University of Crete
School of Sciences and Engineering
Computer Science Department

LEoNIDS: a Low-latency and Energy-efficient Network-level Intrusion Detection System

Nikos Tsikoudis

Master's thesis

June 2013 Heraklion, Greece

University of Crete
Computer Science Department

LEoNIDS: a Low-latency and Energy-efficient Network-level Intrusion Detection System

Thesis submitted by
Nikos Tsikoudis
in partial fulfillment of the requirements for the
Masters' of Science degree in Computer Science

THESIS APPROVAL

Author:	Nikos Tsikoudis
Committee approvals:	Evangelos Markatos Professor, Thesis Supervisor
	Kostas Magoutis Research scientist, ICS-FORTH, Committee Member
	Apostolos Traganitis Professor, Committee Member
Departmental approval:	Angelos Bilas Professor, Director of Graduate Studies

Heraklion, June 2013

Abstract

Over the past decade, the design and implementation of low-power systems has received significant attention. Started with data centers and battery-operated mobile devices, it has recently branched to core network devices such as routers. Towards 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 and core deactivation, lead to a disproportionate increase in packet processing and queuing times. In this work, we show that this increase has a negative impact on the detection latency and impedes a timely reaction of a NIDS to the incoming attacks.

To address this issue, we present LEoNIDS: a NIDS architecture that resolves the energy-latency tradeoff by providing *both* low power consumption *and* low detection latency at the same time. The key idea of LEoNIDS is to identify the packets that are more likely to carry an attack. These packets are given higher priority so as to achieve low attack detection latency, while the rest of the packets are scheduled to run at a much lower priority, achieving overall low power consumption. Our results indicate that LEoNIDS consumes comparable power to a state-of-the-art low-power design, while, at the same time, achieving up to an order of magnitude faster attack detection latency.

Περίληψη

Κατά τη διάρκεια της τελευταίας δεκαετίας έχει γίνει μεγάλη προσπάθεια για την σχεδίαση και υλοποίηση συστημάτων με χαμηλή κατανάλωση ενέργειας. Ξεκινώντας από data centers και κινητές συσκευές που λειτουργούν με μπαταρία, έχει πρόσφατα επεκταθεί σε κύρια τμήματα του δικτύου όπως δρομολογητές. Προς αυτή την κατεύθυνση, έχουμε ως στόχο την μείωση της κατανάλωσης ενέργειας των συστημάτων ανίχνευσης δικτυακών επιθέσεων τα οποία χρησιμοποιούνται για την βελτίωση της ασφαλής λειτουργίας των σύγχρονων δικτύων. Δυστυχώς, οι παραδοσιακές προσεγγίσεις σχεδίασης για χαμηλή κατανάλωση ενέργειας, όπως κλιμάκωση συχνότητας και απενεργοποίηση πυρήνων, οδηγούν σε δυσανάλογη αύξηση του χρόνου επεξεργασίας και αναμονής των πακέτων. Στην εργασία αυτή δείχνουμε ότι η αύξηση αυτή έχει αρνητικό αντίκτυπο στο χρόνο ανίχνευσης των επιθέσεων και παρεμποδίζει την έγκαιρη αντίδραση ενός συστήματος ανίχνευσης δικτυακών επιθέσεων στις εισερχόμενες επιθέσεις.

Για να αντιμετωπίσουμε αυτό το ζήτημα, παρουσιάζουμε το LEoNIDS: μια αρχιτεκτονική για συστήματα ανίχνευσης δικτυακών επιθέσεων που καταφέρνει να παρέχει ταυτόχρονα χαμηλή κατανάλωση ενέργειας και χαμηλό χρόνο ανίχνευσης επιθέσεων. Η βασική ιδέα του LEoNIDS είναι να προσδιορίζει τα πακέτα που είναι πιο πιθανό να περιέχουν μία επίθεση. Στα πακέτα αυτά δίνεται μεγαλύτερη προτεραιότητα ώστε να επιτευχθεί χαμηλός χρόνος ανίχνευσης επιθέσεων, ενώ τα υπόλοιπα πακέτα δρομολογούνται με χαμηλότερη προτεραιότητα, ώστε να επιτευχθεί χαμηλότερη κατανάλωση ενέργειας. Τα αποτελέσματα δείχνουν ότι το LEoNIDS καταναλώνει συγκρίσιμη ενέργεια με ένα τυπικό σύστημα χαμηλής ενεργειακής κατανάλωσης, ενώ ταυτόχρονα επιτυγχάνει μία τάξη μεγέθους χαμηλότερη καθυστέρηση ανίχνευσης επιθέσεων.

Acknowledgments

I would like to thank my Supervisor, Professor Evangelos P. Markatos, for his valuable guidance during my studies and the chance to be a part of the Distributed Computing Systems (DCS) Lab, at FORTH-ICS. I am also grateful to Antonis Papadogiannakis, whose advices and support played a crucial role during my research.

I need to express my gratitude to past and current members of the DCS Lab for their support and for contributing for a pleasant environment in the lab. A heartfelt thank you to my friends for their support and especially to Sofia Gkika for her constant and endless encouragement.

Finally, I would never made it through without my family. I am grateful to my father Giannis, my mother Vangelio and my brother Manos for their support in every single aspect of my life.

This work has been conducted in the **Distributed Computing Systems(DCS)** laboratory, **Institute of Computer Science (ICS)**, **Foundation for Research and Technology - Hellas (FORTH)** and supported by the FP7 project SysSec, funded by the European Commission under Grant Agreement No. 257007.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Low power consumption is 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 [33], high performance computing [6], and mobile devices [26]. Also significant attention receive core network devices like routers [23]. Towards this direction, we aim to build an energy efficient Network-level Intrusion Detection System (NIDS) [27, 31]

NIDS are commonly deployed to detect security violations, enhancing the secure operation of modern computer networks. They perform computationally heavy operations like multiple pattern matching, regular expression matching, and other types of complex analysis on the incoming packets to detect at real time malicious activities in the monitored network. Thus, NIDS usually utilize multi-core systems [28] or cluster of servers [15, 32, 35] to cope with increased link speeds and complicated analysis of network traffic.

1.1 Low-Power NIDS Design

Although NIDS are usually provisioned to operate at link rate, in order to be able to handle a fully utilized network at the worst case, most networks are typically much less utilized than their maximum capacity. 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 are usually found in modern processors. The system should consume the least power needed to sustain the incoming traffic load.

We find that a NIDS consumes less power when it uses the smallest number of cores at the lowest possible frequency, by keeping these cores nearly fully utilized. This is inline with previous studies on energy-efficient networked systems [23]. Our results indicate that this energy-efficient NIDS can process all packets with up to 23% lower power consumption than the original system at low rates. However, we not surprisingly observe a significant increase on the attack detection latency.

There is an up to 7x increase due to higher packet processing times when reducing the frequency, and mostly due to increased queuing delays imposed by the high CPU utilization.

A low detection latency is very important for a NIDS in order to ensure a timely reaction to the attack. Upon the detection of a packet that carries an attack, the NIDS can terminate the offending connection by actively sending TCP reset (RST) packets or installing a new firewall rule. This reaction should happen as soon as possible, definitely before the attack packets reach the victim's machine and the attack succeeds.

1.2 Motivation

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

1.2.1 Why NIDS Power Consumption Matters

NIDS are usually over-provisioned to handle a fully utilized line and tolerate overloads without missed attacks [7]. Thus, they use all the available resources: all cores [28], and the maximum CPU frequency. In high speed networks, the traffic load may also be split among multiple machines [15, 35, 32]. 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 spent and increased cost for running multiple NIDS to protect a large infrastructure.

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

1.2.2 Why Detection Latency Matters

Although a NIDS operating in passive mode does not affect the actual latency of the monitored packets, a low packet processing latency and thus 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. One way to achieve this is to actively terminate an offending TCP connection by sending TCP reset (RST) packets with the correct sequence numbers and spoofed IP addresses of victim and attacker hosts (*e.g.*, using Snort's Flexresp active response). Since a reset packet may not reach the client or server before this host has already responded to the attack packet, the NIDS tries to terminate 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 (round-trip time and throughput), NIDS may or may not win this race. Thus, a NIDS should be able to detect the incoming attacks fast in order to stop fast TCP connections.

Another possible reaction of a NIDS upon an attack detection 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 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.

1.3 Proposed Approach

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 lower latency. In previous work [24] we have found that the first few packets of each connection have a much higher probability to carry an attack. Thus, processing these packets with low latency results in fast detection in most cases. An increased latency for the rest packets, in order to save power, is less likely to affect the overall detection latency.

We explore two alternative approaches to achieve lower latency for the first packets of each connection: *time sharing* and *space sharing*. In time sharing we give higher priority to these packets using a typical priority queue scheduling in each core. In space sharing the high-priority packets follow a different path, using few dedicated cores with much lower utilization. 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. Thus, we keep the first packets of each flow in cores with low utilization to achieve low latency. We experimentally compare the two approaches and both of them with the original system and we find that space sharing has a better power-latency ratio. This is because time sharing cannot efficiently reduce the queuing delays during a high utilization and the original system consumes 22% more power.

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 is based on the popular Snort NIDS [31]. LEoNIDS consumes less power, proportionally to the traffic load, while its detection latency remains low and almost constant at any traffic load.

