UNIVERSIDAD CARLOS III DE MADRID

TESIS DOCTORAL

Optimization of Energy Efficiency in Data and Web Hosting Centers

Autor:	Angelos Chatzipapas, Univesidad Carlos III de Madrid
Director:	Vincenzo Mancuso, IMDEA Networks Institute
Tutor:	Rubén Cuevas Rumín, Universidad Carlos III de Madrid

DEPARTAMENTO DE INGENIERÍA TELEMÁTICA

Leganés (Madrid), septiembre de 2016

UNIVERSIDAD CARLOS III DE MADRID

PH.D. THESIS

OPTIMIZATION OF ENERGY EFFICIENCY IN DATA AND WEB HOSTING CENTERS

Author: Angelos Chatzipapas, Univesidad Carlos III de Madrid
Director: Vincenzo Mancuso, IMDEA Networks Institute
Tutor: Rubén Cuevas Rumín, Universidad Carlos III de Madrid

DEPARTMENT OF TELEMATIC ENGINEERING

Leganés (Madrid), September 2016

Optimization of Energy Efficiency in Data and Web Hosting Centers

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy

Prepared by
Angelos Chatzipapas, Univesidad Carlos III de Madrid

Under the supervision of
Vincenzo Mancuso, IMDEA Networks Institute
Rubén Cuevas Rumín, Universidad Carlos III de Madrid

Departamento de Ingeniería Telemática, Universidad Carlos III de Madrid

---

Date: septiembre, 2016

Web/contact: angelos.chatzipapas@imdea.org

This work has been partially supported by IMDEA Networks Institute and the Greek State Scholarships Foundation.

TESIS DOCTORAL

OPTIMIZATION OF ENERGY EFFICIENCY IN DATA AND WEB HOSTING CENTERS

Autor:	Angelos Chatzipapas, Univesidad Carlos III de Madrid
Director:	Vincenzo Mancuso, IMDEA Networks Institute
Tutor:	Rubén Cuevas Rumín, Universidad Carlos III de Madrid

Firma del tribunal calificador:

Presidente:

Vocal:

Secretario:

Calificación:

Leganés, de de

Acknowledgements

First of all I would like to say a big thank you to my mentor and friend Vincenzo Mancuso. He offered his multi-layer support with all means he could offer. He helped me for the internship, he allowed me to spend enough time with my family, he was reviewing all my works and he was always giving me valuable feedback. I will never forget my first months when we were solving equations together until late afternoon. This is an amazing support and more than what a student needs from his supervisor. It was an absolute honour to be his student and if there was not his patience and support most probably I would have never finished this dissertation.

Moreover I would like to thank my parents, Alexandra and Petros, for their support during this PhD journey. Most probably they were more anxious than me. It was a long process with lots of difficulties and they were always there for support, encouragement and company in my conference trips. Thank you dad for coming with me in US. It was the moment I learned about my babies and I needed you. Sometimes luck or God brings people together for some reason. Dad, I still remember your face in the airplane when I told you the news.

I do not forget my daughters, Alexandra and Despoina, with whom we spent many hours together, writing papers and the dissertation itself (maybe sometimes they were sleeping in my hug but usually they were watching, sit on my legs). I cannot understand what they liked. However, they were putting pressure to finish it, possibly more than Vincenzo ("why you are not writing Dad", "finish this quickly and let's go to play").

Also I want to thank my friends and "co-students" on this journey. Without Christian, Elli, Evgenia, Foivos (alphabetical order guys) and among others the "Quique", "Vinoteca" and "Gallego" nights would not have made this Ph.D. a wonderful and unforgetable experience.

Abstract

This thesis tackles the optimization of energy efficiency in data centers in terms of network and server utilization.

For what concerns networking utilization the work focuses on Energy Efficient Ethernet (EEE) - IEEE 802.3az standard - which is the energy-aware alternative to legacy Ethernet, and an important component of current and future green data centers. More specifically the first contribution of this thesis consists in deriving and analytical model of gigabit EEE links with coalescing using M/G/1 queues with sleep and wake-up periods. Packet coalescing has been proposed to save energy by extending the sojourn in the *Low Power Idle* state of EEE. The model presented in this thesis approximates with a good accuracy both the energy saving and the average packet delay by using a few significant traffic descriptors. While coalescing improves by far the energy efficiency of EEE, it is still far from achieving energy consumption proportional to traffic. Moreover, coalescing can introduce high delays. To this extend, by using sensitivity analysis the thesis evaluates the impact of coalescing timers and buffer sizes, and sheds light on the delay incurred by adopting coalescing schemes. Accordingly, the design and study of a first family of dynamic algorithms, namely measurement-based coalescing control (MBCC), is proposed. MBCC schemes tune the coalescing parameters on-the-fly, according to the instantaneous load and the coalescing delay experienced by the packets. The thesis also discusses a second family of dynamic algorithms, namely NT-policy coalescing control (NTCC), that adjusts the coalescing parameters based on the sole occurrence of timeouts and buffer fill-ups. Furthermore, the performance of static as well as dynamic coalescing schemes is investigated using real traffic traces. The results reported in this work show that, by relying on run-time delay measurements, simple and practical MBCC adaptive coalescing schemes outperform traditional static and dynamic coalescing while the adoption of NTCC coalescing schemes has practically no advantages with respect to static coalescing when delay guarantees have to be provided. Notably, MBCC schemes double the energy saving benefit of legacy EEE coalescing and allow to control the coalescing delay.

For what concerns server utilization, the thesis presents an exhaustive empirical characterization of the power requirements of multiple components of data center servers. The characterization is the second key contribution of this thesis, and is achieved by devising different experiments to stress server components, taking into account the multiple available CPU frequencies and the presence of multicore servers. The described experiments, allow to measure energy consumption of server components and identify their optimal operational points. The study proves that the curve defining the minimal CPU power utilization, as a function of the load expressed in Active Cycles Per Second, is neither concave nor purely convex. Instead, it definitively shows a super-linear dependence on the load. The results illustrate how to improve the efficiency of network cards and disks. Finally, the accuracy of the model derived from the server components consumption characterization is validated by comparing the real energy consumed by two Hadoop applications - PageRank and WordCount - with the estimation from the model, obtaining errors below 4.1%, on average.

Table of Contents

List of Tables

List of Figures

Part I

Background

Chapter 1

Introduction

During the past years a lot of effort has been invested to increase processing, communication, switching speed and data storage with little effort to optimize the power consumption. According to [34], about 14 TWh were consumed in 2005 by the telecom core network in EU-25[1] and the yearly consumption is expected to increase to about 30 TWh by 2020. Similarly, data center infrastructures have become one of the largest and fastest growing consumers of electricity globally, surpassing the aviation industry in terms of energy consumption [19]. To put this into perspective, in 2013, U.S. data center's electricity consumption (91 TWh) was sufficient to power twice the number of all the households in New York City [73]. As a result, ICT energy consumption accounts for 3% of the global consumption and has an annual increase of $\approx$4.3% [90].

Although this power consumption is useful for the human beings, it is also potentially harmful for our environment since it produces an augmented amount of CO_2 emissions and highly contributes to the greenhouse effect. The current threat to the environment could turn into a much more serious threat in the near future, since there is a growing demand of new generation devices that require connection to the Internet (Internet Of Things). In addition, existing network connected devices are now increasing their bandwidth demands (e.g., Web servers, databases, etc.). Indeed, the Internet traffic might grow with the number of data centers in the network and the number of users that demand higher amounts of traffic such as bigger files, videos, TV over IP etc. Hence, as the data traffic demand rises, especially in developing countries, more and more energy consumption is expected for networking and data centers. Consequently, there is a growing interest to improve energy efficiency in networking and data center's design, with obvious environmental and financial motives.

In order to protect the environment and obtain lower service cost, Internet Service Providers and Network Operators are currently deploying new strategies to reduce energy consumptions. A first approach towards building greener ICT was the development of *energy proportional* computing and networking infrastructures [30, 57]. This effort took advantage of energy efficient hardware, like CPU voltage/frequency scaling and sleep states, low-power Ethernet and power-

[1]The first 25 countries that joined the European Union.

efficient OS-level resource management (e.g. on demand Linux governor and PowerNap [32]). However, even at low utilization loads, in the order of 10%, both the network and the server power consumption can reach up to 50% of its peak demand [36], allowing room for further improvement.

To address this, the community additionally has proposed workload consolidation techniques which: (i) keep all infrastructure always on but optimize trafic and task allocations to minimize the number of low-utilized network links and machines, e.g. traffic engineering techniques and Facebook autoscale [95], or (ii) turn-off part of the infrastructure and allocate traffic or tasks to fewer but highly-utilized links or servers correspondingly.

In this context, this thesis investigates on the characterization of servers and the energy they consume and on the recently approved Energy Efficient Ethernet (EEE) standard for power saving in Local Area Networks.

Indeed, in order to obtain full benefit of the aforementioned energy-efficient techniques, it is crucial to have a good characterization of servers in the data center, as a function of the utilization of the server's components. That is, it is necessary to know and understand the energy and power consumption of servers and how this changes under the different configurations. There is a large body of literature on characterizing servers' energy and power consumption. However, the existing literature does not jointly considers phenomena like the irruption of multicore servers and dynamic voltage and frequency scaling (DVFS) [93], which are key to achieve scalability and flexibility in the architecture of a server. With these new parameters, more variables come into play in a server configuration. Learning how to deal with these new parameters and how they interact with other variables is important since this may lead to larger savings.

To further reduce energy consumption, research has proposed workload consolidation algorithms which concentrate computation into a subset of the data center infrastructures. Numerous studies leverage live virtual machine (VM) migration, a modern virtualization functionality which allows seamless relocation of VMs between physical hosts, with relatively short down-times. In most cases, a consolidation strategy is encoded into a VM placement algorithm that maximize energy savings while fulfilling a minimum guaranteed level of performance, expressed in the form of service-level-agreements (SLA). The evaluation of the proposed approximation algorithms is typically based on custom simulation frameworks [14, 41] or small-scale testbeds [35, 76, 89].

Moreover, it has been traditionally considered that the CPU is responsible for most of the power being consumed in a server, and that this power increases linearly with the load. Although the power consumed by the CPU is significant, I believe that the power incurred by other elements of the server, like disks and NICs (Network Interface Cards) are not negligible, and have to be taken into account. Moreover, I believe that the assumption that CPU power consumption depends linearly on the load in a server may be too simplistic, especially when the server has multiple cores and may operate at multiple frequencies. In fact, even the way load is expressed has to be carefully defined (e.g., it cannot be defined as a proportion of the maximal computational capacity of the CPU, since this value changes with the operational frequency). Therefore, more complex/complete models for the power consumed by a server are necessary. In order to be consistent, these models have to be based on empirical values. However, I found that there is a lack of empirical work studying servers energy behavior.

In this thesis I partially fill this void of empirical analyses of server power consumption by proposing a measurement-based characterization —which is the first of its kind— of the energy consumption of a server components with DVFS and multiple cores. I evaluate here different server machines and evaluate what is the contribution to their power consumption of the CPU, hard drive disk, and network card (NIC). My results support, for instance, my belief that more complex models than linear ones are required for CPU power consumption. From the measurements obtained from the servers I evaluate, I propose a holistic energy consumption characterization, that accounts for the power consumed by CPU, disk, and NIC. My approach captures the influence of the processing frequency and the multiple cores, not only to the CPU power consumption, but also to that of disk input/output (I/O) and NIC activity.

Furthermore, legacy Ethernet is a power-unaware standard which consumes a constant amount of power independently from the actual traffic flowing through the wires. However, low speed Ethernet cards consume about 200 mW, which is not a significant consumption considering that a server or a home PC consumes tens to thousands of Watts. Therefore, so far Ethernet power saving strategies did not rise the interest of researchers and developers, due to the irrelevance of potential savings for low speed connections. In contrast, new high speed Gigabit interface cards consume a few tens of Watts [85] which makes reasonable the introduction of a power saving mechanism. In fact, taking into account that the amount of Web Hosting Centers and server farms has been extremely increased due to the new trends and services (YouTube, Facebook, Twitter etc.), there are now billions of running interfaces that consume a constant amount of power. Actually, more than 20% of the energy consumption in data centers is due to the network operation, which establishes network as the second biggest energy consumer in data centers [8]. While high-speed Ethernet cards constantly absorb a considerable part of a server's consumption—e.g., 10 $Gbps$ cards consume $\sim$15 W [85]—recent studies have shown that network links are underutilized: $\sim$40% are "comatose" and another $\sim$40% of the links are loaded no more than 10% [15]. Hence the need to introduce a network-wide energy saving mechanism. Therefore, a new power aware Ethernet standard (standardized late 2010) was introduced to minimize the power consumption of the links when low traffic is present, namely IEEE 802.3az, or EEE [47]. Indeed, according to [29], the authors estimate significant reductions by EEE of about $4\ TWh$ per year over one billion devices.

While some effort has been put in understanding the behavior of EEE links where power saving can be activated independently in each traffic direction, e.g., [42, 58, 64], in this thesis, I am the first to present an analytical model for bidirectional EEE links, e.g., EEE links in which power saving operations can only be activated when there is no traffic in both link directions. The latter (namely, the *bidirectional EEE case*) is a very relevant case, since the EEE standard adopts this bidirectional behavior for 1000Base-T cards, which are the most commonly adopted and diffused gigabit network cards, as of today. However, it is known that EEE underperforms even under low traffic conditions due to LPI transitioning delays [22, 79] and therefore more advanced solutions are needed. Notably, packet coalescing techniques have been proposed to boost EEE performance when traffic is not scarce and packet arrivals have short spacing. The basic idea behind coalescing is to aggregate packets in a buffer of limited size until either the buffer is full or a timeout expires, thus improving the energy proportionality of EEE. The cost of such coalescing techniques consists in additional queueing delay for the packets.

1.1. Main Results and Contributions

The main contributions of this thesis can be outlined as follows:

- I analytically model the behavior of gigabit EEE links with coalescing using M/G/1 queues with sleep and wake-up periods.

- Using sensitivity analysis, I discuss the properties of coalescing techniques for EEE gigabit links, and propose the design of two families of dynamic coalescing schemes that effectively trade off energy saving and delay guarantees.

- I add to the existing literature a unique evaluation of real traces with bidirectional flows over $1\ Gbps$ links. Moreover, I am the first to evaluate the importance of precise timestamping on the trace-based performance evaluation of EEE.

- I suggest a methodology to empirically characterize the energy consumption of a server.

- I provide novel, experimental-based, insights on the power consumption of the components that contribute the most to the server's power consumption, and finally

- I propose an accurate technique to estimate the energy consumption of cloud applications.

Below I give more specific details on the above points.

