. This is because they wait for an increasing number of high-priority packets to be processed.

In space sharing, we see a much lower difference between the latency of low- and high-priority packets. Note that both low- and high-priority packets experience lower latency compared to time sharing. Especially the latency of low-priority packets is significantly lower and clearly decreases as cutoff increases. This is because low- and high-priority packets are processed in parallel in different cores, and the fraction of low-priority packets decrease with higher cutoff values. The latency of high-priority packets is also reduced, as space sharing is able to keep high-priority cores less utilized. As the fraction of high-priority packets increase with cutoff, we see a slight increase on their latency for higher

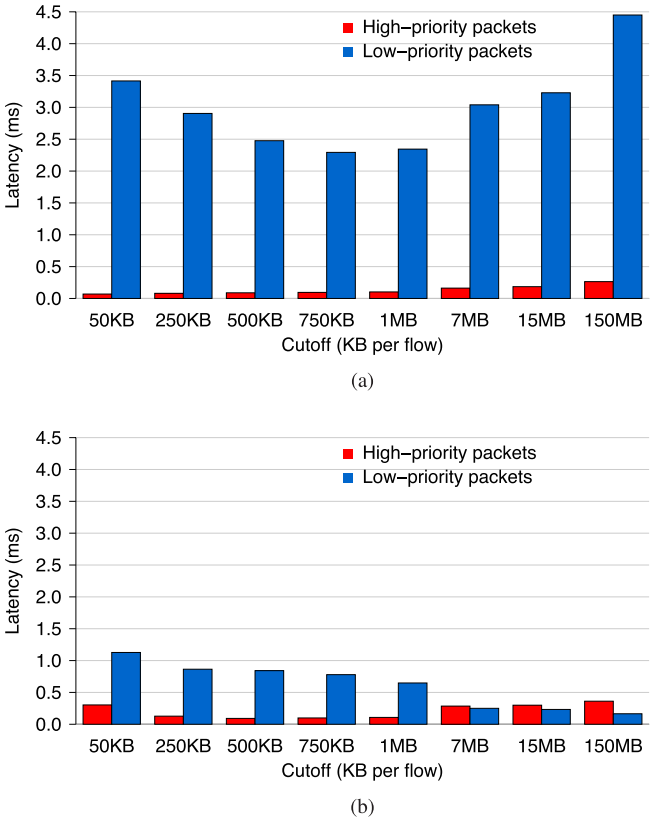

FIGURE 10. Low-priority packets in time sharing experience a much higher latency that high-priority packets. Latency of high- and low-priority packets for time sharing and space sharing as a function of cutoff when sending at 1.0 Gbit/sec. We see that both high-priority and (especially) low-priority packets experience lower latency in space sharing comparing with time sharing. (a) Time sharing. (b) Space sharing.

cutoff values. The increased latency in high-priority packets for very small cutoff values is due to the overhead of the very often FDIR establishments.

B. COMPARING ALL APPROACHES

1) VARYING THE LOAD

We now compare all approaches, *i.e.*, (*i*) the original Snort, (*ii*) the straight-forward power-proportional NIDS we described in section III, (*iii*) LEoNIDS with time sharing, and (*iv*) LEoNIDS with space sharing, in terms of both detection latency and energy efficiency when varying the traffic load. In time sharing we use a 500 KB cutoff, which was found to perform better. In space sharing we use an adaptive cutoff that ranges from 300 KB to 1 MB, *i.e.*, close to the optimal values. We have validated that all approaches detect the same set of attacks, i.e., they have the same detection accuracy. Figure 11 shows the power consumption and detection latency of all approaches as a function of traffic rate. We see that LEoNIDS with both approaches consumes approximately the same power as the power-proportional NIDS, significantly lower than the consumption of the original Snort. Despite the lower consumption, LEoNIDS achieves a significantly lower