1.4 Contributions

The main contributions of this work are:

- We identify a new tradeoff for NIDS: the *energy-latency tradeoff* which affects the detection latency. As we reduce power consumption the detection latency is significantly increased, which impedes a timely reaction to the incoming attacks. We see that the main cause of the increased detection latency is the queuing delays imposed by the high utilization at each active core.

- We resolve the energy-latency tradeoff by identifying the most important packets and processing them with low latency. These are the first packets of each connection, which have a higher probability to contain an attack.

- We introduce space sharing: a new technique based on flow migration that processes the most important packets in dedicated cores with low utilization, and moves the processing of least significant packets to cores with higher utilization.

- We experimentally compare two alternative approaches for low latency in a power-proportional NIDS: time sharing and space sharing. We show that space sharing results in lower detection latency when power consumption is reduced. It utilizes the NIC capabilities better than the time sharing does.

- We present the design, implementation, and experimental evaluation of LEoNIDS, a NIDS architecture that achieves both low latency and reduced power consumption.

1.5 Thesis Outline

The rest of this thesis is structured as follows. Chapter 2 presents similar studies to ours. In chapter 3 we explore the design of a power-proportional NIDS. How detection latency is affected is described in chapter 4 with both experimental and analytical evaluation, indicating the energy-latency tradeoff. Chapter 5 identifies the solution, resolving the energy-latency tradeoff and presents the two possible approaches. Implementation and experimental evaluation of the approaches are described in chapters 6 and 7 respectively. Finally, chapter 10 concludes the thesis

Chapter 2

Related Work

Several recent works show a growing emphasis on improving the energy efficiency in a variety of areas, like data centers [14, 21, 33], high performance computing [6], networks [1, 5, 10, 13, 22, 23] and mobile devices [4, 18, 26, 37]. This has led to the deployment of new advanced options for power management in modern computer systems. Furthermore, there are works that study throughput and latency optimizations, especially in the area of wireless sensor networks [16, 19, 38]. Finally, various research projects share ideas with this work like flow director filters and using dedicated cores for specific processes [3, 29].

2.1 Energy-Efficient Systems

Focusing on energy efficient data centers Kontorinis *et al.* [14] propose a distributed UPS architecture that stores energy in UPS batteries during idle periods and uses it during power spikes. Meisner *et al.* [21] present an approach to reduce the idle power consumption of a server by rapidly switching from a high-performance active state to a minimal-power nap state in response to instantaneous load. The authors in [33] evaluate and control the power and energy usage of workloads on a multi-core server. Some more specialized systems focus on database workloads evaluating current query optimizers in terms of energy efficiency [34] and low-power modes for main memories [6].

Energy efficiency is also a major issue for the mobile devices because of their limited battery capacity. Lindsey *et al.* [18] propose a protocol for power efficient information gathering in sensor systems. Each node communicates only with a close neighbor and takes turns transmitting its packets to the distant base station. Recent works focus on profiling the energy for smartphones apps [26] and monitoring the power consumption of every application process [37]. A closely related to our work, presents the energy constrains when adopting malware detection on mobile devices and determines an optimal spot for protection while consuming a limited amount of battery power [4].

2.2 Energy Efficiency in Networks

Both industrial and research works aim to improve energy efficiency in networked systems. Gupta and Singh [10] suggest to put components of network devices into sleep states in order to save energy, and propose modifications to the current protocols for energy savings. Closer to our work, Niccolini *et al.* [23] present an approach for a power-proportional software router that utilizes DVFS and C-states using an online adaptation algorithm. Iqbal and John [13] propose a predictive power management scheme to pro-actively change the frequency and number of active cores in network processors. Nedevschi *et al.* [22] also use sleep states and rate adaption to save energy. In contrast to the above, authors in [5] propose to modify routing protocol for energy efficiency, avoiding changes to the internals of the network components.

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 performance metrics like latency. Our work is mostly focused on resolving the energy-latency tradeoff for energy-efficient intrusion detection systems.

2.3 Energy-Latency Tradeoff

Kuang and Bhuyan [16] 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 well studied in the area of wireless sensor networks [19, 38]. These works aim to design communications protocols that minimize the power consumption of the 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 although there is growing attention in their performance [28, 36].

2.4 Utilizing Features of Modern NICs

Pesterev *et al.* [29] utilize the flow director filters to improve connection locality in multi-core systems, similarly to the flow migration technique we use in space sharing. An another related approach by Afek *et al.* [3] uses dedicated cores to defend against algorithmic complexity attacks in NIDS.

Chapter 3

Towards a Power Proportional NIDS

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

3.1 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 10GbE network interface. The clock frequency of these processors can be scaled from 1.2 GHz to 2.0 GHz using DVFS, which results in 9 available frequency steps (P-states). They also support Intel Turbo Boost technology to further increase their frequency up to 2.5 GHz. To reduce power consumption, each idle core can be put independently into one of the 3 available sleep states: C1, C3 or C6. We measure the power consumption in the NIDS machine using the Watts up? PRO ES device, by sampling and storing the power at one second intervals. All our measurements run for significantly higher time periods than one second. The idle power consumption is 85.1 W and when all cores are 100% utilized it reaches 152.6 W.

The first machine is used for traffic generation. The generated traffic reaches the second machine, which runs the Snort IDS [31] v2.8.3.2 with the official rule set [2] containing 8308 rules. We use PF_RING [9] v5.3.0 and ixgbe driver v3.7.17 to split the incoming traffic to the active cores using the Receive Side Scaling (RSS) [12] feature of Intel 82599 NIC [11]. 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 full payload anonymized trace captured at the access link of an educational 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 1796 alerts from 76 different rules. Most of these matching rules are related to common threats and protocol violations. In

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

Table 3.1: CPU consumes the larger portion of energy in a NIDS. More than 50% of power is consumed at the CPU.

order to strengthen our evaluation, we augmented the trace with 120 short traces of real attacks captured in the wild [30], adding 233 more alerts from 14 different rules. The resulting trace generates a total of 1938 alerts due to 90 different attack signatures.

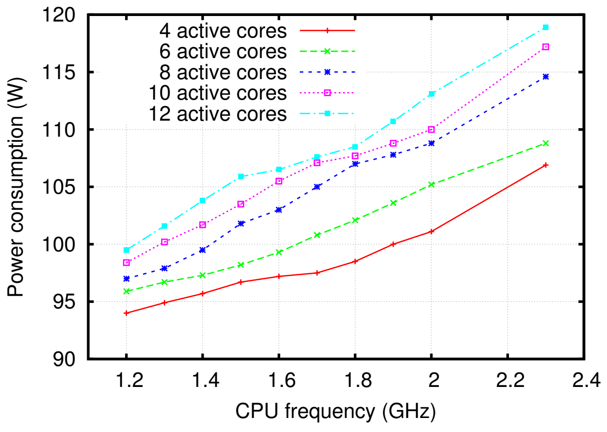
3.2 Power Consumption

The system's idle power is 85.1 W. When running an idle NIDS the power consumption is 87.1 W, while a fully utilized NIDS consumes 145.7 W. Thus, we estimate that the extra power from idle state is consumed by the NIDS. Since NIDS perform heavy computational operations, the CPU consumes the larger portion of energy in the system. Table 3.1 shows the contribution of the CPU in the total power consumption. Varying the traffic rate results in different utilization of the original NIDS. In all cases, 58-62% of the total power is consumed by the two CPUs. We measure the CPU power consumption by accessing the RAPL (Running Average Power Limit) registers provided by Intel Xeon E5-2620 CPUs, which measure the total energy consumed by each chip.

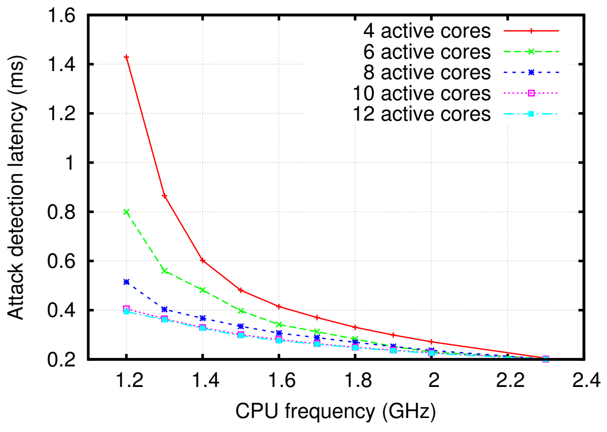
Modern processors offer two ways for reduced power consumption: frequency scaling (DVFS), and sleep states (C-states). Intel processors have a single voltage and frequency regulator, so the frequency change uniformly for all cores of this processor. On the other hand, each core can operate in a different C-state to save energy. The power consumption of each core consists of (i) the active power consumed when the core process packets at the used frequency, (ii) the power consumed to enter a C-state, and (iii) the power consumed during the idle state. We see that idle cores consume less power when they are in C6 state, so we put inactive cores in this state.