For what concerns the study of network transmission efficiency leveraging EEE I am the first that model so deeply the behavior of $1\ Gbps$ links considering both the bidirectional behavior of the links and packet coalescing. I study the special case of $1\ Gbps$ links because they are the most commonly deployed links in data centers and their special behavior is challenging for modeling and for the design of new coalescing methods which boost the performance of $1\ Gbps$ links.

I propose an analytical model that uses simple statistical parameters (such as arrival rates and loads) to estimate the energy consumption of gigabit EEE links, and the packet delay incurred in the link when coalescing techniques are adopted. Note that my model can approximate accurately the energy consumption and the average introduced packet delay. However, since my model introduces an approximation on the effect of burstiness in the computation of busy periods after exiting state LPI, some minor discrepancies appear when traffic bursts dominate my Poisson assumption.

Moreover I show how I collected real traces from a large web hosting center and extracted from them the traffic parameters needed by my EEE model. I further evaluate why precise timestamp of the data is important when capturing real data traces. I use those real traces to validate my model in terms of EEE power saving and packet delay by simulating the EEE and packet coalescing behaviors with real input traffic, using a modified ns-3 simulator [1]. In line with other studies focusing on EEE, e.g., [43, 64], I show that, without coalescing, EEE enables non-negligible power saving only when the offered traffic is rather low (few percents of the link capacity) and packet arrivals are bursty.

Another fundamental contribution of Part II consists in the performance evaluation of packet coalescing strategies for EEE. Not only I evaluate the power saving enhancements achievable by means of static coalescing approaches, but I also propose two dynamic strategies to adapt the coalescing parameters to the traffic characteristics. In the first dynamic strategy my performance analysis shows that both the size of the coalescing queue and the duration of the coalescing timeout should adapt to the offered traffic. However, I show that a dynamic coalescing approach can be used, which seamlessly adapts to any traffic conditions, and achieves nearly optimal results in terms of power saving and packet delay. Nonetheless, my thorough investigation shows that also static coalescing can achieve nearly optimal results, thus questioning the importance of exploring more complex approaches based on run time adaptation of the coalescing parameters. However, in the second dynamic strategy, I derive a sensitivity analysis of the coalescing delay and energy saving with respect to the coalescing timer duration and the coalescing buffer size, and use it to design measurement-based control schemes that outperform static coalescing schemes. My new analytical study reveals the importance of coalescing parameters in different scenarios, and unveils that by adjusting the sole coalescing timer duration, it is possible to tune the link performance to achieve near-optimal energy saving, while incurring controlled coalescing delay. In particular, the incurred delay is a monotonically increasing function of the coalescing timer, which is key to design delay-controlled coalescing algorithms. Exploiting my analytical findings, *I design a simple measurement-based delay-controlled distributed adaptive coalescing scheme* in which network cards at the edge of the link coordinate by running a simple distributed algorithm to sense the delay incurred by packets. My proposal uses the sensed delay as control signal to trigger the dynamic adaptation of the coalescing timer in the direction identified through the analysis.

Finally, I show that significant economy can be achieved in a typical data center (∼\$1.7M annually) by replacing legacy Ethernet links with EEE links with coalescing.

For what concerns server's efficiency, I observe that *active CPU cycles per second* (ACPS) is a convenient metric of CPU (central processing unit) load in multi-core/multi-frequency architectures. I show how to isolate the contribution of energy consumption due to CPU, disk I/O operations, and network activity by just measuring server's total energy consumption and a few activity indicators reported by the operating system. I also show that the *baseline* energy consumption of a server — i.e., the energy consumed just because the server is turned on — has a strong impact on server's total consumption. This result enlightens the need of new efficient solutions for servers. As concerns the components' energy characterization, I show that, besides the *baseline* consumption, the CPU has the largest impact among all components, and its energy consumption is not linear with the load. Disk I/O operations are the second highest cause of consumption, and their efficiency is strongly affected by the I/O block size used by the application. Eventually, network activity plays a minor yet not negligible role in the energy consumption, and the network impact scales almost linearly with the network transmission rate. All other components (e.g., memory, fans, GPU, etc.) can be accounted for the *baseline* energy consumption, which is subject to minor variations under different operational conditions. Specifically, the main results of my measurement campaign are listed below:

- The CPU power utilization depends on the number of working cores, the CPU frequency, and the CPU load (in ACPS units). My measurements confirm that the energy consumption with a single working core at constant frequency can be closely approximated by a linear function of the CPU load. However, given a CPU frequency, the energy consumption in multicore architectures is a concave function of the CPU load and can be approximated by a low-order polynomial. The energy consumption for a fixed CPU load is, in general, minimized by using the highest number of cores and the lowest frequency at which the load can be served. However, the minimum achievable energy consumption is a piecewise concave function of the CPU load.

- The energy consumed by hard disks for reading and writing depends on the CPU frequency and the I/O block sizes. Both reading and writing energy costs increase slightly with the CPU frequency. While the energy consumption due to reading is not affected by block size, the energy consumption due to writing increases with the block size. The reading efficiency (expressed in MB/J) is barely affected by the CPU frequency, while writing efficiency is a concave function of the block size since it boosts the throughput of writing until a saturation value is reached.

- The energy consumption and the efficiency of the NIC (Network Interface Card), both in transmission and reception, depends on the CPU frequency, the packet size, and the transmission rate. The efficiency of data transmission increases almost linearly with the transmission rate, with steeper slopes corresponding to lower CPU frequencies. Although a linear relation between transmission rate and efficiency holds for data reception as well, small packet sizes yield higher efficiency in reception.

Overall, supported by my measurements, I provide a holistic energy consumption model that only requires a few calibration parameters for every different server architecture which I want to evaluate (a universal energy model will be too simplistic and inaccurate). I validate my model by means of a server computing the *PageRank* metric of a graph and a *WordCount* application in a

Hadoop platform, first without network activity, next with bulky network activity, and finally in the cloud. I will find that the error of my energy estimates is below 4.1% on average and never worse than a 10%.

Finally, I challenge the common evaluation practices as they frequently adopt over-simplified and unrealistic models for the estimation of the VM resource requirements and physical host resource availability. Hereafter, I identify a set of important system parameters, commonly ignored in favor of simplicity. In addition, common evaluation methods not only ignore these properties, but also rely on very small scale experiment. The properties are outlined below:

- the dynamic **energy consumption** profiles of servers, which is highly correlated with the utilization levels of individual hardware components;
- the complexity in **resource sharing** between VMs in a single host (e.g. CPU, disk, network, memory), as well as the virtualization overheads;
- the performance characteristics of the underlying **network** infrastructure (topology, speed, configuration) and the employed network protocols;
- the **cost** of live **VM migration** in terms of energy, network traffic and application-level performance;
- complex performance behaviors of networked systems observed in **large scale** deployments.

I argue that underestimating the impact of the aforementioned system properties in the evaluation of VM consolidation algorithms introduces significant inaccuracies. The individual relocation decisions are based on inaccurate performance predictions for co-hosted VMs, as well as they overlook the overhead of large-scale VM migrations. As a result, the fundamental trade-off between energy consumption and application performance is not sufficiently captured, and hence, the estimated power-bill savings of the proposed consolidation strategies have limited practical use.

In an effort to address the aforementioned issues, I point out how existing solutions can be reused, combined and extended in order to create an evaluation framework that allows the reliable exploration of the energy-performance trade-offs in VM consolidation strategies. Such a solution is particularly useful, since only few researchers can access a real-sized data center infrastructure for experimentation.

1.2. Publications

The research described in this dissertation resulted in four conference papers [8, 22, 25, 26], three journal articles [9, 24, 63], and one poster [23]. In this section, I elaborate on the goal of each paper and the author's contribution.

The following papers are related with the optimisation of the energy efficiency on ethernet and I worked on this topic only with my advisor. They include (i) analytical models for the estimation of the packet delay and the energy using EEE and its enhancement EEE with coalescing instead of legacy ethernet, (ii) measurement results which validate my statements and (iii) new algorithms for EEE which boost the energy efficiency with regard to delay constraints:

- **Angelos Chatzipapas**, Vincenzo Mancuso, *Measurement-Based Coalescing Control for 802.3az*, in Proceeding of the IFIP Networking '16, Vienna, Austria, May 2016.
- **Angelos Chatzipapas**, Vincenzo Mancuso, *An M/G/1 Model for Gigabit Energy Efficient Ethernet Links With Coalescing and Real-Trace-Based Evaluation*, IEEE/ACM Transactions on Networking, September 2015.
- **Angelos Chatzipapas**, Vincenzo Mancuso, *Improving the Energy Benefit for 802.3az using Dynamic Coalescing Techniques*, in Proceeding of the IEEE ICDCS '15, Columbus, OH, USA, June 2015.
- **Angelos Chatzipapas**, Vincenzo Mancuso, *Modeling and real-trace-based Evaluation of Static and Dynamic Coalescing for Energy Efficient Ethernet*, in Proceeding of the ACM International Conference on Future Energy Systems (e-Energy '13), Berkeley, CA, USA, May 2013.
- Vincenzo Mancuso, **Angelos Chatzipapas**, *On IEEE 802.03az Energy Efficiency in Web Hosting Centers*, IEEE Communication Letters, November 2012

The following papers are related with the energy efficiency and optimization in data and web hosting centers. They include (i) analytical models for the estimation of the energy consumption of a data center server taking into account the CPU, the disk and the network based on an extensive measurement campaign and (ii) proposed enhancement which can reduce the energy consumption of a data center server:

- Jordi Arjona, **Angelos Chatzipapas**, Antonio Fernandez Anta, Vincenzo Mancuso, *A Measurement-based Characterization of the Energy Consumption in Data Center Servers*, IEEE Journal on Selected Areas in Communications, September 2015.
- Jordi Arjona, **Angelos Chatzipapas**, Antonio Fernandez Anta, Vincenzo Mancuso, *A Measurement-based Analysis of the Energy Consumption of Data Center Servers*, in Proceeding of the ACM International Conference on Future Energy Systems (e-Energy '14), Cambridge, UK, June 2014.
- **Angelos Chatzipapas**, Dimosthenis Pediaditakis, Charalampos Rotsos, Vincenzo Mancuso, Jon Crowcroft, Andrew W. Moore, *Challenge: Resolving Data Center Power Bill Disputes: The Energy-Performance Trade-offs of Consolidation*, in Proceeding of the ACM International Conference on Future Energy Systems (e-Energy '15), Bangalore, India, July 2015.

1.3. Organization of the thesis

The thesis is divided in three parts. Part I contains all the background material which is required to follow the flow of the thesis. Part II is about Energy Efficient Ethernet and Part III is about the energy efficiency in data centers. In particular, Part I consists from the introduction and Chapters 2- 5. Chapter 2 contains all the background information regarding Energy Efficient Ethernet and its enhancement, namely coalescing. Chapter 3 discusses about the problem of resource provisioning in a data center. Chapter 4 describes the methodology I will use for my experiments both with Energy Efficient Ethernet and data center servers. Chapter 5 provides information about related work. Part II consists of Chapters 6- 8. Chapter 6 presents an analytical model which accurately estimates the energy consumption and the delay experienced by Energy Efficient Ethernet links with static coalescing. Moreover a sensitivity analysis studies the impact of the coalescing parameters to both the delay and the the energy consumption. Chapter 7 describes two new families of dynamic coalescing algorithms. Chapter 8 is the evaluation of the model and the comparison of static versus dynamic coalescing schemes. Part III consists of Chapters 9 and 10. Chapter 9 presents my measurement campaign, for every single component of the data center servers which I tested. In Chapter 10 I model the energy consumption of the servers based on a few calibration parameters which I find during my measurement campaign and I discuss my findings and their implications. Finally, Chapter 11 concludes and summarizes the thesis.

Chapter 2

Background on Energy Efficient Ethernet with gigabit Ethernet and coalescing

Energy Efficient Ethernet 802.03az [47] was standardized in September 2010. It aims to provide significant power saving in LANs. Formerly, the evolution of LANs led towards higher link speeds for faster communication and higher bandwidth, in order to satisfy the increased demand for data (link speeds from 10 $Mbps$ to 10 $Gbps$) without power consumption concerns. In fact, the electricity consumption of relatively "old" network interfaces remained in very low levels so the main concern of Ethernet component producers was not to save power. For example, in 100 $Mbps$ Ethernet links, the Ethernet devices consume about 200 mW of power [81]. However, higher speed Ethernet links (1 $Gbps$ or faster) require several Watts of power consumption [85] for each network interface. Considering a usual server that consumes around 200 W, a simple Ethernet device contributes to $\sim$ 10% of this amount. Indeed, data centers and web hosting centers have a huge number of network interface cards which eventually generate a high cost (in terms of electricity bills). Thus, the idea of reducing the power consumption of Ethernet devices appears in the foreground.

Legacy Ethernet consumes a constant amount of power either with or without traffic, which makes it totally inefficient with typical Ethernet traffic profiles. This behavior results in a huge waste of power since it is well known that Ethernet links are inactive most of the time with utilization factors from 5% for a home PC to 30% for heavy loaded data servers [63, 68, 72]. EEE aims to reduce this waste of power and approach *power proportionality*, i.e., a power consumption proportional to the served traffic. The EEE standard introduces four new states for the Ethernet link, namely state "Active" (A) which corresponds to the busy period, state "Low Power Idle" (LPI) in which there is no traffic and the link consumes substantially less power than in state A ($\sim$ 90% less power according to [79]), and states "Sleep" (S) and "WakeUp" (W) which correspond to the time spent during switching from state A to LPI and vice versa, respectively [64]. EEE spec-

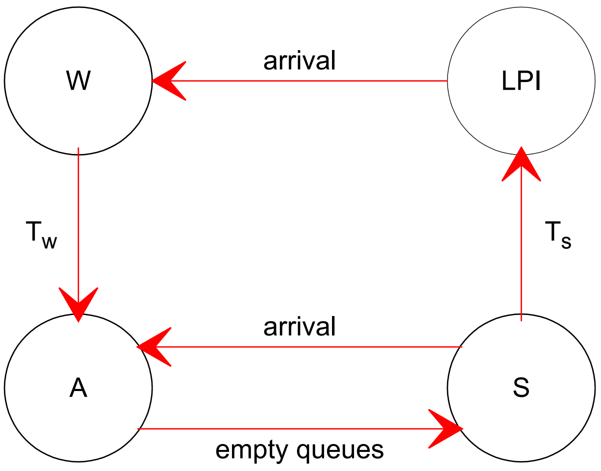


Figure 2.1: State transition diagram for EEE 1000Base-T links.