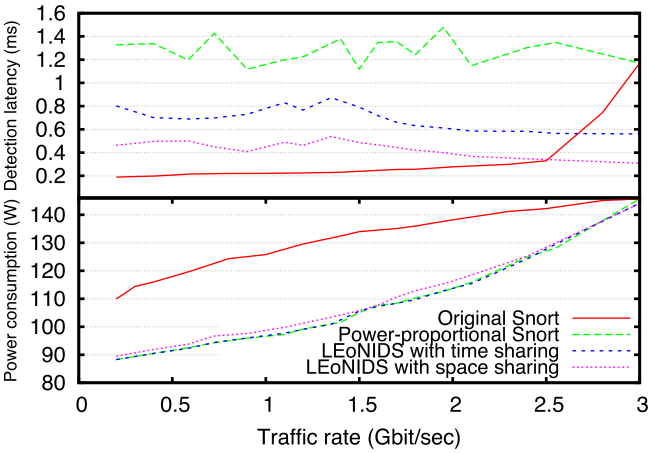

FIGURE 11. Space sharing offers the best power-latency ratio. Power consumption and detection latency of all approaches as a function of traffic rate. We see that LEoNIDS with space sharing consumes the same power with other power-proportional approaches, but with significantly lower detection latency.

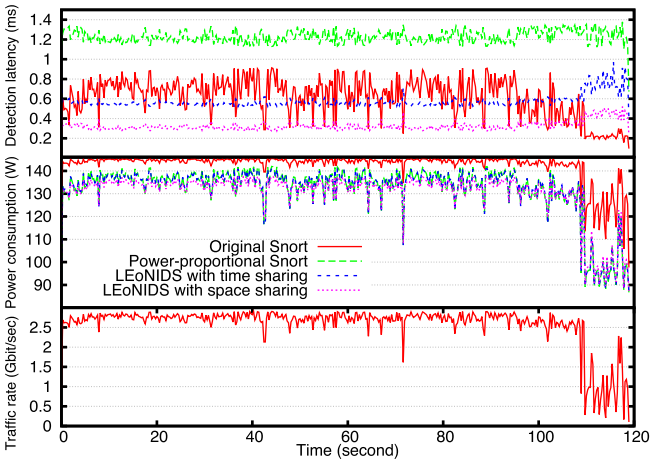

FIGURE 12. Space sharing performs better under realistic traffic variations. Traffic rate, power consumption, and detection latency over time.

detection latency than the power-proportional NIDS, close to the latency of the original system.

Space sharing performs quite better than time sharing: although both consume approximately the same power, space sharing achieves more than 40% lower detection latency. This is due to the non-preemptive priority queues used in time sharing: a high-priority packet may wait for a low-priority packet that is being processed. Moreover, the overall utilization of active cores in time sharing remain very high, so it cannot efficiently reduce the queuing delays of high-priority packets. Overall, LEoNIDS with space sharing consumes 22% less power than the original system and it is able to detect attacks with an order of magnitude lower latency than the straight-forward power-proportional NIDS. Moreover, space sharing achieves a lower detection latency than the original system for rates higher than 2.5 Gbit/sec. This is due to the higher priority given at the beginning of each flow. As the original system does not give priority to these packets, it experiences higher detection latency at high traffic rates where all approaches result in an almost fully utilized system.

2) REALISTIC TRAFFIC VARIATIONS

In our next experiment we compare all approaches in a realistic scenario of traffic variations. We replayed our one-hour long trace at its original rate using a 30x multiplier, resulting in an 120-seconds long experiment. Figure 12 shows the traffic rate, power consumption, and detection latency of all approaches over time. We see that again LEoNIDS with space sharing achieves the lowest detection latency among the other power-proportional approaches.

3) ACTIVE RESPONSE

In our last experiment we examine how the detection latency of each approach affects the effectiveness of a NIDS reaction to actively terminate offending TCP connections. We configured Snort with *flexresp2* plugin for active response, and we added a rule to match a specific string and respond with reset to both source and destination hosts of the matched flow. While sending background traffic at 1.0 Gbit/sec, we were also sending connections with packets matching this string. We sent a constant number of packets per connection, while varying its duration. Figure 13 shows the percentage of successfully closed connections by active response when sending 100 such connections, as a function of connection's duration. We see that the straight-forward power-proportional NIDS cannot respond in time and close connections shorter than 6 ms with more than 50% probability. Contrary, LEoNIDS is able to terminate most connections lasting more than 3 ms, similar to the original Snort.