3.3 Exploring the Design Space

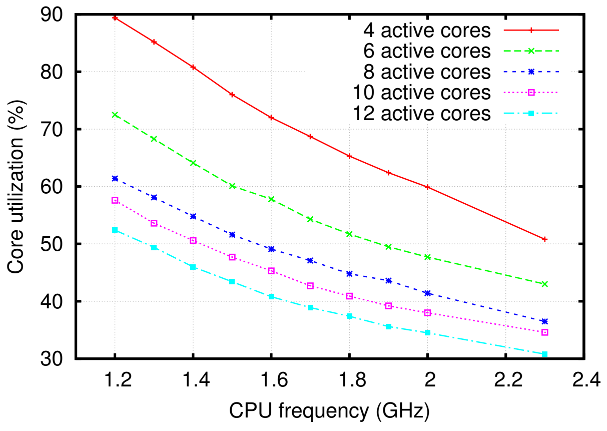
Based on the packet arrival rate, we aim to find the most energy-efficient strategy for a NIDS by properly adapting the CPU frequency and the number of active cores (not in C-states). The two main questions are: (i) is it better to operate at lower frequency or utilize sleep states? (ii) is it better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



(a) Power consumption



(b) Detection latency



(c) Utilization per active core

Figure 3.1: 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. Power consumption decreases as the utilization of active cores approaches 100%. However, this results in increased detection latency.

Active cores	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

Table 3.2: Using more cores at lower frequency is more energy efficient, but results in higher latency.

A related work on power-proportional software routers [23] shows that the less power is consumed when the system operates in the lowest possible frequency with no idle time, compared with higher frequencies and idle times that put cores in C-states. It also shows that it is better to utilize more cores in lower frequency than less cores in higher frequency. Both findings lead to the conclusion that the lowest power consumption is achieved with higher CPU utilization, *i.e.*, when the resulting service rate is closer to the packet arrival rate.

To explore whether the above strategy is also optimal for NIDS, we measure Snort's power consumption as a function of the CPU frequency and the number of active cores, when sending traffic at a constant rate of 0.6 Gbit/sec. Figure 3.1(a) shows that the lowest power consumption is achieved when using 4 active cores at 1.2 GHz, which is the minimum setup that is 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 as with software routers, 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 less cores at higher frequencies. For instance, 6 cores at 1.2 GHz consume 95.9 W, while 4 cores at 1.8 GHz, which offer approximately the same computing power, consume 98.5 W. As another example, Table 3.2 shows three alternative setups that can be used to process 1.5 Gbit/sec: 6 cores at 2.0 GHz, 8 cores at 1.5 GHz, or 10 core at 1.2 GHz. These setups offer approximately the same computing power. We see that 10 cores at 1.2 GHz consume the less power.

Figure 3.1(c) shows the respective utilization of the 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.

3.4 Adapt to the Traffic Load

Our experimental results indicate that a power-proportional NIDS should utilize the smallest number of cores that are able to sustain the incoming traffic without any packet loss when they operate at the lowest possible frequency. Therefore, the system should dynamically adapt to the traffic load by changing the frequency and

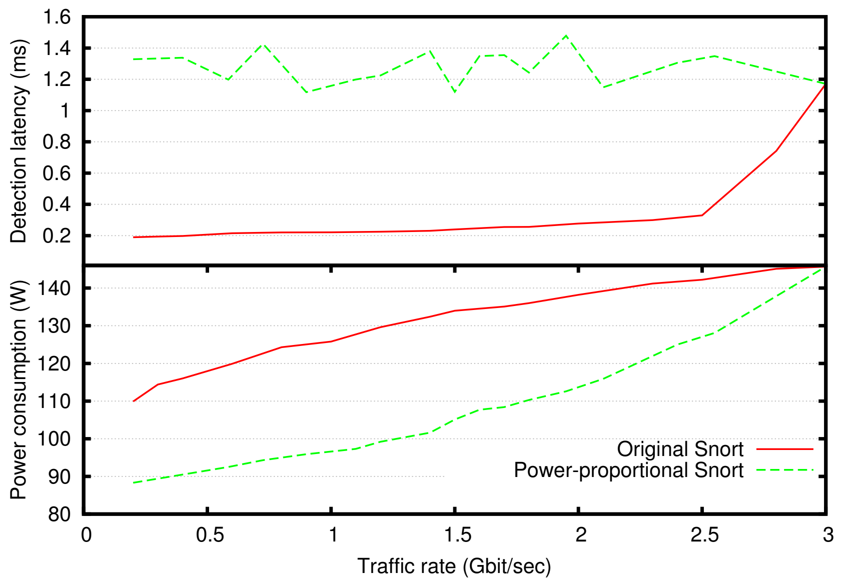


Figure 3.2: Power consumption and detection latency of a straight-forward NIDS versus the original system as a function of traffic rate. A straight-forward power-proportional NIDS consumes less power with higher detection latency.

activating/deactivating cores. The NIDS should be also able to handle short-term overloads, *e.g.*, by spending more energy during these periods, to avoid dropping packets due to full buffers, which results in undetected attacks [25].

A NIDS is based on the underlying packet capturing system to receive packets for processing. The packet capturing system, which relies into the OS kernel, is responsible to pass the incoming packets from NIC to user-level. To tolerate processing spikes or short-term overloads, the packet capturing system is able to store a limited number of packets in memory queues (ring buffers). Modern NICs [11] offer multiple receive queues and are able to distribute the packets among them to allow for efficient multi-core processing. Thus, a packet capturing system with multi-core support [9] uses a separate queue per each core. 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, which is similar to the power-proportional software router:

1. The system starts with a single active core at the minimum frequency while the rest cores are in C6 state.

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 more core uses one more queue.

2.1.2. Else, it increases the frequency of all cores to the next step.

2.2. If queues are filled by less than a *low threshold*:

2.2.2. If the system uses the lowest frequency, 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 when varying the load. Figure 3.2 (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 power consumption of the vanilla system is reduced with the traffic rate up to 24% when processing 200 Mbit/sec, compared with the power consumption at 3 Gbit/sec. However, the power-proportional NIDS adapts much better to the load and reduces significantly the power consumption up to 39% when processing 200 Mbit/sec. This is a 23% improvement on the power consumption compared to the original system.

3.5 Summary

Briefly, this chapter explored the design space in order to build a power-proportional NIDS. We observed that the CPU consumes the larger portion of the energy in these systems. Therefore, a power proportional NIDS should utilize the smallest number of cores and operate at the lowest possible frequency able to process the incoming traffic without any packets drops and with no idle time.

Chapter 4

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 consumption, we would like to explore other aspects related to the performance of a NIDS as well. Thus, in this chapter we turn our attention on the impact of this approach on the detection latency. Detection latency is an important issue for a NIDS operating in passive mode in order to achieve a timely reaction to an attack, *e.g.*, by actively terminating the offending connection. A high latency will make the NIDS reaction pointless. In this chapter we study how the latency is affected when energy is reduced, and we explain the observed behavior with both experimental and analytical evaluation.

4.1 Detection Latency

In our experiments we instrumented Snort to measure the attack detection latency, by subtracting from the time when Snort triggers an alert the timestamp of the packet that contains the attack pattern. The packet's timestamp is set within the packet capturing module before the packet is queued. Figure 3.1(b) shows the detection latency as a function of frequency and number of active cores for 0.6 Gbit/sec traffic. We see a linear increase when frequency is higher than 1.6 GHz and more than 8 cores are used, but we see an exponential increase to the detection latency when the average core utilization exceeds 70%. To better see the relation between power consumption and detection latency we replot the data in Figure 4.1. We see a clear tradeoff: in order to achieve power consumption lower than 100 W, the detection latency should be increased 4–7 times. This trafeoff was not observed in previous works, like the power-proportional software router [23].

Table 3.2 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 3.2 (upper part) shows the detection latency of a power-proportional NIDS, compared to the original system. We see that although it consumes less power

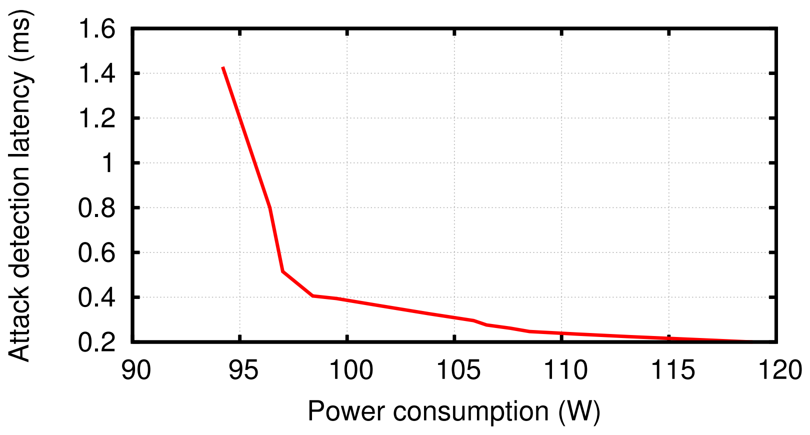


Figure 4.1: The energy-latency tradeoff. Detection latency as a function of power consumption when sending 0.6 Gbit/sec traffic.

for lower traffic loads, this power-proportional NIDS has a significantly higher detection latency at all rates. This is because at all rates this system always selects the frequency and number of cores that lead to high utilization, close to 100%, in order to save energy. As a consequence, the detection latency remains always high.

4.2 Deconstructing Detection Latency

An attack vector may span among multiple successive packets of the same flow. We define detection latency as the time passed from the arrival and capturing of the last packet that contains the attack till the alert generation in the NIDS. Thus, the detection latency is equal to the latency imposed per each attack packet, from the capturing time till it finished 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, (driver, packet capturing system), (ii) *queuing delay*, which is the time that the packet waits in a queue to be delivered for processing, and (iii) *processing time* by the NIDS at user level. We see that the time spent in kernel per packet 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 CPU utilization.