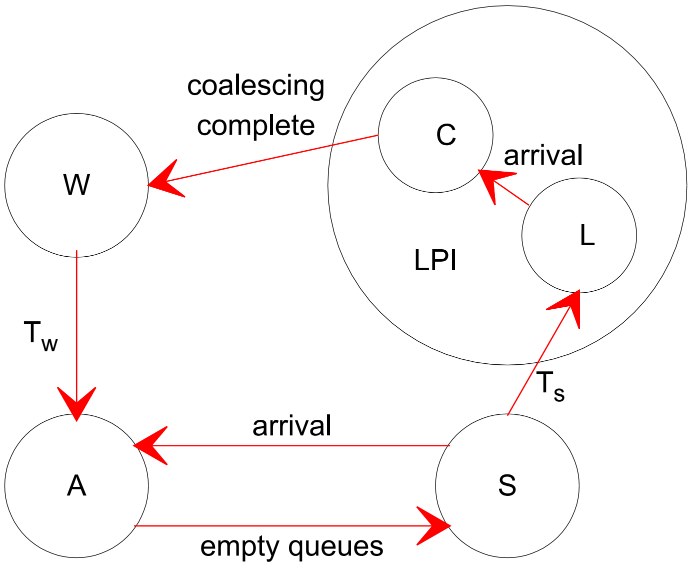


Figure 2.2: Modified state transition diagram for EEE 1000Base-T links with coalescing.

ifications and state transition schemes are different for 100Base-T, 1000Base-T and 10GBase-T links. In particular, since I focus on commonly deployed 1 $Gbps$ links, in the following subsection I describe in detail how EEE 1000Base-T links behave, since they represent the only case in which the EEE standard accounts for the bidirectional behavior of traffic.

2.1. Gigabit EEE Link Operation

Behavior of 1Gbps EEE links. EEE 1000Base-T links can be in one of the following four states: Active (A), Sleep (S), WakeUp (W) and Low Power Idle (LPI). The state transition diagram is illustrated in Figure 2.1. Frame transmissions in either of the traffic directions only occur in state A. When the two network cards connected to the link complete transmitting all the buffered frames, the link enters state S as a transition to state LPI. If no frame arrives for T_s seconds while in state S, the link enters state LPI, during which power consumption is minimized. A frame arrival in state LPI results in the link transitioning to state W which lasts T_w seconds. After this wake interval, the link transitions to state A and any of the network interfaces connected to the link can transmit. Standard values for T_s and T_w are 182 μs and 16 μs, respectively. Thus, the fact that a frame arrival in the sleep interval (state S) causes an immediate transition to state A, avoids incurring in delays of up to 182 μs, which can be quite large if compared to the transmission time of a single packet (e.g., 12 μs for a 1500-byte packet).

Behavior of 1Gbps EEE links with coalescing: Packet coalescing techniques have been proposed to boost EEE performance when traffic is not scarce and packet arrivals have short spacing. The basic idea behind coalescing is to aggregate packets in a buffer of limited size until either the buffer is full or a timeout expires. When coalescing techniques are used, the transition from state LPI to state W is delayed. Therefore, I fictitiously split state LPI into two states, as shown in Figure 2.2: state L, which represents state LPI when there is no packet queued in the coalescing buffers—and which is equivalent to LPI in systems with no coalescing—and state C, in which coalescing buffers are not empty but neither the coalescing timer expired nor the coalescing buffers were completely full. Indeed, the system transitions from state C to state W as soon as one of the coalescing buffers gets full or the coalescing timeout expires. Remark that the newly introduced state C is fictitious, since the transition $L \to C$ does not represents a change of state for the EEE Ethernet hardware. State C simply represents the extension of an LPI interval due to coalescing operations, counting from the arrival of the first packet in one of the two coalescing queues. The time spent in state C, namely τ_c, is a random variable which depends on the size N_c of the coalescing buffers, and it is limited by the coalescing timeout T_c.

2.1.1. Efficiency Problems of EEE

As shown in previous evaluation works, such as [81, 82], the problem of EEE links (and especially in 1000Base-T links) is the transitioning time. When the traffic is scarse and packets are spaced rather that bursty, the EEE mechanism rarely allows the link to complete the transition to LPI. Thereby, the link spends more time transitioning than transmiting. Specifically, as I said earlier, we can observe that for transmiting one big packet, 1 $Gbps$ links require about 12 μs while for transitioning they spend at least $T_s + T_w = 182 + 16 = 198$ μs. For smaller packets the correspondence is even worst. In Table 2.1 I show the time which is required for waking up

Table 2.1: Time required for Wake Up, Sleep and Frame Transmission [μs]

Speed	Min. T_s	Min. T_w	Transmission Time for 1500 $bytes$ (efficiency)	Transmission Time for 150 $bytes$ (efficiency)
100Base-T	200	30	120 (48%)	12 (4.8%)
1000Base-T	182	16	12 (5.7%)	1.2 (0.57%)
10GBase-T	2.88	4.48	1.2 (14.6%)	0.12 (1.46%)

the link, putting the link to Sleep state and transmitting a small and a big packet using different link speeds, including in parenthesis the percentage of time that the link requires for transmitting only a single packet over the cycle duration. The ratio for transmitting single small packets at any link speed are excessive and for this reason, packet coalescing is proposed to avoid frequent state transitions and prolong the duration of state LPI. Packet coalescing tends to approach *energy proportionality* at the cost of additional delay to the packets. However, I will show in Chapter 8 that the delay is negligible if compared with the energy benefits.

Chapter 3

Background on resource provisioning in data centers

Performing resource provisioning in a single data center infrastructure is a compound problem with multiple competing objectives (see Figure 3.1 for a brief taxonomy).

Firstly, *consolidation* aims to compress workloads into as few physical hosts as possible, and either turn off or leave idle the unused part of the infrastructure. During this step, the objective is to maximize the energy saving, at the cost of performance. Some approaches use live VM migration to implement consolidation [14, 70], while some others steer new workloads to different servers [95]. Secondly, the opposite to the process of consolidation, is the *elimination of performance hot-spots*, which spreads VMs across the data center, increasing the active physical hosts. Some techniques aim to remove network-related hot-spots [31], while others utilize end-host information to avoid high server utilization [89]. Lastly, a *load-balancing* process can run in the background and relocate VMs aiming to smoothen the load variations across the infrastructure, and therefore, better absorb the performance spikes of bursty workloads. Load-balancing solutions may be based on software proxies [94], proprietary hardware designs [74] or they can be built as software defined network applications [77].

Usually hot-spot removal and consolidation are used together, hand in hand. The two functionalities have opposing goals, but are equally necessary to achieve an equilibrium between performance and energy saving. Specifically, this is the most important aspect in designing greener data center solutions: *be in position to make informed decisions about the application-level performance which is sacrificed in trade for lower energy consumption, and vice versa*.

Energy-efficient VM placement algorithms: The energy/performance trade-off is controlled by the VM placement algorithms, which implement the decision-making logic for the followings:

- **Choose a source host** with average utilization above, in case of hotspot removal, or below, in a case of consolidation, a threshold.
- **Choose a VM** from the selected host based on its resource requirements. For example, during

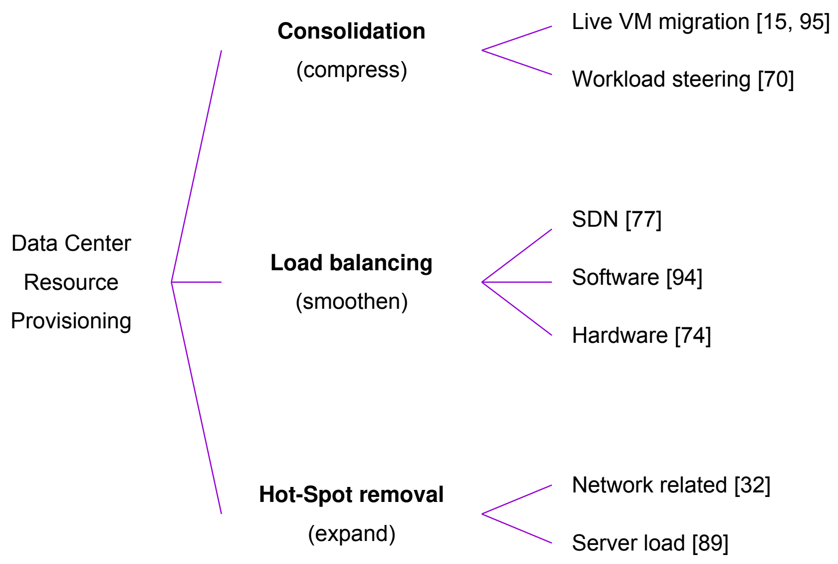


Figure 3.1: A brief taxonomy of resource provisioning functions for data center environments.

the evacuation of an under-utilized server, VMs are ordered based on their resource requirements.

- **Choose a destination host** with sufficient available resources (e.g. disk, network, CPU, memory) to fulfill the resource requirements of the previously selected VM, determined by its SLA.

Numerous research efforts transform this decision making process into a vectorized bin packing problem [14, 70]. VMs are represented as n-dimensional vectors of estimated resource demands, while each host is represented as an n-dimensional vector of available resources. VM placement aims to fulfill the minimum guaranteed resources, specified by the service SLAs, while minimizing the number of active hosts. Since the vector bin packing problem is NP-hard, a number of near-optimal solutions have been proposed using a variety of heuristics [84, 88] (e.g. first-fit decreasing, best-fit decreasing, worst-fit decreasing, etc.). Alternative approaches towards the placement problem use genetic algorithms [50] and dynamic programming [41].

3.1. Common pitfalls

The VM migration decisions use as inputs: (i) the resource requirements of a VM (given an SLA), (ii) the expected load increase in the destination host, (iii) the available resources of the physical hosts, and (iv) the expected level of performance for VM applications. The main argument of this section states that the majority of the existing works does not accurately capture the aforementioned decision criteria.

Next, I elaborate on the aforementioned evaluation and design pitfalls, related to the specific properties of large-scale data centers which tend to ignore: (i) the dynamic of the underlying resource sharing model and the migration cost (§ 3.1.1) and (ii) the energy consumption profiles under mixed workloads (§ 3.1.2).

3.1.1. Modeling the availability of resources

Cloud providers have been refraining from using consolidation algorithms on their infrastructures mainly because it is not easy to predict the performance penalties on hosted applications. While the overhead of virtualization has been significantly reduced (e.g. *paravirtualized* I/O, hardware support), the interaction model with a host's physical resources has become more complex. For example, Wang *et al.* [37], exemplify some interesting artifacts in the perceived CPU and network resource availability by guest OS. Such performance variability has been measured to significantly affect large-scale time-sensitive services [49]. This performance variability is a direct consequence of the resource sharing functionality implementation between co-hosted VMs. Nevertheless, most of the heuristics used in VM placement algorithms, assume that the virtualization platforms provide perfect performance isolation. Hence, they suggest that VM resource utilization, and consequently application-level performance, remains the same across different hosts.

The above assumption, however, can lead to incorrect VM placement decisions. The amount of the resources which each VM receives depends on three factors: the scheduling policy of the hypervisor, the available resources of the hosting platform, and the activity of co-hosted VMs. None of these three factors can be considered static, and moreover, they exhibit a high degree of interdependencies. For example, consider many highly-utilized VMs collocated on a server, each receiving a fair share of the CPU time. On a lower utilized server, the same VM would almost certainly reach a higher peak. Therefore, a typical hot spot removal algorithm would underestimate the peak CPU requirements of a VM, and could potentially make sub-optimal decisions. From the above it is clear that estimating the application level performance is a fairly difficult task.

Another over-simplifying assumption which is commonly made, is the inference of application SLA violations, based on VM or host-level utilization metrics. First, the poor resource sharing models which are used during evaluation, do not provide accurate utilization estimations. Second, it is fairly unreliable to employ only the CPU utilization to infer SLA violations, since this approach is susceptible to false negatives, especially for bursty workloads. This problem has been pointed out by *Wood et al.* in [89], via extensive experiments.

Finally, the available network resources is another important factor which also determines the application-level performance. This includes the available bandwidth at the end-hosts (including the CPU overheads of packet processing), the employed protocols, the topology of the data center's networking infrastructure, the speed of physical links, and the scheduling algorithms at intermediate devices.

3.1.2. Accuracy of energy consumption models

Cloud computing is based on resource sharing of the available servers, i.e., CPU, disk, memory, network. In previous works (see the introduction of this chapter)the authors either theoretically describe VM placement algorithms or they identify the cost of a placement considering a dedicated use of the available resources. Consequently, the challenge is what happens when simulataneous VM placements happen and new jobs arrive and current jobs have to continue their work. How can we estimate the energy needs and the performance degradation (or improvement) using one technique over the other?

Many of the VM placement approaches, covered in the introduction of this chapter,provide only gross insights on the resulting energy savings. The achieved accuracy in the estimated savings is usually limited at the level of accounting the number of powered-on servers over the unit of time. This reduced level of detail does not allow users to effectively evaluate the energy-performance trade-off.

Some research efforts, like [6, 14], consider the use of a more detailed energy model. Effectively, they are based on the observation that CPU utilization is highly correlated with the overall energy consumption of a server. As a result, they use linear models which are based on current utilization levels to estimate the energy consumption.

The importance of the above facts has been pointed out by several studies (e.g. [5,8]), showing that depending on the characteristics of a workload, the level of CPU-load alone might not be a very accurate metric. This is especially true for storage and network devices which implement energy saving features and have a wide and dynamic energy range. The system-level utilization is not modeled accurately in the simulation frameworks which are commonly used to evaluate VM consolidation algorithms. Hence, the input which is used in their linear energy/CPU-utilization models, is not reliable.

Chapter 4

Methodology

In this chapter I explain the process I follow in order to perform the measurements. First, I describe how I capture real data traces from a big data center and the topology and equipment we require. Second, I expose the measurement techniques I used to characterize the energy consumption of the main components of a data center server, i.e. the CPU, the disk and the network.

4.1. Network traces collection for the study of EEE with coalescing

Using the model like the one presented in [64] or the ones proposed in this thesis in Chapter 6, I can estimate the EEE power saving by means of the average and standard deviation of the packet interarrival time, the average packet size, and the offered load. The models use queuing theory, and treat EEE state transitions as a renewal process. The analysis yields the average time that the link spends in the four different EEE states: Active, Low Power Idle (LPI), Sleep, and Wake Up. The time that the link remains in each state, times the power consumed in the corresponding state, gives the total power consumption of the EEE link.

I can also simulate EEE operations by using real traces for packet arrivals. With a *C++* simulator, I compute the exact EEE power saving, and compare this value to the estimate yielded by the model. The simulator uses trace files containing, for each packet, the arrival timestamp and the packet size. The input for my model and for simulations has been obtained by collecting real traces in InterHost, an operational data center hosting web servers and located in Madrid, Spain.