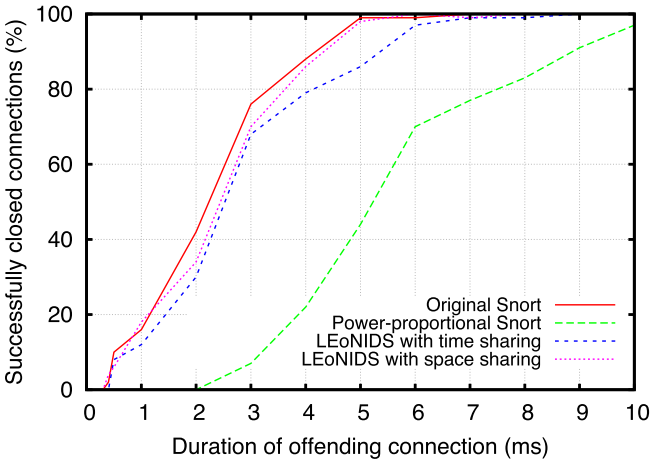

FIGURE 13. LEoNIDS close most offending connections longer than 3 ms. Percentage of closed connections as a function of connection's duration.

VIII. RELATED WORK

Recent works show a growing emphasis on improving the energy efficiency in a variety of areas, like data centers [1], [22], high performance computing [2], networks [4], [23] and mobile devices [3]. Gupta and Singh [4] suggest to put components of network devices into sleep states and propose protocol modifications to save energy. Closer to our work, Niccolini *et al.* [24] present an approach for a power-proportional software router using DVFS and C-states. Iqbal and John [25] propose a predictive power management scheme to pro-actively change the frequency and number of active cores in network processors. Nedevschi *et al.* [23] also use sleep states and rate adaption to save energy. We follow a similar approach to the above works regarding the power management. However, these works do not consider the impact of reduced energy on systems' latency. In contrast, our work is mostly focused on resolving the energy-latency tradeoff for energy-efficient intrusion detection systems.

Kuang and Bhuyan [26] propose a scheduling algorithm to optimize throughput and latency given a power budget for network packet processing on multi-core processors. The energy-latency tradeoff has been studied in the area of wireless sensor networks [27], [28]. These works design communications protocols that minimize the power consumption of sensor nodes while satisfying latency constraints. To the best of our knowledge, our work is the first effort to study and resolve the energy-latency tradeoff in the area of network monitoring and intrusion detection systems.

Pesterev *et al.* [29] utilize FDIR filters to improve connection locality in multi-core systems, similarly to our flow migration technique we use in space sharing. Another related approach by Afek *et al.* [30] uses dedicated cores to defend against algorithmic complexity attacks in intrusion detection systems.

IX. CONCLUSIONS

In this work we studied the problem of improving the energy efficiency of NIDS using common power management capabilities like DVFS and C-states. First, we identified an energy-latency tradeoff: the reduced power consumption results in a significant increase of the detection latency, which impedes a timely reaction of NIDS to incoming attacks. We showed that the main reason of this increase is the high queuing delays imposed by high core utilization. Then, we presented the design, implementation, and evaluation of LEoNIDS: a NIDS that resolves the energy-latency tradeoff. They key idea of LEoNIDS is to process with higher priority the first few bytes of each flow, which have a higher probability to carry an attack, to achieve lower latency for them and faster attack detection. We proposed two alternative techniques: time sharing, which uses a typical priority queue scheduling, and space sharing, which uses dedicated cores with low utilization to process high-priority packets. Our experimental evaluation shows that LEoNIDS performs better with space sharing, resulting in low power consumption, proportionally to the load, and constantly low attack detection latency at the same time.

ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers for their valuable feedback. Antonis Papadogiannakis and Evangelos Markatos are also with the University of Crete, 700 13, Heraklion, Greece.

REFERENCES

[1] K. Shen, A. Shriraman, S. Dwarkadas, X. Zhang, and Z. Chen, "Power containers: An OS facility for fine-grained power and energy management on multicore servers," in *Proc. ACM Int. Conf. Archit. Support Program. Lang. Oper. Syst. (ASPLOS)*, 2013, pp. 65–76.