To explore why latency is increased, we measure how much each part contribute to the detection latency as we vary the input traffic rate for few different frequencies. We instrumented Snort to measure the queuing delay per packet, by subtracting the packet's timestamp set by the capturing system from the time that Snort receives the packet, and the packet's processing time by Snort's inspection engine for attack detection. For each packet we reported the respective times to calculate

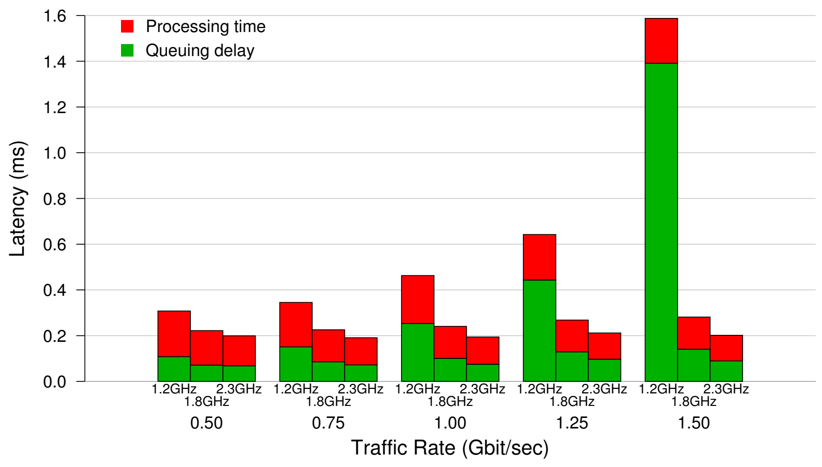


Figure 4.2: Processing time and queuing delay of attack packets as a function of traffic rate when running 12 cores at 1.2 GHz, 1.8 GHz, and 2.3 GHz. The main cause of increased detection latency is higher queuing delay.

the average values.

Figure 4.2 shows the average processing time and queuing delay per each attack packet, which are summed to the detection latency, 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. We observe a higher processing time for attack packets compared to benign packets, which is due to logging and alert generation. We see that for low frequency and high rates, when the system is more utilized, the queuing delay is the main factor of the increased detection latency. For instance, we observe a large increase on the detection latency when processing 1.5 Gbit/sec at 1.2 GHz, which result in the lowest power consumption. In this case, the queuing delay is 7x higher than the processing time for the attack packets. This is because the higher utilization results in a large number of packets waiting at the queue and thus in an exponentially higher queuing delay. On the other hand, the processing time increases normally as we decrease the frequency.

4.3 Delay Analysis

Our results indicate that the main reason for increased latency when reducing the power consumption is the higher CPU utilization that leads to increased packet queuing delays. To better understand how the queuing delay is affected by reduced frequency and reduced 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

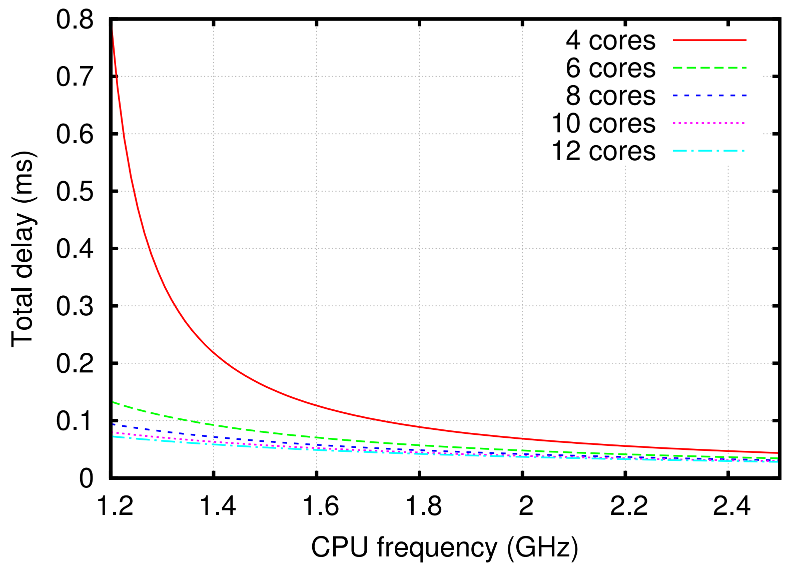


Figure 4.3: Total delay as a function of frequency and number of cores. When using 4 cores to process 75,000 packets per second the delay increases exponentially as we decrease frequency. Total delay approximates the behavior of detection latency.

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 the incoming packets results in an increased arrival rate λ_c per core. Reducing the frequency f results in reduced service rate per core:

$$\mu_c(f) = \frac{\mu_{max}}{c_{max}} \frac{f}{f_{max}} \tag{4.1}$$

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

$$T = \frac{1}{\mu_c(f) - \lambda_c} = \frac{c_{max} \cdot f_{max} \cdot c}{\mu_{max} \cdot f \cdot c - \lambda \cdot c_{max} \cdot f_{max}} \tag{4.2}$$

Respectively, the average queuing delay W is:

$$W = \frac{\rho_c}{\mu_c(f) - \lambda_c} = \frac{\lambda \cdot c_{max}^2 \cdot f_{max}^2}{\mu_{max} \cdot f \cdot (\mu_{max} \cdot f \cdot c - \lambda \cdot f_{max} \cdot c_{max})} \tag{4.3}$$

The packets loss probability is: $P_{loss} = \sum\limits_{k=N}^{\infty} \rho_C^k (1 - \rho_c)$.

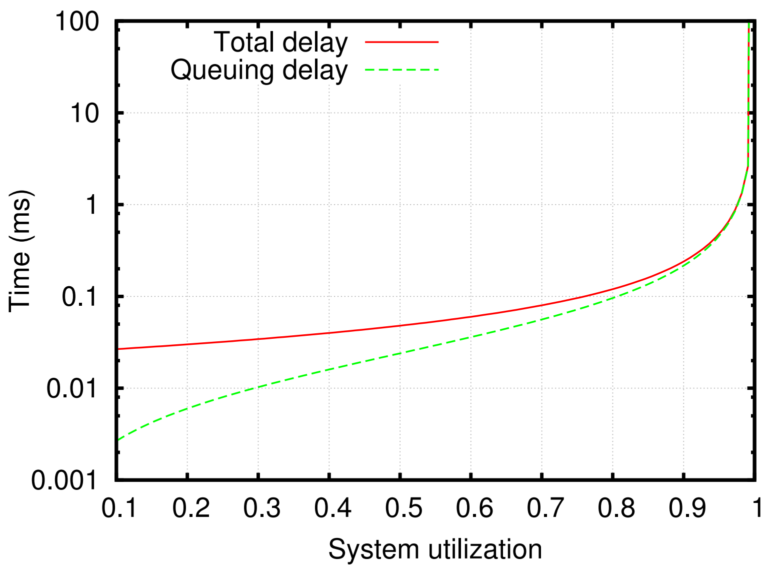


Figure 4.4: Total and queuing delay as a function of system utilization. When utilization exceeds 80%, the queuing delay approaches the total delay and they are both increased exponentially.

Based on equation 4.2, Figure 4.3 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 the maximum service rate $\mu_{max} = 500,000$ 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, the system is less utilized so we see a linear increase on the total delay per packet.

Our results in section 3.2 indicate that we have the lower power consumption when $\rho_c \approx 1$. Figure 4.4 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 λ from $50,000$ till $500,000$, which results in system utilization from 10% to 100% respectively. We use 12 active cores and 2.5 GHz frequency in equations 4.2 and 4.3. The results show that total and queuing delay increase exponentially at high utilization. When utilization is increased above 80%, the queuing delay approaches the total delay and both increase exponentially. It is clear that for high system utilization values the queuing delay becomes the main reason for the increased total delay, as we also observe in section 4.2. 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 ms in this case, the utilization should be kept lower than 50%.

4.4 Summary

Reducing the power consumed in a system its performance is usually degraded. In this chapter we studied the performance of a power-proportional NIDS. Our findings indicate an up to 7x increase of the detection latency, which is an important issue for a NIDS in order to achieve a timely reaction to an attack. The main reason for this increase is the high CPU utilization that leads to exponentially increased queuing delays.

Chapter 5

Solving the Energy-Latency Tradeoff

In this chapter we aim to solve the energy-latency tradeoff using domain-specific knowledge on NIDS. This will enable us to build a NIDS with both low power consumption *and* low detection latency. The key idea of our approach is based on the fact that there is a small percentage of packets with significantly higher probability to contain an attack. These are the first few packets of each connection. Then we propose two alternative approaches to process these packets with low latency, while consuming less power proportional to the workload.

5.1 Identify the Most Important Packets for Detection Latency

Previous works have shown that most attacks are found among the first few packets of each flow [17, 24]. 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 KB of the flow. By 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 [8], the first N packets of each flow correspond to a small percentage of the total traffic. Thus, processing these packets with higher priority and lower latency will result in faster detection for most of the attacks.

We first measure the position of the attack packets within their flows while running Snort with our trace. The trace contains 1796 alerts in the captured traffic and 233 alerts from the attack traces we manually added, as we explained in section 3.1. Figure 5.1 shows the CDF of the rankings of the attack packets detected. We see that 50% of the attacks are found within the first 10 packets of a flow, while the 90% of the attacks are detected in the first 100 packets of their flows.