In particular, to run simulations I modified the ns-3 simulator to implement EEE and packet coalescing.[1] First, I designed and coded a novel Ethernet channel object in ns-3. Such an Ethernet channel can simulate the bidirectional behavior of Ethernet links. Second, I added on the network devices the EEE functionality, i.e., I defined the EEE states. Third, I implemented packet coalescing and coordinated packet transmission so that a simulated EEE link enters state L only when both link directions are inactive for T_S seconds, and exits state C only when coalescing operations are complete either because the coalescing timeout expires or one of the coalescing

[1] Simulation code available at `https://github.com/ferrarif50gr/ns3-dynamic/`

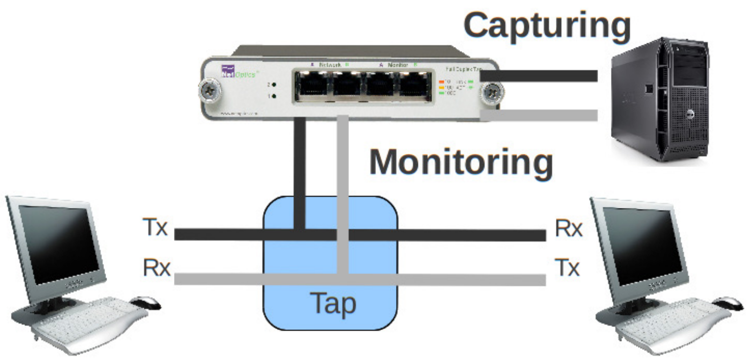


Figure 4.1: Measurement architecture with passive tap.

buffers fills up.

To take measurements, I deployed a monitoring server to capture and store the traffic flowing through a link, and a *tap* to sniff and duplicate real packets without affecting the traffic, as shown in Figure 4.1. I use a NetOptics passive device which is inserted in a 1000Base-TX Ethernet link and duplicates each and every signal over the link [2]. The NetOptics device, as shown in the figure, is also able to replicate uplink and downlink traffic over two separate cables connected to the monitoring server. The two monitoring ports of the tap are connected to a digital capture card. Specifically, a high accuracy two-port Endace DAG card [3] is mounted on the monitoring server. The DAG card is a capture device with dedicated CPU and memory, able to capture 100% of the traffic at up to 10 $Gbps$ over each port. Furthermore, the DAG card has a unique timestamping engine that guarantees clock synchronization to the nanosecond over the two monitoring ports. The DAG is activated once per hour to collect at most 100 bytes per packet for 200 seconds. I keep a remote ssh connection with the server, so that I can periodically transfer the collected traces to a Linux server in Institute IMDEA Networks. Once the traces are in our lab, I post-process the traces with the *tshark* [2] packet analyzer and create simplified and anonymous trace files containing only arrival timestamps and packet sizes.

4.2. InterHost Measurements

This section presents traffic measurements that have been taken at InterHost. InterHost allowed me to install traffic measurement tools in front of one of their firewalls, which protects part of InterHost's customer web servers. The goal of those measurements is to collect enough data from a 1000Base-TX link to characterize the traffic behavior of a real commercial installation, and to estimate the power saving that might be achieved at the hosting center by replacing existing Ethernet links with IEEE 802.3az EEE links. Savings are estimated by means of a simulator, using real traces as input, and by means of a model previously proposed for EEE links [64]. Note that the IEEE 802.3az standard specifies that 1 $Gbps$ EEE links can go to low power state only

[2]`https://www.wireshark.org/docs/man-pages/tshark.html`

when no traffic is present in both link directions. Therefore, the correlation of traffic in the two link directions is important for estimating the achievable power saving not captured by the model in [64]. Note that, unless coalescing techniques are used, 1 $Gbps$ EEE links save power by introducing negligible delay (few μs) to wake up interfaces in power saving mode [47], therefore, μs accuracy is important in traffic measurements.

Since obtaining real data is usually very difficult, synthetic traffic generators are often utilized, though they lead to less accurate results. In this case, InterHost allowed me to monitor the traffic at the interface between Internet and one of their firewalls. This gives me the advantage of using real data and real traffic that yields more realistic results.

However, in respect of user's and company's privacy and security, I only capture few bytes of each packet, and I do not inspect the payload. Actually, my goal is to capture the arrival time of the packet and its size, which is enough for my purposes.

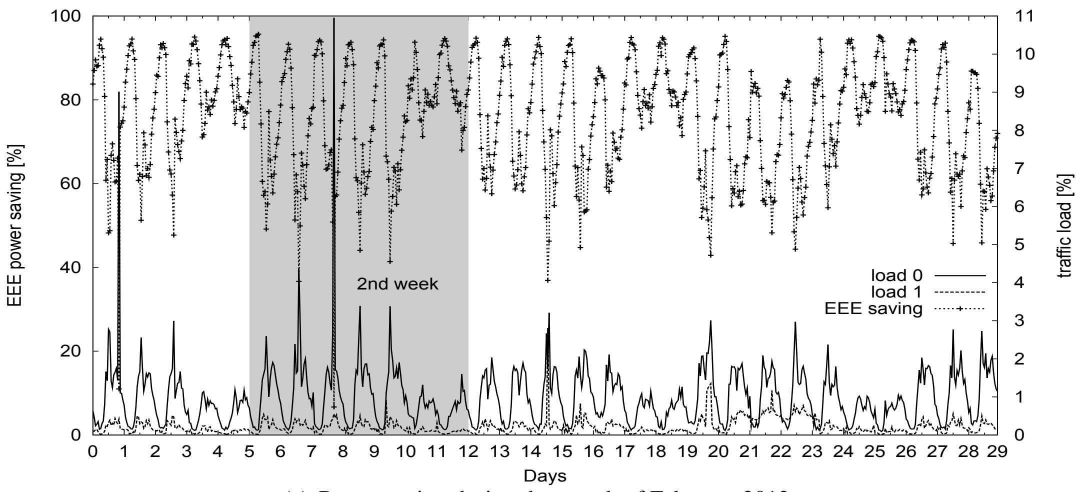
To achieve my goal, I need to collect precise and clock-synchronized timestamps for packet arrivals over the two link directions. Therefore I need precise measurement tools and a measurable source of real traffic.

Similarly to [64], I use real traces to evaluate the potential EEE power saving, but I focus exclusively on 1 $Gbps$ links and on the impact of uplink/downlink traffic correlation on such saving. With my measurements, I show that EEE might save more than 40% of the link power most of the time, with peaks of 90% or more during night hours. I also unveil that high precision timestamps are key to achieve high accuracy estimations via simulation, and to enable the use of simplified analytical computations. In particular, noisy measurements severely impact the quality of EEE power saving estimates as soon as the maximum timestamp deviation due to noise reaches a few milliseconds, which is below the typical timestamp accuracy of non-dedicated network hardware, i.e., of inexpensive but imprecise driver timestamping. This justifies using specialized high accuracy timestamping hardware.

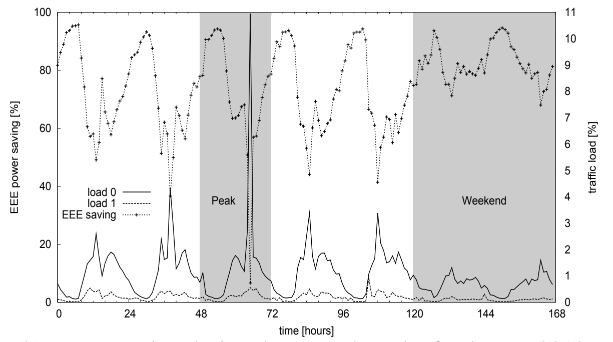
4.2.1. Trace-based Simulation and Analysis

I run EEE simulations based on the traces captured by the monitoring server. For the simulation, I use the same *C++* simulator used in [64]. I also extract the statistical parameters needed to run the EEE model in [64], and estimate the EEE power saving through the model as well. In particular, the model uses the average frame size, the average load, and the average and standard deviation of frame interarrival times, to compute the expected energy consumption of EEE links. In contrast, without EEE, the power consumption is constant.

I have captured traffic traces from November 2011 until March 2014, but I only show results for February 2012, which are representative of the rest of my measurements. For that month I observed a weekly periodic behavior, and a daily typical maximum traffic of about 4% in the most loaded link direction, with the exceptional values of 9% and 11% on one traffic direction on February 1st and 8th, respectively. Over the whole duration of my measurement campaign the maximum values might change however the periodic behavior remains the same.



(a) Power saving during the month of February 2012.

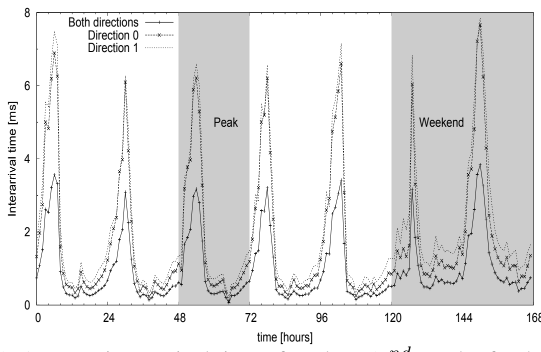


(b) Power saving during the second week of February 2012.

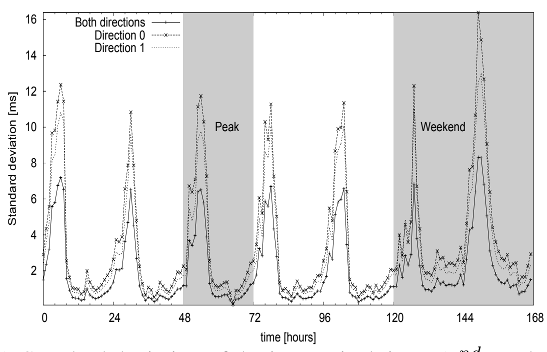
Figure 4.2: Monthly and weekly plots for traffic and power saving (February 2012).

In Figure 4.2(a) I plot the monthly load in each link direction (rightmost y-axis), labeled as "load 0" for one link direction and "load 1" for the other link direction. The figure also shows the power saving that might be achieved by means of EEE links, computed through simulation (leftmost y-axis). My first observation is that there is a maximum traffic load of about 11% in the most loaded direction, whereas during weekends and overnight the peak load is below 2%. In the figure, traffic patterns are quite regular, showing higher traffic activity over weekdays, followed by lower traffic intensity over the weekend. It is also evident that overnight traffic is very low.

Figure 4.2(b) illustrates results for the second week of February. I choose this week because it shows a traffic spike on February 8th. Daily spikes occurred at about 1 and 6 PM. This traffic distribution over time clearly depends on the nature of the websites hosted at InterHost premises, about which I have no information. However, the measured traffic patterns are qualitatively in line with other patterns reported in literature (e.g., see [62] and references therein). Processing the collected data with the EEE simulator reveals that overnight and during the weekend, EEE might save 70-90% of the power with respect to legacy gigabit Ethernet. Noticeably, the power saving exceeds 80% in more than 40% of the samples, and during weekdays the power saving is



(a) Average interarrival time of packets (2^{nd} week of February 2012).



(b) Standard deviation of the interarrival times (2^{nd} week of February 2012).

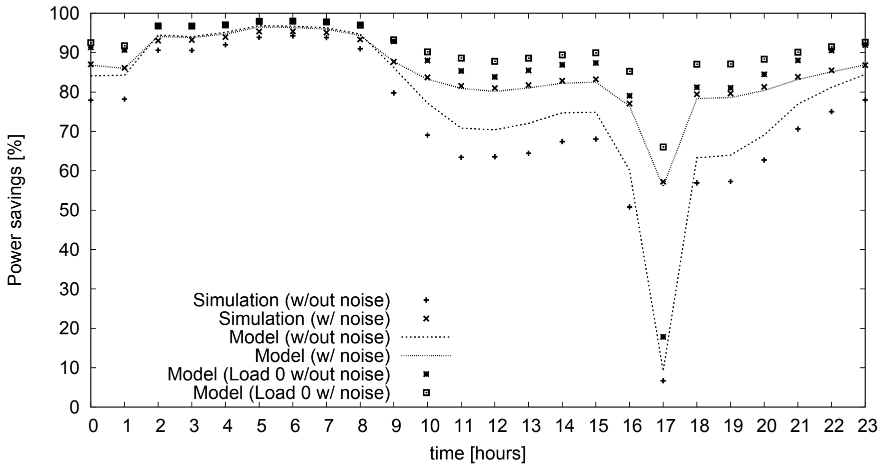
Figure 4.3: Weekly plots for the average and standard deviation of packet interarrival times (February 2012).

larger than 40% in 99% of the samples.

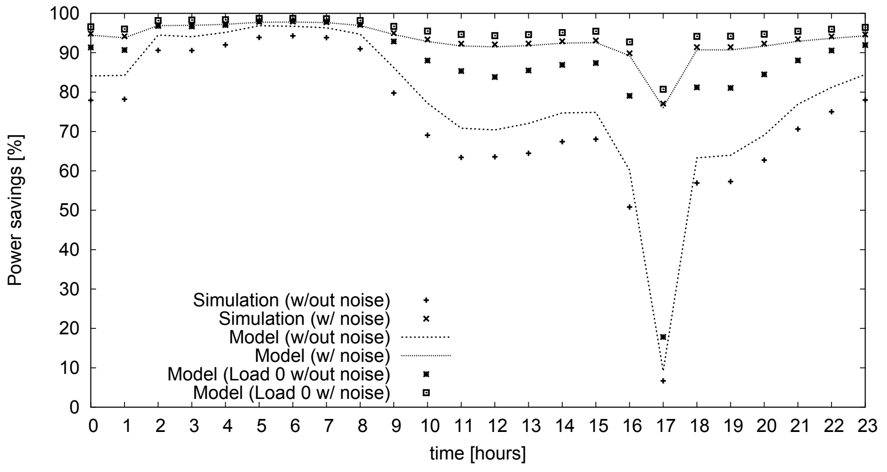
Figures 4.3(a) and 4.3(b) give more information about the traffic arrival characteristics, namely the average interarrival time of the packets and the standard deviation of interarrival times, which are needed in order to run the model in [64] and analytically estimate the EEE power saving. In each of the two subfigures, two of the lines correspond to the traffic in each direction independently, whereas the third line corresponds to the overall link traffic. Both subfigures cover the same time interval covered by Figure 4.2(b). In Figure 4.3(a) I observe that the packet interarrival pattern is similar for each day but the third day of the week, when I observe a traffic spike (see Figure 4.2(b)), which corresponds to lower interarrival times. Similarly, the pattern observed for the standard deviation of the interarrival time in Figure 4.3(b) is quite regular, and approaches zero in correspondence to the traffic spike observed during the third day of the week. Note that, in both Figures 4.3(a) and 4.3(b), I observe higher values in the weekend, which corresponds to lower traffic activity. As expected from the analysis in [64], Figures 4.2(b), 4.3(a), and 4.3(b) confirm that the higher the average and standard deviation of the interarrival time, the higher the power saving. For instance, the valley in the power saving plot highlighted in Figure 4.2(b), corresponds to the minimum of interarrival time (Figure 4.3(a)) and of its standard deviation (Figure 4.3(b)). Similarly, peaks of power saving correspond to peaks of average and standard deviation of the interarrival time.