[2] Q. Deng, D. Meisner, L. Ramos, T. F. Wenisch, and R. Bianchini, "MemScale: Active low-power modes for main memory," in *Proc. ACM Int. Conf. Archit. Support Program. Lang. Oper. Syst. (ASPLOS)*, 2011, pp. 225–238.

[3] A. Pathak, Y. C. Hu, and M. Zhang, "Where is the energy spent inside my app?: Fine grained energy accounting on smartphones with Eprof," in *Proc. ACM Eur. Conf. Comput. Syst. (EuroSys)*, 2012, pp. 29–42.

[4] M. Gupta and S. Singh, "Greening of the internet," in *Proc. ACM SIGCOMM Conf. Data Commun.*, 2003, pp. 19–26.

[5] V. Paxson, R. Sommer, and N. Weaver, "An architecture for exploiting multi-core processors to parallelize network intrusion prevention," in *Proc. IEEE Sarnoff Symp.*, Apr./May 2007, pp. 1–7.

[6] C. Kruegel, F. Valeur, G. Vigna, and R. Kemmerer, "Stateful intrusion detection for high-speed networks," in *Proc. IEEE Symp. Secur. Privacy*, May 2002, pp. 285–293.

[7] M. Vallentin, R. Sommer, J. Lee, C. Leres, V. Paxson, and B. Tierney, "The NIDS cluster: Scalable, stateful network intrusion detection on commodity hardware," in *Proc. Int. Symp. Recent Adv. Intrusion Detection (RAID)*, 2007, pp. 107–126.

[8] M. Roesch, "Snort—Lightweight intrusion detection for networks," in *Proc. USENIX Large Installation Syst. Admin. Conf. (LISA)*, 1999, pp. 229–238.

[9] *Snort Active Response*. [Online]. Available: http://manual.snort.org/node26.html

[10] H. Dreger, A. Feldmann, V. Paxson, and R. Sommer, "Operational experiences with high-volume network intrusion detection," in *Proc. ACM Conf. Comput. Commun. Secur. (CCS)*, 2004, pp. 2–11.

[11] *Sourcefire Vulnerability Research Team (VRT)*. [Online]. Available: http://www.snort.org/vrt/

[12] F. Fusco and L. Deri, "High speed network traffic analysis with commodity multi-core systems," in *Proc. ACM SIGCOMM Conf. Internet Meas. (IMC)*, 2010, pp. 218–224.

[13] Intel Server Adapters. *Receive Side Scaling on Intel Network Adapters*. [Online]. Available: http://www.intel.com/support/network/adapter/pro100/sb/cs-027574.htm

[14] Intel. (2011). *82599 10 GbE Controller Datasheet*. [Online]. Available: http://download.intel.com/design/network/datashts/82599_datasheet.pdf

[15] M. Polychronakis, K. G. Anagnostakis, and E. P. Markatos, "An empirical study of real-world polymorphic code injection attacks," in *Proc. USENIX Workshop Large-Scale Exploits Emergent Threats (LEET)*, 2009, p. 9.

[16] A. Papadogiannakis, M. Polychronakis, and E. P. Markatos, "Tolerating overload attacks against packet capturing systems," in *Proc. USENIX Annu. Tech. Conf. (ATC)*, 2012, p. 18.

[17] A. Papadogiannakis, M. Polychronakis, and E. P. Markatos, "Improving the accuracy of network intrusion detection systems under load using selective packet discarding," in *Proc. ACM Eur. Workshop Syst. Secur. (EUROSEC)*, 2010, pp. 15–21.

[18] T. Limmer and F. Dressler, "Improving the performance of intrusion detection using dialog-based payload aggregation," in *Proc. IEEE Global Internet Symp. (GI)*, Apr. 2011, pp. 822–827.

[19] G. Maier, R. Sommer, H. Dreger, A. Feldmann, V. Paxson, and F. Schneider, "Enriching network security analysis with time travel," in *Proc. ACM SIGCOMM Conf. Data Commun.*, 2008, pp. 183–194.

[20] W. Fang and L. Peterson, "Inter-AS traffic patterns and their implications," in *Proc. IEEE Global Telecommun. Conf. (GLOBECOM)*, Dec. 1999, pp. 1859–1868.