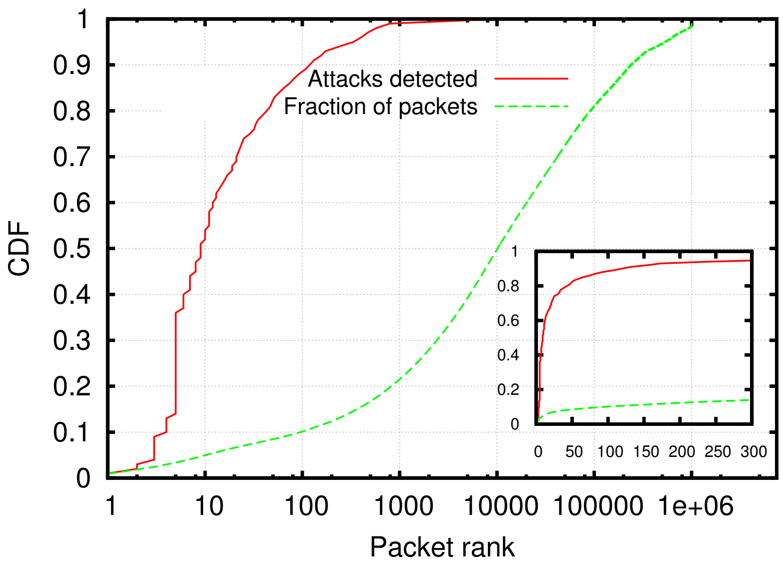


Figure 5.1: CDF of the attack packet ranking and fraction of packets per each rank. A large percentage of the attacks can be detected in a small fraction of the total packets using a flow cutoff value.

Only 1% of the attacks are found beyond the first 800 packets of a flow. Thus, the first few hundred packets 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 these packets from the less important 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 of each flow, if has more packets than the cutoff value.

Figure 5.1 also presents the CDF of the fraction of packets with ranking lower or equal to the corresponding value on the x-axis. This fraction is also the percentage of high-priority packets as a function of the cutoff applied. For instance, 10% of the total packets have ranking lower than 100 in their flows. This means that a cutoff value of 100 packets per flow will classify the 10% of the total packets as high-priority packets, and 90% of the attacks can be detected on them. Even a higher cutoff of 1,000 packets per flow corresponds to just 21% of the total packets, while more than 99% of the attacks can be detected on them.

Since we have identified the most important packets for fast detection, we need to ensure a low latency for these packets when the system enters into a power saving mode and active cores' utilization increases. We propose two alternative techniques to ensure low latency for the high-priority packets: *time sharing* and *space sharing*.

5.2 Time Sharing

The *time sharing* technique uses a typical priority queue scheduling to favor the high-priority packets. Time sharing first classifies packets into their corresponding transport-layer 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 will choose the next packet from the high priority queue. If there is no such packet, a lower priority packet is chosen. However, this priority queue scheduling is non preemptive: when a high-priority packet arrives and a low priority is being processed, the NIDS cannot evict the low-priority packet to serve immediately the high-priority packet. Time sharing follows the same strategy described at section 3.4 to adapt the number of cores and their frequency according to the load.

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.

5.3 Space Sharing

Besides time sharing, we also propose a second approach to reduce the latency of high priority packets: *space sharing.* In *space sharing* we choose to use separate cores for each priority. Thus, we use few dedicated cores for the high-priority packets. In time sharing the cores of a power-proportional NIDS will remain almost fully utilized. This may be a cause of reduced performance due to the non-preemptive priority queue scheduling. On the other hand, in space sharing, we aim to keep the cores that serve the high priority packets less utilized, to ensure a low per-packet latency.

The cores that serve low-priority packets should remain almost fully utilized to allow for reduced power consumption. The increased latency for low-priority packets is less likely to affect the overall detection latency. Fortunately, as Figure 5.1 indicates, the majority of the packets have a low priority. Therefore, most of the cores can be used to serve low-priority packets with high CPU utilization that is necessary to achieve significant energy savings in a power-proportional NIDS.

In order to reduce even more 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 chapter 4 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.*

5.3.1 Flow Migration

The flow migration technique, assisted by advanced features of modern NICs, is used to distribute efficiently the packets into the proper 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, which are the first N packets of each flow, remain for processing into the high priority cores. The low-priority packets are moved to the rest cores using the flow migration technique.

5.3.2 Adaptive Core Management

Another aspect of space sharing is the *adaptive core management.* Space sharing is able to partition the active cores into high-priority and low-priority cores. This partitioning may vary depending on the workload. Space sharing should use the optimum number of high priority cores that keep the core utilization within a desirable range. Using more cores than necessary may increase power consumption, while less cores may increase the latency of high-priority packets. Therefore, we propose the following adaptive core management algorithm, which is a variant of the core/frequency adaptive algorithm we presented in section 3.4:

1. The system starts with one high-priority core and one low-priority core.

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 there are inactive cores, activate another high-priority core.
  - **2.1.2.** Else increase the CPU frequency.
  - **2.1.3.** If maximum frequency is used, reduce flow cutoff (so less packets will be considered as high priority) until it reaches a certain limit.
- **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 CPU frequency.
  - **2.2.2.** If the lowest frequency is used, 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 there are inactive cores, activate another low-priority core.
  - **2.3.2.** Else increase the CPU frequency.

- **2.4.** If low-priority queues are filled by less than a *low-priority down threshold*:
    - **2.4.1.** Reduce the CPU frequency.
    - **2.4.2.** In case of the lowest frequency, deactivate a low-priority core.

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. It is significantly higher than *high-priority up threshold.* We can also control the load of high- and low- priority cores by changing the flow cutoff value, which divides the traffic into high- and low-priority packets. However, decreasing the flow cutoff is not always a good choice, as the probability that an attack occurs in a low-priority packet increases. Since low-priority packets experience a higher latency, the average detection latency in this case may also increase.

5.4 Summary

To summarize, in this section we resolved the energy-latency tradeoff by identifying the most important packets for the detection latency. These are the first packets of each flow and they should to be processed with lower latency. Finally, we proposed two approaches to achieve low latency and power consumption in NIDS: Time Sharing and Space Sharing that use priority queue scheduling and dedicated cores with lower utilization respectively, for the most important packets.

Chapter 6

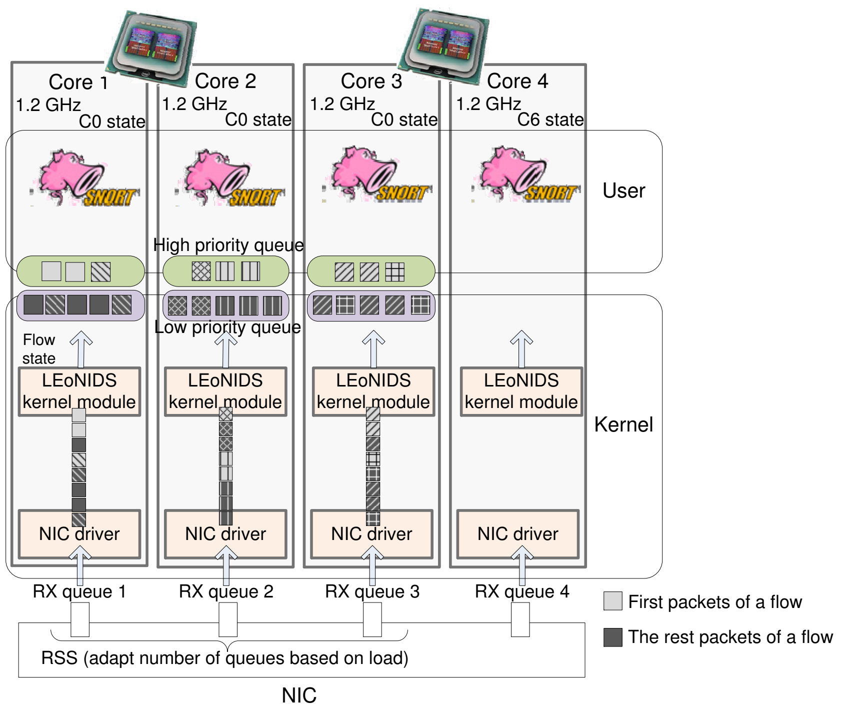
Implementation

Based on the two alternative approaches this chapter describes the implementation of LEoNIDS: a NIDS architecture that offers both low power consumption, proportionally to the load, and low detection latency. Figure 6.1 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 the Snort NIDS [31].