4.2.2. Impact of noisy measurements

Here I use modified traffic traces to show the impact of noisy measurements on the quality of EEE power saving estimates. In particular, I picked the day with the maximum traffic recorded over the entire measurement campaign, which corresponds to February 8th, 2012, and perturbed the originally collected timestamps by adding zero-mean uniformly distributed noise. For each packet trace, I plot the EEE power saving based on simulation and model. However, since the



(a) Simulation and model with noise in [-5, 5] *ms*.

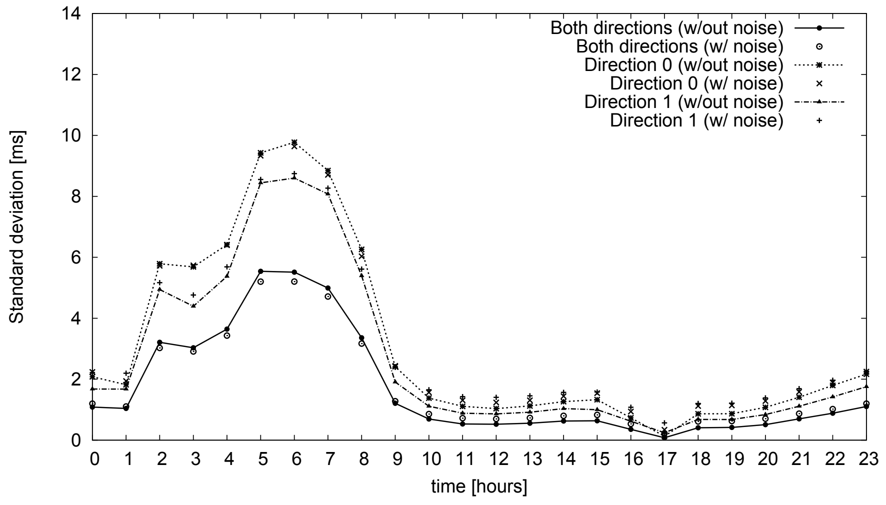


(b) Simulation and model with noise in [-50, 50] *ms*.

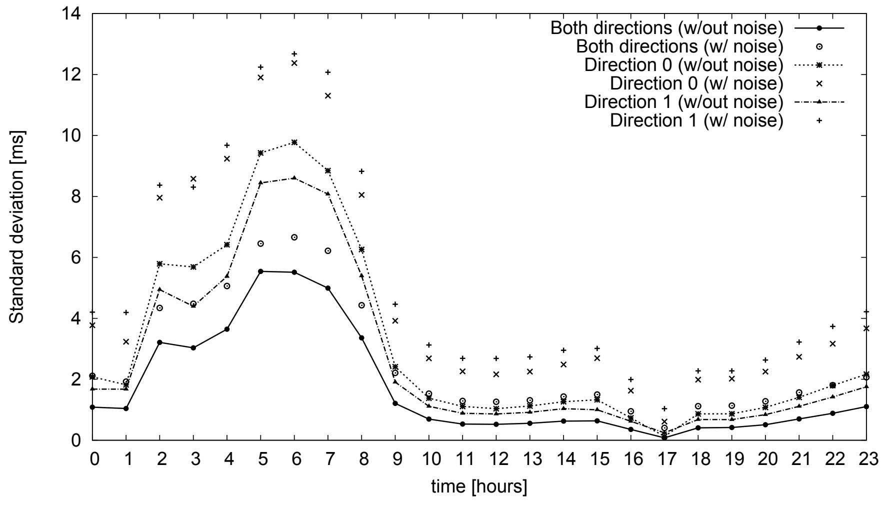
Figure 4.4: Evaluation of EEE power saving with model and simulation, with and without noisy timestamp measurements.

model runs for unidirectional traffic only, I feed the model with either the traffic of each link direction separately, or with a single trace representing both traffic directions. Specifically, when the model is computed over the aggregate traffic measured over the two link directions, I merge the traffic traces obtained for the two directions. Consequently, the model cannot completely capture the correlation of traffic in the two directions. In fact, the IEEE 802.3az standard says that link interfaces can enter the LPI state only when *both link directions* are idle. Instead, I use a simplified model to cope with a unique link direction, resulting from the merging of the two real link directions.

Figures 4.4(a) and 4.4(b) depict the EEE power saving computed via simulation and model. Input traces were used both with the original high precision timestamp and with modified times-



(a) Standard deviation of interarrival times with noise in [-5, 5] *ms*.



(b) Standard deviation of interarrival times with noise in [-50, 50] *ms*.

Figure 4.5: Standard deviation of packet interarrival times with and without noisy timestamp measurements.

tamps. In Figure 4.4(a), original timestamps have been altered by adding uniformly distributed noise in the range $[-5, 5]$ ms. In Figure 4.4(b), noise ranges in $[-50, 50]$ ms.

Each plot contains the power saving estimates obtained with the model computed on the aggregate link traffic (merging of the two traffic directions, labeled as "Model" in the figures), with and without the artificially added noise. The figures also report the EEE power saving as estimated by the model when considering only the traffic in the most loaded link direction ("load 0" in the figures), with and without noise. Finally, the figures include the EEE power saving computed through simulations, with and without noise. Therefore, in each plot, the values obtained without noise are always the same, thus representing the benchmark for the experiment.

From the figures, I note that model and simulator report similar trends, with close but distinct power saving estimates. Differences are well explained considering that the model does not capture the bidirectional nature of the EEE power saving mechanism. Limited differences are due to the low load measured on both link directions. The model which only considers the most loaded link is obviously the one reporting the highest power saving. Similarly, the model considering the aggregate traffic yields the lower power saving, since it does not consider that packets belonging to opposite traffic directions might be served in parallel.

Let's now consider the impact of noise. In Figures 4.4(a) and 4.4(b), we can see that savings estimated with noisy measurements tend to be higher. In fact, adding noise to timestamps contributes to break the traffic correlation between the two directions, so that (i) simulator and model yield very similar results, and (ii) power saving occurs with roughly the product of probabilities of having each link direction idle. As a result, $\sim 80\%$ power saving can be estimated even for the peak hour.

I remark that noisy measurements cause erroneous power saving estimates, and conclude that timestamp errors larger than few *ms* are not tolerable to achieve accurate estimates. Considering that *ms* accuracy in time-stamping is barely achievable with ordinary operating systems and driver-operated time-stamping, which depends on system interrupts, I also conclude that dedicated traffic measurement tools are needed, as the ones that I have used for the measurements.

Note that timestamping noise does not affect load, interarrival average and packet size. It changes only the standard deviation of the interarrival time. Therefore, I show in Figures 4.5(a) and 4.5(b) the standard deviation of interarrival times with and without noise, for the same time interval used in Figures 4.4(a) and 4.4(b). Figure 4.5(a) illustrates the standard deviation with noise uniformly distributed in [-5, 5] ms, and Figure 4.5(b) refers to uniform noise in [-50, 50] ms. The figures report separately the interarrival statistics for each traffic direction, plus the statistics for the overall arrival process (i.e., considering interarrivals between packets accessing the link, independently from their flow direction). I observe that the presence of noise in the timestamps can induce to deal with packets separated by a short interval as if they were sent back-to-back (note that I do not allow timestamp noise to induce packet sequence reordering), this error reducing the standard deviation of interarrival times at particular measurement epochs. Most importantly, I note that small errors in the estimate of the standard deviation of the interarrival time cause large errors in the estimate of the EEE power saving.

Overall, we need a better model than [64] and this is my motivation for the work described in Part II of this thesis.

4.3. Data center servers consumption measurements

In this section I introduce the measurement setup and techniques I used to characterize the energy consumption of CPU activity, disk access (read and write operations), and network activity. I start my measurements by profiling the CPU energy consumption, from where I obtain information about the *baseline* energy consumption of the servers and the energy consumption due to

Table 4.1: Characteristics of the servers under study

Component	Servers		
	`Survivor`	`Nemesis`	`Erdos`
CPU (#cores)	4	4	64
Freqs List ([GHz])	1.2, 1.333, 1.467, 1.6, 1.733, 1.867, 2.0, 2.133	1.596, 1.729, 1.862, 1.995, 2.128, 2.261, 2.394, 2.527, 2.666, 2.793, 2.794	1.4, 1.6, 1.8, 1.8, 2.1,2.3
RAM	4 *GB*	4 *GB*	512 *GB*
Disk	2 *TB*	2+3 *TB*	2×146 *GB*, 4×1 *TB*
Network	1 *Gbps*	3×1 *Gbps*	4×1 *Gbps*, 2×10 *Gbps*

CPU load. Afterwards, I profile the other two components, namely, disk and network. Note that CPU and baseline measurements are of capital importance in order to evaluate the other components, because every time that I run a script to profile the behavior of another component, some CPU cycles are needed in order to execute it as well as to use the component that has to perform the task. Therefore, to understand the contribution of any component, I first need to identify the contribution of the CPU and the baseline and calculate the difference.

To explore the possible parameters which determine the energy consumption of a data center server and to obtain statistical consistency, I run the experiments multiple times. Similarly, I run these experiments in different server architectures in order to validate my results and give consistency to my conclusions.

4.3.1. Devices and Setup

In order to monitor and store the instantaneous power used by a server during the different experiments I used a Voltech PM1000+ power analyzer[3], which is able to measure the total instantaneous power used by the server under test on a per-second basis. In Figure 4.6 I show a schematic representation of the setup I used and the components under testing. More specificaly, in order to take my measurements I connected the server being measured to the power analyzer and the latter to the power supply. In the case of servers with power redundancy one of the two power sources was unplugged to ensure that the power measurement was correct. In the experiments where the network was not involved (CPU and disk), I unplugged the network cable from the server, which has an impact on the power utilization as the port goes idle. In the network based experiments I established an Ethernet connection between the server under study and a second machine in order to study the server behavior, both as a receiver and as a sender.

I evaluated three different servers: `Survivor`, `Nemesis`, and `Erdos`. I will now present these servers although their main characteristics, including their sets of available CPU frequencies, can be also found in Table 4.1. `Survivor` has an Intel Xeon E5606 4-core processor[4], with 4 *GB* of RAM, a 2 *TB* Seagate Barracuda XT hard drive and a 1 *Gigabit* Ethernet card integrated in the motherboard. `Nemesis` is a Dell Precision T3500 with an Intel Xeon W3530 4-core processor, 4 *GB* of RAM, 2 hard drives (a 2 *TB* Seagate Barracuda XT and a 3 *TB* Seagate

[3]More information about the PM1000 can be found in `http://www.farnell.com/datasheets/320316.pdf`

[4]FSB frequency was fixed for all CPU frequencies in the experiments performed with Intel machines.

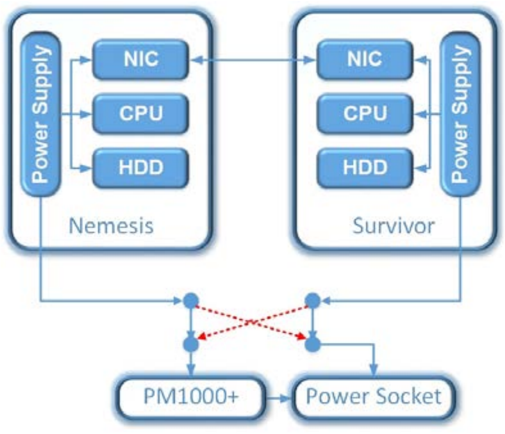


Figure 4.6: Schematic representation of the setup when Nemesis is being measured. Red arrows show the alternative scheme to measure Survivor (or Erdos).

Barracuda), a 1 *Gigabit* Ethernet card integrated in the motherboard, and a separate Ethernet card with two 1 *Gigabit* ports. In this study I only evaluate the Seagate Barracuda XT disk and the integrated Ethernet card. Both `Survivor` and `Nemesis` use the Ubuntu Server edition 10.4 LTS Linux distribution. Finally, `Erdos` is a Dell PowerEdge R815 with 4 AMD Opteron 6276 16-core processors (i.e., 64 cores in total), 512 *GB* of RAM, two 146 *GB* SAS hard drives configured as a single RAID1 system (which is the “disk” analyzed here) and four 1 *TB* Near-line SAS hard drives. It also includes four 1 *Gigabit* and two 10 *Gigabit* ports. `Erdos` is a high-end server and uses Linux Debian 7 Wheezy.

4.3.2. Collecting System Data and Fixing Frequency Parameters

One prerequisite for those experiments is to have Linux machines because we can freely modify and check the Linux kernel, for instance to add kernel modules and utilities[5] which allow to change CPU frequencies at will, or to log CPU activity stats so that I can periodically read the core frequency and the number of *active* and *passive* CPU ticks at each core[6]. Once I have the number of ticks and the core frequency, since a tick represents a hundredth of second, cycles can be calculated as 100 *ticks/frequency*.

I use active cycles per second (ACPS) instead of CPU load percentage to characterize CPU load because ACPS depend on the CPU frequency used, as the higher the frequency the more the work that can be processed. In contrast, CPU load percentages cannot be compared when different frequencies are used, while the amount of ACPS that can be processed can be considered as an absolute magnitude. In order to get (set) information about the operative frequency of the system I

[5]e.g., cpufrequtils, acpi-cpufreq.

[6]File `/proc/stat` reports the number of ticks since the server started, devoted to *user*, *niced* and *system* processes, waiting (*iowait*), processing interrupts (i.e., *irq* and *softirq*), and *idle*. In the experiments I count both waiting and idle ticks as *passive* ticks, while I denote the aggregated value of the rest of ticks as *active*.

used the cpufrequtils package[7]. With those tools, I can monitor the CPU frequency at which the system works and assign different frequencies to the cores. However, to limit the number of possible combinations to characterize, I assign the same frequency to all cores.

4.3.3. CPU

In order to evaluate the CPU power utilization I prepared a script based on a benchmark application, lookbusy.[8] Note that lookbusy allows to load one or more CPU cores with the same load. The lookbusy-based experiment follows the next steps: I first fix the CPU frequency to the lowest possible frequency in the system; then I run lookbusy with fixed amount of load for one core during timeslots of 30 seconds, starting with the maximum load and then decreasing the load gradually. After the last lookbusy run I measure the power used during an additional timeslot with *no* lookbusy load offered. I register the active cycles and the power used during each timeslot.

After taking these different samples for one frequency I move to the immediately higher frequency (we can list and change frequencies thanks to cpufrequtils) and repeat the previous steps. After going through all the available frequencies, I restart the whole process but increasing by one the number of active cores. I repeat this whole process until all the cores of the server are active. Note that when I change the frequency of the cores I change it in all of them, active or not, for consistency. Similarly, when more than one core is active, the load for all the active cores is the same.