[21] S. McCanne, C. Leres, and V. Jacobson. *Libpcap*. [Online]. Available: http://www.tcpdump.org/
[22] D. Meisner, B. T. Gold, and T. F. Wenisch, "PowerNap: Eliminating server idle power," in *Proc. ACM Int. Conf. Archit. Support Program. Lang. Oper. Syst. (ASPLOS)*, 2009, pp. 205–216.
[23] S. Nedevschi, L. Popa, G. Iannaccone, S. Ratnasamy, and D. Wetherall, "Reducing network energy consumption via sleeping and rate-adaptation," in *Proc. USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2008, pp. 323–336.
[24] L. Niccolini, G. Iannaccone, S. Ratnasamy, J. Chandrashekar, and L. Rizzo, "Building a power-proportional software router," in *Proc. USENIX Annu. Tech. Conf. (ATC)*, 2012, p. 8.
[25] M. F. Iqbal and L. K. John, "Efficient traffic aware power management in multicore communications processors," in *Proc. ACM/IEEE Symp. Archit. Netw. Commun. Syst. (ANCS)*, 2012, pp. 123–134.
[26] J. Kuang and L. Bhuyan, "Optimizing throughput and latency under given power budget for network packet processing," in *Proc. IEEE Int. Conf. Comput. Commun. (INFOCOM)*, Mar. 2010, pp. 1–9.
[27] Y. Yu, B. Krishnamachari, and V. K. Prasanna, "Energy-latency tradeoffs for data gathering in wireless sensor networks," in *Proc. IEEE Int. Conf. Comput. Commun. (INFOCOM)*, Mar. 2004, p. 255.
[28] G. Lu, B. Krishnamachari, and C. S. Raghavendra, "An adaptive energy-efficient and low-latency MAC for tree-based data gathering in sensor networks," *Wireless Commun. Mobile Comput.*, vol. 7, no. 7, pp. 863–875, Sep. 2007.
[29] A. Pesterev, J. Strauss, N. Zeldovich, and R. T. Morris, "Improving network connection locality on multicore systems," in *Proc. ACM Eur. Conf. Comput. Syst. (EuroSys)*, 2012, pp. 337–350.
[30] Y. Afek, A. Bremler-Barr, Y. Harchol, D. Hay, and Y. Koral, "MCA2: Multi-core architecture for mitigating complexity attacks," in *Proc. ACM/IEEE Symp. Archit. Netw. Commun. Syst. (ANCS)*, Oct. 2012, pp. 235–246.

NIKOS TSIKOUDIS is currently pursuing the Ph.D. degree with the Department of Computer Science, Brandeis University, Waltham, MA, USA, where he is also a Research Assistant. He received the B.Sc. and M.Sc. degrees in computer science from the University of Crete, Heraklion, Greece, in 2010 and 2013, respectively, while working as a Research Assistant with the Distributed Computing Systems Laboratory, Foundation for Research and Technology—Hellas, Institute of Computer Science, Heraklion, Greece. His main research interests are in the area of distributed systems.

ANTONIS PAPADOGIANNAKIS is currently a Research Assistant with the Distributed Computing Systems Laboratory, Foundation for Research and Technology—Hellas, Institute of Computer Science, Heraklion, Greece. He received the B.Sc., M.Sc., and Ph.D. degrees in computer science from the University of Crete, Heraklion, Greece, in 2005, 2007, and 2014, respectively. His main research interests are in the areas of network monitoring, packet capture, network and systems security, privacy, and network measurements.

EVANGELOS P. MARKATOS is currently a Full Professor of Computer Science with the University of Crete, Heraklion, Greece, and the Director of the Distributed Computing Systems Laboratory with the Foundation for Research and Technology—Hellas, Institute of Computer Science, Heraklion, Greece. He received the Diploma degree in computer engineering from the University of Partas, Partas, Greece, in 1988, and the M.Sc. and Ph.D. degrees in computer science from the University of Rochester, Rochester, NY, USA, in 1990 and 1993, respectively. His research interests are in the areas of computing systems, security, and privacy.