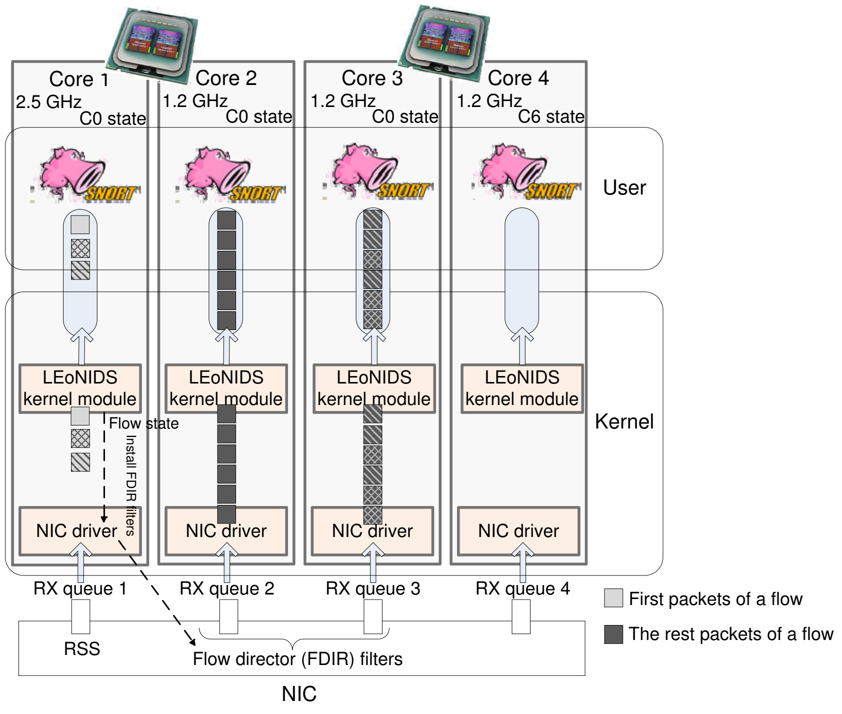
We implemented the online frequency adaptation and core management algorithm within the packet capturing subsystem as a Linux kernel loadable module. Both time sharing and space sharing are implemented within this module. The module runs as a protocol handler and processes all captured packets. It is 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 [12] that is supported by the NIC, in case of time sharing, or with a dynamic load balancing scheme, in case of space sharing. Space sharing uses the flow director filters (FDIR) to manually define the core that will serve each flow. The FDIR filters are used in conjunction with RSS.

RSS sets the number of queues according to the number of the available cores that will process the packets. LEoNIDS runs an instance of the detection system that processes the packets on each core. Each queue is assigned to one core and the traffic is split across the queues. The traffic distribution is based on the 5-tuple (source and destination IP, source and destination port and protocol) of the packet header.

We use the PF_RING API [9] to deliver packets at user-level through memory mapped buffers, and a libpcap [20] wrapper library. Then we link Snort with this user-level library, instead of the original libpcap. In the rest of this chapter we give more details about the implementation of time sharing and space sharing.



(a) LEoNIDS with time sharing



(b) LEoNIDS with space sharing

Figure 6.1: The LEoNIDS architecture with time sharing and space sharing.

6.1 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 according to the cutoff value. 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 this is queue is empty, it reads the next packet from the low priority queue. This packet is then delivered to Snort for processing.

The algorithm described in 3.4 is implemented within the packet capturing subsystem. It monitors the packet queues per each core and properly adapts the number of active cores and the CPU frequency. Cores are woken up or put to sleep and the CPU frequency is increased or decreased according to the thresholds.

6.2 Space Sharing

In space sharing, we use few dedicated cores to process the most important packets with reduced latency. Thus, only the first N packets of each flow are processed by these cores, where N is the per-flow cutoff value, and these cores operate with low utilization to avoid increased queuing delays. We aim to keep the utilization of these cores between 30%–50%, which results in relatively low queuing delays as we see in chapter 4. Based on the current queue utilization we properly adapt the number of high-priority cores and the CPU frequency.

To implement space sharing we first modify the RSS to split all the packets only to the high-priority cores. Then, these cores classify packets into flows. When a flow exceeds the specified cutoff size, an FDIR filter is added to the NIC in order to move the processing of this flow to a low-priority core (flow migration). The low-priority core is chosen in a round-robin fashion. Each flow that exceeds the cutoff value moves from one core to another only once, so the cache locality is not significantly affected. 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 [11] 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.

6.3 Adapting the Number of Active Cores

The RSS uses a redirection table to distribute the incoming packets to the available cores. As shown in Figure 6.2, after a packet arrival a hash function on its 5-tuple is computed and the result of the function is used as an index in the redirection table. The entries of this table are the identities of the queues that should be used

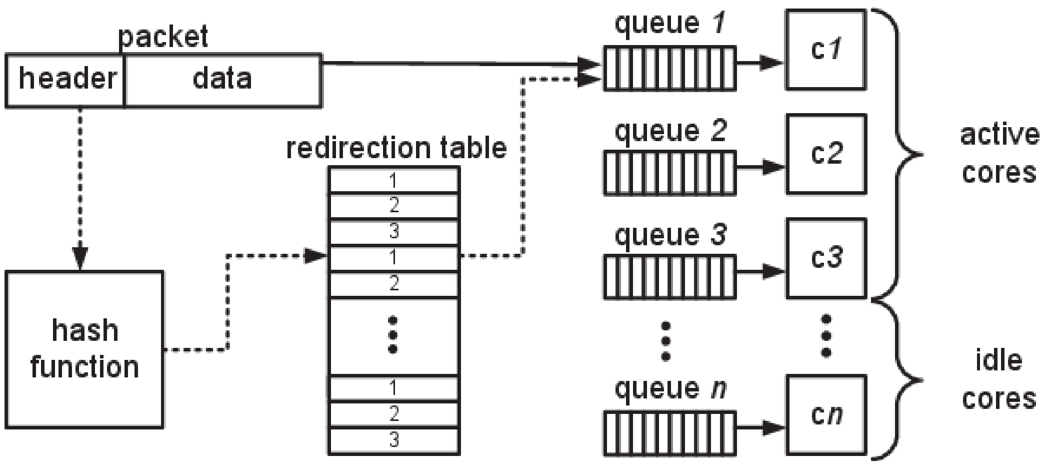


Figure 6.2: Redirection table and NIC queues

to store and forward the packet to the dedicated core. Each queue is assigned to one core and an interrupt is raised at the packet's arrival in order to be routed to the core.

The queues can only be defined at startup of the NIC-driver and they can only change after a reset of the card driver. Therefore, before LEoNIDS starts all queues, one per each core, are set. In order to keep cores inactive we configure the redirection table such as to maintain its entries for queues assigned only to active cores. The 128-byte size of the redirection table permits fast modifications. While there is no packet reception by a queue there is no interrupt at the corresponding core. During this period the core remains in idle state (C6 low power state).

When a queue is filled with more than the threshold, we modify the redirection table in order to activate an additional core. The entries need to include the identity of the queue assigned to the new core. At the arrival of the first packet to a newly added queue, an interrupt wakes up the corresponding core, which then starts to process and route packets to the LEoNIDS instance running on this core. A respective process is followed to deactivate a core. The redirection table is reconfigured to exclude one queue from the traffic distribution and when the queue becomes empty the core enters the idle state.

Chapter 7

Experimental Evaluation

In this chapter we experimentally evaluate the two alternative approaches. We first evaluate each technique separately to find out the optimal cutoff values, and then we compare the original system with PF_RING and Snort, the straight-forward power-proportional system and LEoNIDS with time sharing and space sharing. In our experiments we use the same experimental environment with Section 3.1.

7.1 Time Sharing

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 the low-priority packets, which experience a much higher delay, increases with a small cutoff. To find out the optimal cutoff for time sharing we vary the cutoff values from 50 to 50,000 packets per flow while sending traffic at three different rates.

Figure 7.1 shows that in all cases the lower latency is observed for cutoff value 500 packets per flow. Using this cutoff value, 97% of the attacks reside into the high-priority packets. Lower or higher values lead to higher detection latency. Lower cutoff values result in more attacks found in low-priority packets with increased detection latencies. With higher cutoff values high priority queues contain more benign packets leading to higher queuing latencies for the larger fraction of high priority packets.

To better understand the detection latency we observe with time sharing, we study how the packet latency changes with the cutoff for high- and low-priority packets and priority. Figure 7.2 presents the processing and queuing delay for high- and low-priority packets as a function of the cutoff value when sending at 1.0 Gbit/sec. We see that the low-priority packets experience up to 49.2x higher latency than the high-priority packets. The highest difference is observed for the smallest cutoff of 50 packets per flow. As the cutoff increases, the latency of low-priority packets decreases significantly until the value of 750 packets while it increases for higher cutoff values. It is firstly decreasing because the fraction of

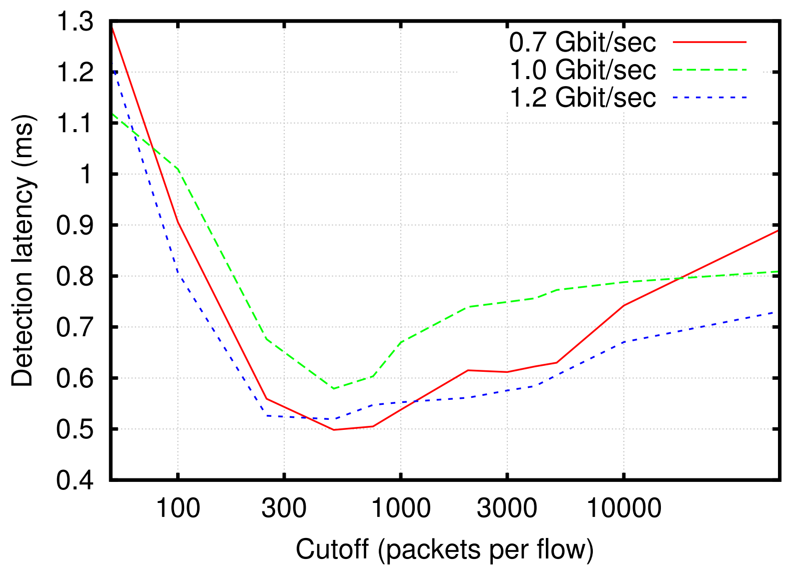


Figure 7.1: Detection latency as a function of cutoff for three different rates when using time sharing. In all cases we observe the lowest latency for a cutoff value of 500 packets per flow.

low-priority packets is respectively decreasing compared with the fraction of high-priority packets leading to lower utilization in low-priority queues. For cutoff values greater than 750 packet it is increasing because low-priority packets are waiting for a large number of high-priority packet to be firstly processed. The latency of high-priority packets indicates a steady rise as the cutoff increases due to the higher number of packets arriving in high-priority queues. Therefore, both low and high cutoff values lead to higher detection latencies. Moreover, however using a cutoff of 750 or 1000 packets per flow result in slightly lower queuing latency for low-latency packets, the optimal cutoff is 500 packets, where the high-priority packets experience lower latency than using higher cutoff values.