Once explained the scheme of the experiments, I must clarify the meaning of running a timeslot with *no* load. Note that zero-load is clearly not possible as there is always going to be load in the system due to, e.g., the operating system. However, during the timeslot in which I do not run lookbusy, I measure the power corresponding to the operational conditions which are as close as possible to the ones of an idle system. Moreover, the decision of using timeslots of 30 seconds is to guarantee enough, yet not excessive, time for the measurements. In fact, as I start and stop lookbusy at the beginning and end of the timeslots, I need to ignore the first and the last few seconds of measurements in each timeslot to avoid measurement noise due to power ramps and operational transitions.

The measured values of load (in ACPS) and power in each timeslot are used to obtain a least squares polynomial fittings curve. These fittings characterize the CPU power utilization for each combination of frequency and number of active cores. I will use as *baseline power utilization* of each one of these configurations the zero-order coefficient of the polynomial of these fittings curves.

[7]https://wiki.archlinux.org/index.php/CPU_Frequency_Scaling
[8]http://www.devin.com/lookbusy.

4.3.4. Disks

The energy consumption of the hard drive was evaluated using two different scripts (for reading and writing) based on the `dd` linux command.[9] I chose `dd` as it allows to read files, write files from scratch, control the size of the blocks I write (read), control the amount of blocks written (read) and force the commit of writing operations after each block in order to reduce the effect of operating system caches and memory. I combine this tool with flushing the RAM and caches after each reading experiment.

In both the scripts I perform write (read) operations for a set of different I/O block sizes and for different data volumes to be written (read). I record the CPU active cycles, the total power and time used in each one of these operations for each combination of block size and available frequency.

Finally, I identify the contribution of the disk to the total power utilization by subtracting the contribution of both the baseline and the CPU from the measured total power.

Disk I/O experiments shed light on the relevance of the block sizes when reading or writing as well as whether there is an influence of the frequency on these operations.

4.3.5. Network card

In order to evaluate the contribution of the network card (NIC) to the energy consumption of a cloud data center server, I devised a set of experiments based on a client-server C script devised on purpose for this task.

There are a few aspects that I consider relevant in order to characterize the impact of the NIC on the total energy consumption of a server and that led me to choose these two tools. First, the ability of performing tests in which the server under study acts as sender or as receiver during a network connection, and therefore I can observe server's energy consumption while sending data or receiving it. To clarify the terms, *sender* is the server which injects traffic to the network, and *receiver* is the server which accepts traffic from the network. Second, the ability of those tools to change several parameters that I consider relevant for the energy characterization of the servers, namely, the packet size and the offered load, jointly with the frequency of the system.

The experiments consist, then, on measuring the achieved data rate, the CPU active cycles per second (ACPS) and the total energy consumption of the server under study either as sender or as receiver using different packet sizes and different transfer rates. I run each experiment multiple times for statistical consistency.

Finally, using the CPU active cycles per second which were measured during the experiment, I identify the energy consumption due to CPU. Subtracting both CPU energy consumption and the baseline energy consumption from the total energy consumption of the experiment, I can isolate the energy consumption of the network.

[9]`http://linux.die.net/man/1/dd`.

Chapter 5

Related Work

In this chapter, I focus on existing literature on EEE (§ 5.1) and energy efficiency in data centers (§ 5.2). First, in § 5.1, I will cover works related to the modeling of EEE (§ 5.1.1) then performance evaluation works for EEE (§ 5.1.2), existing coalescing-like techniques (§ 5.1.3) and finally a limited amount of works on dynamic coalescing (§ 5.1.4). Second, in § 5.2 I will summarize the works regarding the energy efficiency improvement of modern data centers and in particular, I will analyze existing models for estimating the power consumption of data centers.

5.1. Energy Efficient Ethernet

5.1.1. Modeling of EEE

Various analytical models exist in the literature for *unidirectional* links. In [64] the authors propose an analytical model that allows to compute fast the potential EEE energy saving, using simple statistical parameters for unidirectional traffic. In [69], using parameters such as the packet arrival time and the service rate of the *coalescer*, the model is able to compute the mean queue length, the mean packet delay and the delay for the downstream queue for 10 *Gbps* links. A two-state analytical model for 10GBASE-T links is presented in [58] which estimates the energy consumption of EEE links. This is a modeling tool, but it is not very accurate in case of small state transition intervals, like the standard ones. Herrería-Alonso *et al.* [42] propose and analyze a model for both legacy EEE and burst transmission with 10 *Gbps* cards. Their model estimates the energy saving using the arrival rate for Poisson traffic and the average service rate. They also propose a model with $GI/G/1$ queues for both packet and burst transmissions [43]. The model allows to compute the average delay of packets and the energy saving of the link using Poisson and deterministic traffic, but it is specifically designed for the case of 10 *Gbps* links and thus it cannot be used to estimate the energy saving of the widely used 1 *Gbps* links. Bolla et *al.* [17] show a complete framework, which accounts for the bidirectional behavior of 1 *Gbps* links. The model is evaluated with real traces but the traces are downsampled by a factor of 4 since short packet interarrivals were not allowing EEE to enter in state LPI. The main drawback of this model is that it requires as input the a priori knowledge of the packet size distribution of the trace.

In contrast to previous works I study in this thesis the special case of gigabit links, which are the most commonly used in modern data centers. I propose two models using $M/G/1$ queues which take into account the bidirectional behavior of gigabit links and estimate the energy consumption and the delay experienced by the link due to coalescing.

5.1.2. EEE performance evaluation

A few number of EEE evaluations and extension proposals have appeared during the last few years to study and improve EEE's performance. Reviriego *et al.* proved initially the inefficiency of EEE by simulating the standard on ns-2 for 100Base-T, 1000Base-T and 10GBase-T links [81]. In [79] the authors provide a first evaluation on newly released EEE NIC cards for 1 $Gbps$ links. They measure the energy consumption of the cards with real traffic and prove that: (i) the energy consumption during transitions is similar to the energy consumption in "Active" state and, (ii) great energy saving can be achieved but for very low loads, reporting saving up to 30% for 100 $Mbps$ links and up to 70% for 1 $Gbps$ links. In [75] and [83] the authors measure the energy consumption of 1 $Gbps$ EEE switches. As can be seen in their paper, state LPI consumes about 40-50% fewer energy than the active state. Additionally, both papers state that traffic loads more than 40% per port do not allow any energy saving.

My EEE model for gigabit links is evaluated using real traffic traces to feed the ns-3 simulator. I also present an economical analysis which shows the saving which we can achieve if we replace legacy Ethernet links with EEE links with coalescing.

5.1.3. Coalescing-like techniques

A few methods have been proposed to tackle the inefficiency of EEE in medium and high loads. Among them, coordinated transmission with EEE in 10 $Gbps$ links is analyzed by Reviriego *et al.* [80]. They show that for links with loads less than 50% this method can reduce the energy consumption by powering down some PHY layer components. Adaptive Link Rate (ALR) [40] is another solution that changes the link speed in low loads and therefore the link requires less energy to operate. Packet coalescing (or burst transmission) allows to extend state LPI using packet buffers in the two link edges [29]. Packet coalescing, has attracted the interest of research community since it does not require any power down of the electronics which introduces long delays. One of the first evaluations of packet coalescing for EEE is presented in [29] for 10 $Gbps$ Ethernet links. The results show that packet coalescing outperforms legacy EEE in terms of energy consumption and it overcomes the major problem of EEE, namely the overhead due to protocol state transitions (which correspond to hardware operational states). However, EEE introduces additional delay for the packets to cross the Ethernet link due to packet coalescing. For the measurements in [29], only two pairs of timeout-buffer size values are used (buffer size of 10 packets with a 12 μs timeout and buffer size of 100 packets with 120 μs timeout). The authors of [82] perform more extensive simulations on packet coalescing by using a timeout of 10 μs for testing 100 $Mbps$, 1 $Gbps$ and 10 $Gbps$ Ethernet links. In [45] the authors approximate the energy saving and the delay that the packets suffer due to coalescing over 10 $Gbps$ links. Kim et *al.* [53] present a similar mathematical analysis and evaluation based on synthetic traffic but instead they use an $M/G/1$ model.

In all the above performance evaluation works, it is assumed that traffic is unidirectional and energy saving is operated independently over the two link directions. In this thesis instead, I consider the bidirectional behavior of gigabit EEE links with coalescing.

5.1.4. Dynamic Coalescing

EEE with packet coalescing is still far from achieving energy consumption proportional to the load, i.e., *energy proportionality*. A couple of dynamic schemes tried to improve the results without success. In [44] the authors propose a dynamic scheme that tunes the coalescing buffer size. The buffer size can be adapted based on the energy consumption difference between the ideal energy proportional model (without considering transition time among states), and the actual model allowing some degree of freedom. If the degree of freedom is exceeded then the buffer size grows otherwise it shrinks.

In this thesis, I evaluate two classes of dynamic coalescing algorithms. In the first class, the dynamic coalescing algorithm adapts either the coalescing timeout or the coalescing buffer size. The event that triggers the adaptation of the corresponding parameter is either the timeout expiration, or the fill-up of one of the buffers, with no further considerations on the network performance. Moreover I study various parameters used to increase and decrease the timeout and the buffer size and conclude that this class of dynamic algorithms (based on timeout expiration or buffer fill-up) does not outperform static coalescing schemes. In the second class I design and study measurement-based coalescing control solutions that tune the coalescing parameters on-the-fly, according to the instantaneous load and the coalescing delay experienced by the packets. My results show that, by relying on run-time delay measurements, this dynamic coalescing scheme outperforms traditional static and dynamic coalescing. Notably, this scheme doubles the energy saving benefit of static EEE coalescing and allows to control the coalescing delay.

5.2. Energy Efficiency in Data centers

There is a large body of work in the field of modeling server energy consumption and its components, both theoretically and empirically. In available models, the consumption of servers follow a linear or not linear behavior depending on the load or utilization of the machine, in fact, we can find theoretical works e.g., by Wang *et al.* [92], Mishra *et al.* [65] or Beloglazov *et al.* [4], who assumed models in which energy consumption mainly depends linearly on CPU utilization. Based on the models, they proposed bin-packing-like algorithms to reduce energy consumption. Other works like the ones from Andrews *et al.* [7] or Irani *et al.* [48] proposed non-linear models, claiming that energy could be saved by running processes at the lowest possible speed. However, I experimentally show that current data center servers exhibit non-linear behaviors in terms of energy consumption and that the impact of frequency is not straightforward in modern servers.

Moving to the empirical field, we first classify works in two different groups, depending on whether they consider the effect of frequency in their analysis. I start with works not considering frequency. In this category we find articles proposing models where server components follow a linear behavior, like in [55, 61, 91] or more complex ones, like in [12, 33, 60]. In [61] Liu *et al.* proposed a simple linear model and evaluate different hardware configurations and types of workloads by varying the number of available cores, the available memory, and considering also the contribution of other components such as disks. Vasan *et al.* [91] monitored multiple servers on a datacenter as well as the energy consumption of several of the internal elements of a server. However, they considered that the behavior of this server could be approximated by a model based only on CPU utilization. Similarly, Krishnan *et al.* [55] explored the feasibility of lightweight virtual machine power metering methods and examined the contribution of some of the elements that consume energy in a server like CPU, memory and disks. Their model depends linearly on each of these components. In [33], Economou *et al.* proposed a non-intrusive method for modeling full-system energy consumption by stressing its components with different workloads. Their resulting model is also linear on the utilization of server components. Finally, Lewis *et al.* [60] and Basmasjian *et al.* [12] presented much more complex models which, apart from the contribution of different components of the server, consider extra parameters like temperature and cache misses as well as multiple cores. In particular, Lewis *et al.* [60] reported also an extensive study on the behavior of reading and writing operations in hard disk and solid state drives. I go beyond existing work by showing that, in data centers, non-linear models and a new load metric are required to improve the accuracy of energy consumption estimation. Furthermore, I complement existing studies by showing both individual and joint effects of load, I/O block sizes, network activity and CPU frequencies.

Next, I move to the works which also consider frequency in their analysis. Miyoshi *et al.* [66] analyzed the runtime effects of frequency scaling on power and energy. Brihi *et al.* [18] presented an exhaustive study of DVFS using a `cpufrequtils` as I do. Main differences with my work were that they studied four different power management policies under DVFS and centered their study on the relationship between CPU and power utilization. However, they also presented interesting results about disk consumption that match partially my results, showing a flat consumption in reading operations and variations in the writing ones that they attribute to the size of the files being written. Although it was not the main objective of their work, Raghavendra *et al.* [78] performed a per-frequency and core CPU power characterization of two different blade servers. However, they claimed that CPU power depends linearly on its utilization. The main difference with my analysis is that I consider that the load supported by a server increases with the number of active cores and, hence, this load should not be represented in percentage. Gandhi *et al.* [5] published the analysis of global energy consumption versus frequency, based on DVFS and DFS and gave some intuition about the non-linearity of this relation. However, so far there has been no work like ours, i.e., presenting a per-component analysis that allows me to enter into deeper details on the energy versus frequency analysis.

Moreover, there are studies that model the energy consumption behavior for clouds and try to balance the load in order to operate the cluster in its most efficient load-power combination. MUSE [21] is one of the first works that consider a resource management architecture for data centers. Its energy efficient approach dynamically assigns jobs to the servers based on the workload (for CPU and disk) and the potential energy consumption. The authors measure the energy consumption of servers and switches involved in the cluster and conclude that at least 29% of the energy can be saved by MUSE for typical web workloads. In [86] the authors proposed a consolidation algorithm that considers the workloads of the servers in the cloud in order to find the least possible energy consumption point. Their study shows that the energy consumption of a server using variable loads for CPU and disks has an optimal operating point. Given the data from the various servers the algorithm can estimate the ideal load distribution among the servers. The authors in [10] modeled the energy consumption of data centers equipment (i.e., servers, storage, switches) for cloud computing based on existing energy consumption measurements or publicly available data sheets for each of the components (CPU, disk, network, switches). The model estimates the energy consumption per bit from the data center to the user and further analyzes the energy consumption for different types of services, i.e., storage, software, processing. However, existing works on clouds lack experimental inputs on energy consumption. Moreover, not only in my experiments I had a complete control of servers and network and I was able to correlate activity and consumption of different components, but also I unveiled that baseline energy consumption is key to achieve good analytical estimates.

I conclude with some works that also consider frequency but do not model the energy consumption of a server. First of them, the work from Le Sueur *et al.* [87] presented an analysis of the evolution of the effectiveness of DVFS and how it is reduced in the newest and most optimized servers. They show that DVSF might be soon obsoleted by the adoption of ultra low power sleep modes. Ge *et al.* proposed PowerPack [39], a framework that includes a set of toolkits to perform an exhaustive profiling of the power utilization of servers and its components. Their analysis is centered in showing the contribution of multicore system to the efficiency of several applications and, hence, no power characterization is presented. Finally, Basmadjian *et al.* [13] published an in deep analysis of the components of a processor and its contribution to the energy consumption of the CPU, shedding some light on the behavior of multicore servers. Some of their conclusions are very relevant to my work, as they show, for instance, that the energy consumption of multiple cores performing parallel computations is not equal to the sum of the power of each of those active cores. The experiments and model of this thesis support their findings and shed light on the nature of such effect.