7.2 Space Sharing

We now turn our attention to space sharing. We aim to find out the optimal cutoff value for this approach. The most significant difference from time sharing is that now dedicated cores are used for high-priority packets, so high- and low-priority packets are processed simultaneously.

Figure 7.3 depicts the detection latency as a function of the cutoff when using space sharing to process 1.0 Gbit/sec traffic. We see that the optimal cutoff for space sharing is also around 500 packets per flow for the same reasons as in case of the time sharing: small cutoff values result in more attacks detected in low-priority packets, while a large cutoff results in higher utilization in high-priority cores.

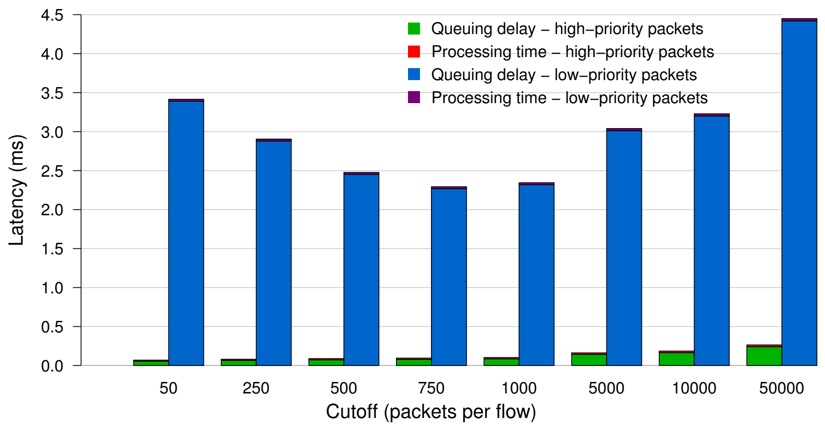


Figure 7.2: Latency breakdown for high- and low-priority packets with time sharing as a function of cutoff when we send 1.0 Gbit/sec traffic. Low-priority packets experience up to 49.2x higher latency than high-priority packets.

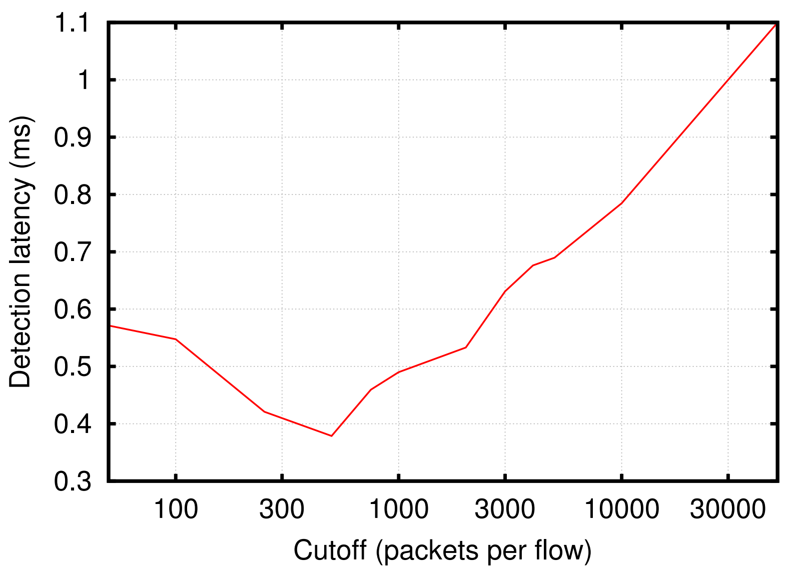


Figure 7.3: Detection latency as a function of cutoff when using space sharing to process 1.0 Gbit/sec. The optimal cutoff for space sharing is around 500 packets per flow.

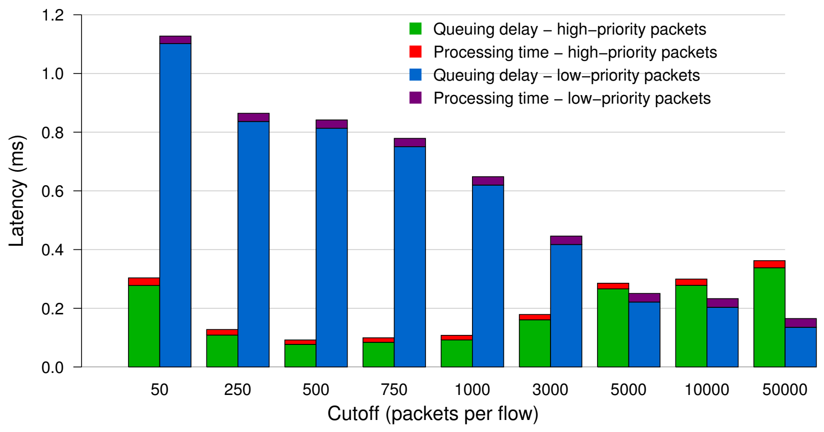


Figure 7.4: Latency breakdown for high- and low-priority packets with space sharing as a function of cutoff. Both high- and low-priority packets experience lower latency in space sharing compared with time-sharing

To better understand the optimal cutoff and explain the lower detection latencies with space sharing compared to time sharing we plot the processing and queuing delay for low- and high-priority packets when space sharing processes 1.0 Gbit/sec. As Figure 7.4 illustrates, we see much lower latency for the low-priority packets in space sharing compared to time sharing, which also decreases as the cutoff value increases. This is because their are processed in parallel with high-priority packets and their number is lower when using higher cutoff values. Moreover, we observe lower latency for high-priority packets compared with the respective latency of time sharing due to the ability to keep high-priority core less utilized in space sharing. The high latencies at high-priority cores for small cutoff values in space sharing is the result of the increased CPU utilization because of the often FDIR filter establishments.

7.3 Comparison of all Approaches

Finally, we compare all approaches in terms of latency and energy efficiency. We compare time sharing with space sharing, and both of them with the original system and with the straight-forward power-proportional NIDS we described in chapter 3. We use a cutoff of 500 packets per flow in both time and space sharing, which was found to give the best results. Figure 7.5 shows the power consumption and latency of all approaches as a function of traffic rate.

We see that LEoNIDS with time sharing and space sharing consume approximately the same power as the power-proportional NIDS, which is significantly lower than the power consumption of the original Snort. Despite the lower consumption, LEoNIDS with both approaches achieve a significantly lower detection latency than the power-proportional NIDS, closer to the latency of the original system that uses

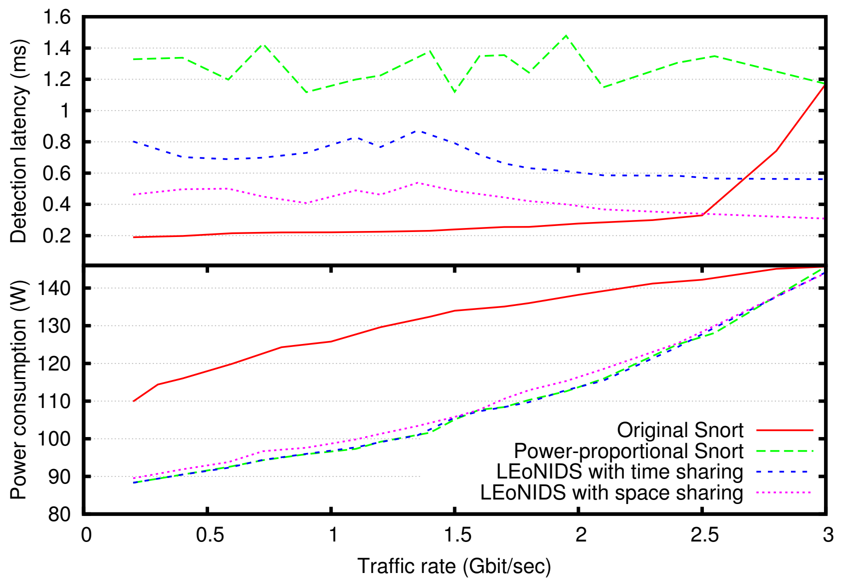


Figure 7.5: Power consumption and detection latency of all approaches as a function of the traffic rate. LEoNIDS with space sharing consumes approximately the same power as the other power-proportional approaches, but with significantly lower detection latency.

the highest frequency.

We observe that space sharing performs quite better than time sharing: although both consume approximately the same amount of power, space sharing achieves more than 40% lower detection latency. This is due to the lower latency that space sharing experience in high- and low-priority packets. The attacks reside in low-priority packets are the 3% of the total attacks and experience 66% lower latency in space sharing than in time sharing. Although, we can use higher cutoff values to reduce these attacks, the increased latency of high-priority packets lead to higher detection latency. Another reason is the non-preemptive priority queues used in time sharing: the low-priority packets interfere with high-priority packets. When a high-priority packet arrives, a low-priority packet may be processed. Then, the high-priority packet waits in the queue for a low-priority packet. 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 74% lower latency than the straightforward 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 in the first packets of each flow at these rates by space sharing where the original system has a high CPU utilization. Since the original system does not give priority to these packets, it experiences a higher detection latency compared to both time and space sharing at high traffic rates where all approaches result in an almost fully utilized system.

Chapter 8

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. Since NIDS are usually overprovisioned to operate at the maximum link capacity, while these links are usually much less utilized, there are significant opportunities to reduce power consumption. However, while building a power-proportional NIDS, we identified an energy-latency tradeoff: the reduced power consumption results in a significant increase on the detection latency, which impedes a timely automatic reaction of the NIDS to the incoming attacks. By analyzing the detection latency we showed that the main reason for this increase is the higher packet queuing delays imposed by the high core utilization.

To resolve this tradeoff we presented the design, implementation, and evaluation of LEoNIDS, a NIDS that resolves the energy-latency tradeoff. The key idea of our approach is that the first few packets of each flow should be processed with lower latency for a fast detection. Indeed, we show that there is a higher probability for these packets to carry an attack. Then we proposed two alternative techniques: time sharing and space sharing. Time sharing uses a typical priority queue scheduling, while space sharing uses dedicated cores with lower utilization for the processing of high-priority packets. Our experimental evaluation shows that space sharing performs better than time sharing. Overall, LEoNIDS with space sharing consumes significantly less power, proportionally to the load, and detects attack with low latency.

Bibliography

[1] Cisco green research symposium. `http://www.cisco.com/web/about/ac50/ac207/crc_new/events/symposium_details.html`.

[2] Sourcefire vulnerability research team (vrt). `http://www.snort.org/vrt/`.

[3] Y. Afek, A. Bremler-Barr, Y. Harchol, D. Hay, and Y. Koral. Mca2: multi-core architecture for mitigating complexity attacks. In *Proceedings of the eighth ACM/IEEE symposium on Architectures for networking and communications systems (ANCS)*, 2012.

[4] J. Bickford, H. A. Lagar-Cavilla, A. Varshavsky, V. Ganapathy, and L. Iftode. Security versus energy tradeoffs in host-based mobile malware detection. In *Proceedings of the 9th international conference on Mobile systems, applications, and services (MobiSys)*, 2011.

[5] J. Chabarek, J. Sommers, P. Barford, C. Estan, D. Tsiang, and S. Wright. Power awareness in network design and routing. In *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE*, 2008.

[6] Q. Deng, D. Meisner, L. Ramos, T. F. Wenisch, and R. Bianchini. Memscale: active low-power modes for main memory. In *Proceedings of the sixteenth international conference on Architectural support for programming languages and operating systems (ASPLOS)*, 2011.

[7] H. Dreger, A. Feldmann, V. Paxson, and R. Sommer. Operational experiences with high-volume network intrusion detection. In *Proceedings of the 11th ACM Conference on Computer and communications security (CCS)*, 2004.

[8] W. Fang and L. Peterson. Inter-as traffic patterns and their implications. In *Global Telecommunications Conf.*, Dec 1999.

[9] F. Fusco and L. Deri. High speed network traffic analysis with commodity multi-core systems. In *Proceedings of the 10th annual Conference on Internet measurement (IMC)*, 2010.

[10] M. Gupta and S. Singh. Greening of the internet. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM)*, 2003.

[11] Intel. 82599 10 GbE Controller Datasheet. http://download.intel.com/design/network/datashts/82599_datasheet.pdf, October 2011.

[12] Intel Server Adapters. Receive side scaling on Intel Network Adapters. http://www.intel.com/support/network/adapter/pro100/sb/cs-027574.htm.

[13] M. F. Iqbal and L. K. John. Efficient traffic aware power management in multicore communications processors. In *Proceedings of the eighth ACM/IEEE symposium on Architectures for networking and communications systems (ANCS)*, 2012.

[14] V. Kontorinis, L. Zhang, B. Aksanli, J. Sampson, H. Homayoun, E. Pettis, D. Tullsen, and T. Simunic Rosing. Managing distributed ups energy for effective power capping in data centers. In *2012 39th Annual International Symposium on Computer Architecture (ISCA)*, 2012.

[15] C. Kruegel, F. Valeur, G. Vigna, and R. Kemmerer. Stateful intrusion detection for high-speed networks. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2002.

[16] J. Kuang and L. Bhuyan. Optimizing throughput and latency under given power budget for network packet processing. In *Proceedings of the 29th conference on Information communications (INFOCOM)*, 2010.

[17] T. Limmer and F. Dressler. Improving the Performance of Intrusion Detection using Dialog-based Payload Aggregation. In *14th IEEE Global Internet Symposium (GI)*, 2011.

[18] S. Lindsey and C. Raghavendra. Pegasis: Power-efficient gathering in sensor information systems. In *IEEE Aerospace Conference Proceedings*, 2002.

[19] G. Lu, B. Krishnamachari, and C. Raghavendra. An adaptive energy-efficient and low-latency mac for tree-based data gathering in sensor networks. *Wireless Communications and Mobile Computing*, 2007.

[20] S. McCanne, C. Leres, and V. Jacobson. libpcap. Lawrence Berkeley Lab., Berkeley, CA. (http://www.tcpdump.org/).

[21] D. Meisner, B. T. Gold, and T. F. Wenisch. Powernap: eliminating server idle power. In *Proceedings of the 14th international conference on Architectural support for programming languages and operating systems (ASPLOS)*, 2009.

[22] S. Nedevschi, L. Popa, G. Iannaccone, S. Ratnasamy, and D. Wetherall. Reducing network energy consumption via sleeping and rate-adaptation. In *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2008.

[23] L. Niccolini, G. Iannaccone, S. Ratnasamy, J. Chandrashekar, and L. Rizzo. Building a power-proportional software router. In *Proceedings of the 2012 USENIX conference on Annual Technical Conference (ATC)*, 2012.

[24] A. Papadogiannakis, M. Polychronakis, and E. P. Markatos. Improving the accuracy of network intrusion detection systems under load using selective packet discarding. In *Proceedings of the Third European Workshop on System Security (EUROSEC)*, 2010.

[25] A. Papadogiannakis, M. Polychronakis, and E. P. Markatos. Tolerating overload attacks against packet capturing systems. In *Proceedings of the USENIX Annual Technical Conference (ATC)*, 2012.

[26] 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 *Proceedings of the 7th ACM european conference on Computer Systems (EuroSys)*, 2012.

[27] V. Paxson. Bro: A system for detecting network intruders in real-time. *Computer Networks*, 1999.

[28] V. Paxson, R. Sommer, and N. Weaver. An architecture for exploiting multicore processors to parallelize network intrusion prevention. In *Proceedings of the IEEE Sarnoff Symposium*, 2007.

[29] A. Pesterev, J. Strauss, N. Zeldovich, and R. T. Morris. Improving network connection locality on multicore systems. In *Proceedings of the 7th ACM european conference on Computer Systems (EuroSys)*, 2012.

[30] M. Polychronakis, K. G. Anagnostakis, and E. P. Markatos. An empirical study of real-world polymorphic code injection attacks. In *Proceedings of the 2nd USENIX Workshop on Large-scale Exploits and Emergent Threats (LEET)*, 2009.

[31] M. Roesch. Snort: Lightweight intrusion detection for networks. In *Proceedings of the 1999 USENIX LISA Systems Administration Conference*, 1999.

[32] L. Schaelicke, K. Wheeler, and C. Freeland. SPANIDS: a scalable network intrusion detection loadbalancer. In *Proceedings of the 2nd Conference on Computing frontiers (CF)*, 2005.

[33] 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 *Proceedings of the eighteenth international conference on Architectural support for programming languages and operating systems (ASPLOS)*, 2013.

[34] D. Tsirogiannis, S. Harizopoulos, and M. A. Shah. Analyzing the energy efficiency of a database server. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, 2010.

[35] 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 *Proceeding of the 10th International Symposium on Recent Advances in Intrusion Detection*, 2007.

[36] G. Vasiliadis, S. Antonatos, M. Polychronakis, E. P. Markatos, and S. Ioannidis. Gnort: High performance network intrusion detection using graphics processors. In *Proceedings of the 11th International Symposium on Recent Advances in Intrusion Detection (RAID)*, 2008.

[37] C. Yoon, D. Kim, W. Jung, C. Kang, and H. Cha. Appscope: Application energy metering framework for android smartphone using kernel activity monitoring. In *Proceedings of the USENIX Annual Technical Conference (ATC)*, 2012.

[38] Y. Yu, B. Krishnamachari, and V. Prasanna. Energy-latency tradeoffs for data gathering in wireless sensor networks. In *Proceedings of the 23rd conference on Information communications (INFOCOM)*, 2004.