Part II

Energy Efficient Ethernet

Chapter 6

EEE modeling with static coalescing

The goal of using EEE is to reduce the power consumption of Ethernet links; therefore, it is of great interest to evaluate the performance of EEE under different working conditions. I derive an analytical model for EEE with bidirectional traffic with and without coalescing in § 6.1 and its extended version in § 6.2. The importance of considering bidirectional traffic, rather than a mere superposition of the two traffic flows, stems from the fact that models considering superposition of traffic cannot capture the effect of simultaneous transmissions in the two directions (e.g., they introduce delay due to the serialization of packet service for packets arriving from different directions). Moreover coalescing state C is not the superposition of the two directions but it is the same coalescing state for the two directions.

Next, based on the model of § 6.1, I will proceed with a study on the sensitivity analysis of EEE performance with respect to the coalescing parameters in § 6.3. Specifically, I want to study the impact in both energy saving and average packet delay when I modify either T_c or N_c. Thus, I apply the method of partial derivatives with respect to T_c and N_c.

6.1. Analytical Model for 1 $Gbps$ EEE links and Coalescing

I model the behavior of EEE links with coalescing assuming two steady-state $M/G/1$ queueing systems, Q_1 and Q_2 (one per link direction). To control the duration of the coalescing period I combine a buffer of N_c packets ("N policy" according to [96]) and a timeout T_c counting from the first packet that initiates coalescing ("T policy" according to [46]). I use the expression "NT policy" to refer to this class of coalescing algorithms, and I make the following assumptions for my system:

1. The system uses Poisson arrival processes with arrival rate λ_i, where i corresponds to link direction 1 or 2.

2. I consider average packet size, $E[S_p^{(i)}]$, in each link direction $i \in \{1, 2\}$ and R is the constant link speed.

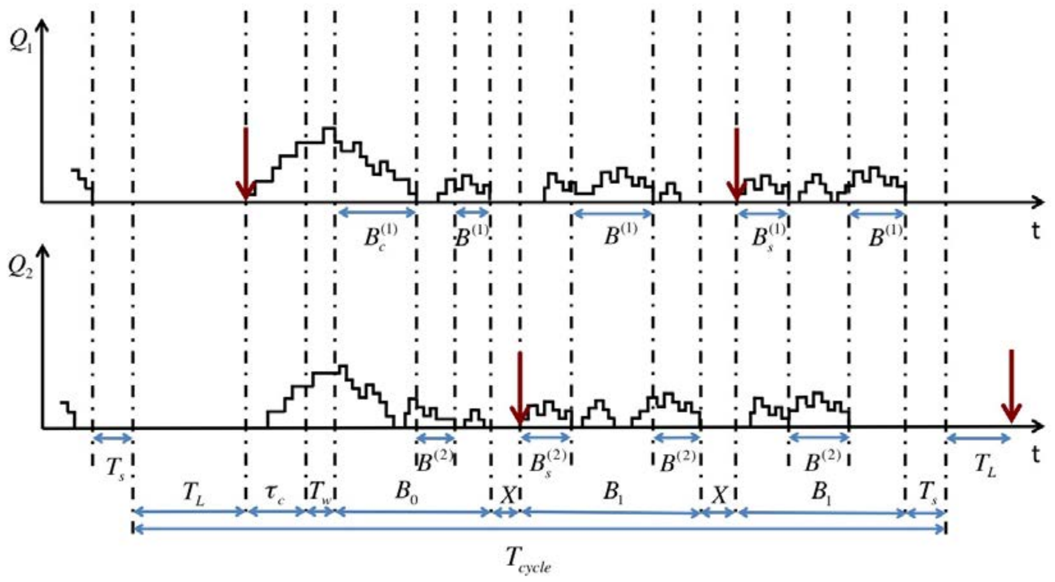


Figure 6.1: System cycle with coalescing.

3. Service times, $\sigma_i = E[S_p^{(i)}]/R$, are *i.d.* random variables with mean $E[\sigma_i] = 1/\mu_i$. The traffic load ρ_i in the link direction $i \in \{1, 2\}$ is $\rho_i = \lambda_i/\mu_i$.

4. FIFO service discipline with infinite waiting space.

Note that using Poisson arrivals is appropriate for highly aggregated traffic, e.g., when multiple TCP flows mix and cause large bursts of big packets in one link direction and small acknowledgments evenly spaced-out in the other direction [52]. The sample path of the queue can be viewed as a sequence of cycles as illustrated in Figure 6.1. A cycle starts with the packet arrival that induces the transition from state L to state C. Then, when the coalescing timer T_c expires or the number of coalesced packets reaches N_c, the link transitions to state W. Note that both transitions $L \rightarrow C$ and $C \rightarrow W$ can be caused by arrivals in either link directions. Note also that setting $T_c = 0$ and/or $N_c = 1$ yields the legacy EEE operation with no coalescing (i.e., the duration of state C is 0). In particular, transition $L \rightarrow C$ happens because of an arrival to queue Q_1 (namely, an arrival in direction 1) with probability $P_1 = \lambda_1/(\lambda_1 + \lambda_2)$, or because of an arrival to Q_2 (i.e., in direction 2) with probability $P_2 = 1 - P_1$.

The coalescing interval is followed by a busy period with the link in state A, whose duration is denoted by B_0. This initial busy period is followed by a random number ψ of sleep/active interval pairs, with each pair corresponding to an arrival in state S in either direction 1 or 2, before a time T_s has elapsed.

Note that the sleep time is then reduced to the random time interval between the start of state S and the next frame arrival, i.e., it is upper bounded by T_s. Finally, a sleep interval of duration T_s precedes the idle period T_L, whose random duration corresponds to the time interval before the beginning of a new cycle, i.e., before the next arrival in the system. I denote the length of a cycle by T_{cycle} and its average by $E[T_{cycle}]$.

Using results from renewal theory [54], I can focus on the system cycle, and compute the fraction of time spent in each link state as the ratio between the average time in each state in a cycle and the average cycle duration. I denote the average fraction of time spent in state α as η_α, for $\alpha \in \{A, C, L, S, W\}$ and $\eta_{LPI} = \eta_L + \eta_C$ is the fraction of time spent in power save mode. These fractions of time represent state probabilities in an ergodic system. The following theorem shows how to compute the average duration of a system cycle.

Theorem 1. *For bidirectional EEE links in which arrivals in S are served immediately, the average cycle duration is given by:*

$$\begin{aligned} E[T_{cycle}] = & \quad (T_w + E[\tau_C])\left[1 + \frac{1}{\lambda_1+\lambda_2}\left(\frac{\lambda_1\rho_1}{1-\rho_1} + \frac{\lambda_2\rho_2}{1-\rho_2}\right)\right] \\ + & \quad \frac{e^{(\lambda_1+\lambda_2)T_s}}{\lambda_1+\lambda_2}\left[1 + \frac{\rho_1}{1-\rho_1} + \frac{\rho_1^2(2-\rho_1)(\lambda_1\rho_2+\lambda_2)}{2\lambda_1(1-\rho_1\rho_2)(1-\rho_1)^2}\right. \\ + & \quad \left.\frac{\rho_2}{1-\rho_2} + \frac{\rho_2^2(2-\rho_2)(\lambda_2\rho_1+\lambda_1)}{2\lambda_2(1-\rho_1\rho_2)(1-\rho_2)^2}\right] \end{aligned} \tag{6.1}$$

with $\rho_i = \lambda_i/\mu_i$, and $\mu_i = R/E[S_p^{(i)}]$, $i \in \{1, 2\}$.

Proof: Consider the different intervals included in T_{cycle}, starting with the beginning of an L interval. The cycle is composed by the following elements: (i) an interval in state L, with random duration T_L until the first arrival to Q_1 or Q_2; (ii) a coalescing interval C of duration $\tau_C \leq T_c$; (iii) a wake-up interval of fixed duration T_w; (iv) an interval B_0 lasting till the first epoch at which both queues Q_1 and Q_2 are empty; (v) an interval $X < T_s$ with exactly one arrival at time X, followed by an interval B_1 lasting until both queues are empty again; (vi) and finally a sleep interval of fixed duration T_s which triggers a new state L. Element (v) is optional, since it occurs only if there is one arrival within T_s seconds after B_0. Moreover, element (v) can repeat $\psi \geq 0$ times, until the idle interval following B_1 is longer than T_s. Each repetition of X and B_1 exhibits the same distribution because of the memoryless property of Poisson arrivals. However, busy intervals B_0 and B_1 can start either because of Q_1 or Q_2 activities, the two cases leading to different *conditional* average durations, as I will discuss in the following. Overall, the total cycle duration is:

$$T_{cycle} = T_L + \tau_C + T_w + B_0 + \psi(X + B_1) + T_s. \tag{6.2}$$

Note that $T_L + \tau_C$ is the time spent in state LPI during a cycle, while $B_0 + \psi B_1$ is the total time during which the link is active, and $T_s + \psi X$ is the time spent in state S. Let me now compute the average value for each of the elements composing the system cycle.

Interval T_L. State L lasts until the first packet arrival to Q_1 or Q_2. Since both arrival processes are Poisson, the first arrival behaves as the first of a Poisson flow with rate $\lambda_1 + \lambda_2$, i.e., with the following probability distribution:

$$f_{T_L}(t) = (\lambda_1 + \lambda_2)e^{-(\lambda_1+\lambda_2)t}, \quad t \geq 0; \tag{6.3}$$

and its average is then:

$$E[T_L] = \frac{1}{\lambda_1 + \lambda_2}. \tag{6.4}$$

Interval τ_C. For large values of N_c, I can approximate the duration of state C as the minimum time before $N_c - 1$ packet arrivals occur in either direction 1 or 2, and T_c. Let me denote with τ_{c_1} and τ_{c_2} the time before $N_c - 1$ arrivals appear in direction 1 or 2, respectively. Thereby, the time spent in coalescing is $\tau_C = \min\{\tau_{c_1}, \tau_{c_2}, T_c\}$. Recalling that the cumulative distribution function (CDF) of the minimum of n independent random variables is given by the following formula:

$$F_{\min\limits_{i=1\ldots n}\{X_i\}}(x) = 1 - \prod_{i=1}^{n}(1 - F_{X_i}(x)), \tag{6.5}$$

and considering that the distributions of τ_{c_1}, τ_{c_2}, and T_c are as follows:

$$F_{\tau_{c_1}}(t) = \quad u(t)\left[1 - \sum_{k=0}^{N_c-2} \frac{(\lambda_1\, t)^k}{k!} e^{-\lambda_1\, t}\right]; \tag{6.6}$$

$$F_{\tau_{c_2}}(t) = \quad u(t)\left[1 - \sum_{k=0}^{N_c-2} \frac{(\lambda_2\, t)^k}{k!} e^{-\lambda_2\, t}\right]; \tag{6.7}$$

$$F_{T_c}(t) = \quad u(t - T_c); \tag{6.8}$$

where $u(t)$ is the unit step function, then the CDF of τ_C is given by:

$$F_{\tau_C}(t) = 1 - u(T_c - t)\left[\sum_{k=0}^{N_c-2} \frac{(\lambda_1 t)^k}{k!} e^{-\lambda_1 t}\right]\left[\sum_{k=0}^{N_c-2} \frac{(\lambda_2 t)^k}{k!} e^{-\lambda_2 t}\right], \tag{6.9}$$

where I used $1 - u(t) = u(-t)$. The average coalescing time is then as follows:

$$E[\tau_C] = \int_0^{T_c} t \cdot dF_{\tau_C}(t). \tag{6.10}$$

However, assuming that I can tune N_c and T_c in a way that the coalescing queues do not fill completely with probability almost one, then $E[\tau_C] \simeq T_c$. Note that the latter assumption is realistic for actual implementations since the power saving increases with N_c, although delay increases too and I want to bound it to T_c.

Interval T_w. The wake-up interval which precedes the first busy interval has fixed duration T_w.

Interval B_0. This interval is composed of various subparts, as shown in Figure 6.1. After the link transitions to state A from state C, at least one queue is not empty. The system remains busy until *both* queues are empty again. Let me observe the system from the viewpoint of the queue that received the packet that caused the beginning of the the coalescing state (transition $L \to C$). I denote with $B_c^{(i)}$, $i \in \{1, 2\}$, the first busy period seen at queue i only, after the transition $C \to A$. This is the busy period of an $M/G/1$ queue, for which the average depends on the arrival rate λ_i, the mean service time $E[S_p^{(i)}]/R$, and the queue size $Z_c^{(i)}$ at the beginning of the busy period [54]:

$$E[B_c^{(i)}] = \frac{E\left[Z_c^{(i)}\right] E\left[S_p^{(i)}\right]/R}{1-\rho_i} = \frac{E\left[Z_c^{(i)}\right]\rho_i}{\lambda_i(1-\rho_i)}. \tag{6.11}$$

If queue i is the one who received the packet that triggered the transition $L \to C$, then $E\left[Z_c^{(i)}\right] = 1 + \lambda_i\left(T_w + E[\tau_C]\right)$, i.e., the initial queue size equals the arrival that triggers the coalescing timer, plus the average number of Poisson arrivals during the average coalescing time $E[\tau_C]$ and the wake-up interval T_w. The probability that queue Q_1 is the one who initiates the coalescing procedure is simply given by the probability of having a Poisson arrival with rate λ_1 before a Poisson arrival with rate λ_2, counting from the beginning of state L. I.e.:

$$Pr(Q_1 \text{ triggers coalescing}) = \frac{\lambda_1}{\lambda_1+\lambda_2}; \tag{6.12}$$

$$Pr(Q_2 \text{ triggers coalescing}) = \frac{\lambda_2}{\lambda_1+\lambda_2}. \tag{6.13}$$

With no loss of generality, assume now that queue Q_1 triggers the coalescing. Therefore, as observed from Q_1, the system goes through a busy period $B_c^{(1)}$, at the end of which Q_1 is empty with probability 1, while Q_2 is empty with probability $1-\rho_2$. If Q_2 is not empty, let me observe the followup in the evolution of the system from the viewpoint of Q_2: there is a busy period $B^{(2)}$ for Q_2, at the end of which Q_2 will be empty, while Q_1 can be empty with probability $1-\rho_1$. The process can replicate by alternating busy intervals $B^{(1)}$ and $B^{(2)}$, i.e., I alternate the observation of the system from the viewpoint of a queue or the other, until the observation of the queue status at the end of a busy period reveals that both queues are empty. At that point we have a transition $A \to S$. Busy periods $B^{(i)}$ have different average duration with respect to $B_c^{(i)}$. In fact, the initial backlog of the queue in $B^{(i)}$ is not $Z_c^{(i)}$. Using the Pollaczek-Khinchin mean formula to estimate the average backlog of an $M/G/1$ queue at a random observation point, I would have $\lambda_i\, E\left[S_p\right]/R + \frac{\lambda_i^2\, E[S_p^2]/R^2}{2(1-\rho_i)}$ as initial backlog. However, I am interested in the conditional initial backlog $Z^{(i)}$, given that the observed queue is not empty (otherwise there is no busy period), which happens with probability ρ_i. Therefore I have $E\left[Z^{(i)}\right] = 1 + \frac{\rho_i}{2(1-\rho_i)}$, and the observed busy periods are given by:

$$E\left[B^{(i)}\right] = \rho_i \frac{2-\rho_i}{2\lambda_i(1-\rho_i)^2}. \tag{6.14}$$

Following the above procedure, one can compute the average duration of the first period after state C during which either queue 1 or 2 are busy:

$$\begin{aligned} E[B_0] =& \frac{1}{\lambda_1+\lambda_2}\left[\lambda_1 E\left[B_c^{(1)}\right] + \lambda_2 E\left[B_c^{(2)}\right]\right. \\ +& \left.\frac{\rho_1(\lambda_1\rho_2+\lambda_2)E\left[B^{(1)}\right]}{1-\rho_1\rho_2} + \frac{\rho_2(\lambda_2\rho_1+\lambda_1)E\left[B^{(2)}\right]}{1-\rho_1\rho_2}\right]. \end{aligned} \tag{6.15}$$

Interval X. The interval X between the end of a busy period and the beginning of the next busy period is exponentially distributed with rate $\lambda_1+\lambda_2$, given that the next arrival occurs within T_s seconds:

$$f_X(t) = \frac{(\lambda_1+\lambda_2)e^{-(\lambda_1+\lambda_2)t}}{1-e^{-(\lambda_1+\lambda_2)T_s}}, \quad t \in [0, T_s].$$

Accordingly, the average of X is as follows:

$$E[X] = \frac{1}{\lambda_1+\lambda_2} - \frac{T_s}{e^{(\lambda_1+\lambda_2)T_s}-1}.$$

Interval B_1. Similarly to the case of B_0, this interval is composed by various subparts. The first part is $B_s^{(i)}$, which is the first busy interval on either Q_1 or Q_2 after the period X. Since packets are served immediately when they arrive in state S, the initial backlog is exactly 1. For the rest, the computation of $B_s^{(i)}$ is analogue to the one of $B_c^{(i)}$:

$$E[B_s^{(i)}] = \frac{\rho_i}{\lambda_i(1-\rho_i)}.$$

The following alternating busy intervals are exactly like for the case of B_0, i.e., we have intervals $B^{(1)}$ and $B^{(2)}$.

Considering that each interval B_1 starts because of an arrival in Q_1 or Q_2 before T_s expires, and since arrivals for Q_1 and Q_2 in state S are independent and both follow a Poisson process, the probability of starting an interval B_1 due to arrivals for Q_i is $\frac{\lambda_i}{\lambda_1+\lambda_2}$.

Putting together the pieces, the average of B_1 is as follows:

$$\begin{aligned} E[B_1] =& \frac{1}{\lambda_1+\lambda_2}\left[\lambda_1 E\left[B_s^{(1)}\right] + \lambda_2 E\left[B_s^{(2)}\right]\right. \\ +& \left.\frac{\rho_1(\lambda_1\rho_2+\lambda_2)E\left[B^{(1)}\right]}{1-\rho_1\rho_2} + \frac{\rho_2(\lambda_2\rho_1+\lambda_1)E\left[B^{(2)}\right]}{1-\rho_1\rho_2}\right]. \end{aligned} \tag{6.16}$$

Number of repetitions ψ. Busy intervals B_1 occur if the residual interarrival time at the end of the previous busy interval is shorter than T_s. Since arrivals are Poisson, the probability of having no arrivals in any link direction in T_s is $P_0 = e^{-(\lambda_1+\lambda_2)T_s}$. Thereby, the number $\psi \geq 0$ of busy periods of type B_1 in a cycle, i.e., not counting B_0, can be seen as the number of consecutive successes of a geometric random variable ψ with success probability $1-P_0$. Hence, its average value is:

$$E[\psi] = \frac{1 - P_0}{P_0} = e^{(\lambda_1+\lambda_2)\,T_s} - 1.$$

Interval T_s. The sleep interval which follows the last busy interval before entering state L has fixed duration T_s.

Average cycle duration. Putting together the results obtained for the cycle components, after some algebraic elaboration, result (6.1) follows.

■

Corollary 1. *The fraction of time spent in LPI is:*

$$\eta_{LPI} = \frac{\frac{1}{\lambda_1+\lambda_2} + E[\tau_C]}{E[T_{cycle}]}. \tag{6.17}$$

Proof: The time spent in LPI corresponds to states L and C, i.e., intervals T_L and τ_C, as described in the proof of Theorem 1; therefore the proof follows. ■

Corollary 2. *The fraction of time spent in $WakeUp$ state is given by:*

$$\eta_W = \frac{T_w}{E[T_{cycle}]}. \tag{6.18}$$

Proof: The time spent in $WakeUp$ state is constant in each cycle and corresponds to the interval T_w. Therefore the proof follows. ■

Corollary 3. *The fraction of time spent in $Sleep$ state is given by:*

$$\eta_S = \frac{E[X]E[\psi] + T_s}{E[T_{cycle}]}. \tag{6.19}$$

Proof: The time spent in $Sleep$ state corresponds to intervals X, which are repeated ψ times, plus a complete sleep interval of T_s seconds, occurring once in a cycle, just before entering state L. Considering the proof of Theorem 1, the proof follows. ■

Corollary 4. *The fraction of time spent in $Active$ state is given by:*

$$\eta_A = \frac{E[B_0] + E[B_1]E[\psi]}{E[T_{cycle}]}. \tag{6.20}$$

Proof: The time spent in $Active$ state is given by the sum of busy intervals during which at least one network interface transmits. Therefore, considering the proof of Theorem 1, the proof follows. ■

6.1.1. Power Saving

Using the results of the analysis carried out for the cycle duration, I can now compute the power saving factor Φ achieved by EEE with or without coalescing.

Theorem 2. *The average power consumption achieved by EEE is proportional to the fraction of time spent in LPI:*

$$\Phi \propto \eta_{LPI}. \tag{6.21}$$

Proof: The average power consumption over a system cycle, is computed by considering that the power consumption in states W (namely $P^{(W)}$) and S (namely $P^{(S)}$) is practically the same as in state A (namely $P^{(A)}$), while in LPI (i.e., $P^{(LPI)}$ in states L and C), the power consumption decreases by a factor $k \simeq 10$, as experimentally shown by Reviriego *et al.* [79]. In legacy gigabit cards, the power consumption $P^{(legacy)}$ is practically constant and equals the one consumed in state A in an EEE card. Therefore I have the following expression for the power saving factor Φ:

$$\Phi = 1 - \frac{\sum_{\alpha \in \{A,S,LPI,W\}} \eta_\alpha P^{(\alpha)}}{P^{(legacy)}} \simeq \frac{k-1}{k} \cdot \eta_{LPI}. \tag{6.22}$$

Φ is thus shown to be proportional to η_{LPI} with a proportionality factor $(k-1)/k \simeq 0.9$. ■

The power saving achieved with EEE is proportional to η_{LPI} and the values of Φ and η_{LPI} are similar. As a consequence, in the rest of the paper I will refer to *power saving performance* either in case of actual power saving figures or when discussing η_{LPI} values.

6.1.2. Packet Delay

Theorem 3. *The average packet delay in Q_i is given by:*

$$D^{(i)} = \frac{\sum_{\alpha \in A,S,W,L,C} n_\alpha^{(i)} D_\alpha^{(i)}}{n_{cycle}^{(i)}}, \quad i \in 1,2; \tag{6.23}$$

where $n_\alpha^{(i)}$ is the amount of packets received in each state $\alpha \in \{A,S,W,L,C\}$ and link direction $i \in \{1,2\}$ and $n_{cycle}^{(i)}$ is the total amount of packets received per link direction $i \in \{1,2\}$.

Proof: Next, I identify the average packet delay of a packet in the queue depending on the state α at arrival time. Note that the resulting delays are different in the two link directions.

Delay D_A of packets arriving in A. If a packet arrives in state A, I can use results for M/G/1 queues. Therefore, the average waiting time can be simply computed by means of the P-K formula [54]:

$$D_A^{(i)} = \frac{\lambda_i E[\sigma_i^2]}{2(1-\rho_i)}, \quad i \in \{1,2\}, \tag{6.24}$$

where $\sigma_i = S_p^{(i)}/R$ is the random service time, with $E[\sigma_i] = 1/\mu_i$. The statistics of σ_i depends on the packet size distribution. For the sake of simplicity, here I assume that packet size is constant and equal to $1/\mu_i$, so that I use $E[\sigma_i^2] = 1/\mu_i^2$, which yields $D_A^{(i)} = \frac{\rho_i}{2\mu_i(1-\rho_i)}$.

Delay D_S of packets arriving in S. If a packet arrives while the device is in state S, the device will directly transition to state A and the packet is immediately served. Thus, the delay is $D_S = 0$.

Delay D_L of packets arriving in L. Only one packet can arrive while the device is in state L, which triggers an immediate transmission to state C. The delay of this packet is at most $T_c + T_w$, in case of scarce traffic which yields the expiration of the coalescing timeout. More in general, the average queuing delay experienced by this packet is the sum of the average coalescing time, given in Eq. (6.10), plus the constant wake-up time T_w:

$$D_L = E[\tau_C] + T_w. \tag{6.25}$$

Delay D_C of packets arriving in C. When a packet arrives in state C, it suffers from (i) the residual coalescing interval, (ii) the constant wake-up interval T_w, and (iii) the time to process and transmit packets already present in the queue at the arrival epoch of the new packet.

Considering that Poisson arrivals are uniformly distributed over time, I estimate the average residual coalescing time as $E[\tau_C]/2$. Correspondingly, the average queue size at the arrival epoch is $\lambda_i E[\tau_C] + \frac{\lambda_i}{\lambda_1+\lambda_2}$ for an arrival in direction $i \in \{1,2\}$, where the second term represents the probability that the packet triggering the $L \to C$ transition belongs to direction i. The average cumulative serving time for those packets is then $\rho_i \left(\frac{E[\tau_C]}{2} + \frac{1}{\lambda_1+\lambda_2}\right)$. Therefore, the delay suffered by a packet arriving in state C is, on average:

$$D_C^{(i)} = T_w + \frac{E[\tau_C]}{2} + \rho_i\left(\frac{E[\tau_C]}{2} + \frac{1}{\lambda_1+\lambda_2}\right), \; i \in \{1,2\}. \tag{6.26}$$

Considering that the delay D_C is affected by the arrival rate, this delay assumes different average values for packets sent in the two different link directions.

Delay D_W of packets arriving in W. If a packet arrives while the device is in state W, the delay is composed of (i) the average residual wake-up time (for uniformly distributed Poisson arrivals, this equals $T_w/2$), and (ii) the required time to serve all packets arrived earlier, since the beginning of state C: $\frac{\lambda_i}{\lambda_1+\lambda_2} + \lambda_i(E[\tau_C] + T_w/2)$ packets, on average (again the first term is due to the packet which triggers the $L \to C$ transition). Therefore, the delay is:

$$D_W^{(i)} = \frac{T_w}{2} + \rho_i\left(\frac{1}{\lambda_1+\lambda_2} + E[\tau_C] + T_w/2\right), \; i \in \{1,2\}. \tag{6.27}$$

This delay is different for different traffic directions, as it was the case for D_A and D_C.

Average delay of a packet. To find the average delay that a packet can suffer, I need to compute the probability that a packet arrives in any of the possible EEE states. For each link direction, these probabilities can be seen as the average number of packets received in each of the different states divided by the average number of arrivals in a system cycle.

There is only one packet per cycle arriving in state L: it belongs to link direction i with probability $\lambda_i/(\lambda_1+\lambda_2)$, which is then the average number of packets received in state L in direction i, namely $n_L^{(i)}$. Similarly, since there are ψ arrivals per cycle in state S, the average number of packet arrivals in direction i in state S is given by $n_S^{(i)} = \frac{\lambda_i}{\lambda_1+\lambda_2}E[\psi]$.

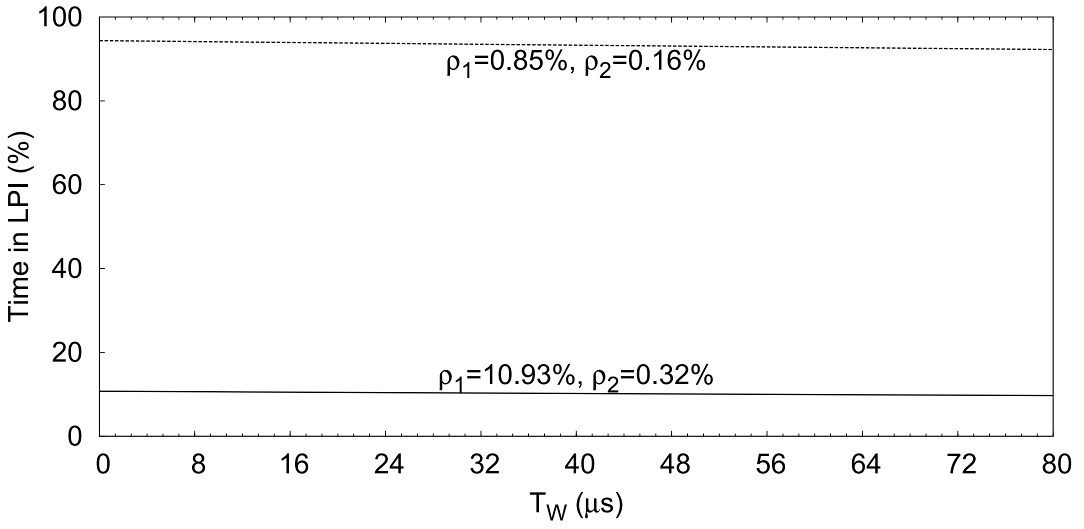
Since arrivals follow a Poisson process, packets received in link direction i during the fixed-length wake-up interval, which is present only once in a cycle, are $n_W = \lambda_i T_w$, on average. Similarly, the number of arrivals in direction i during the coalescing interval (state C) is $n_C = \lambda_i E[\tau_C]$, on average. Eventually, arrivals in direction i in state A are the total number of (Poisson) arrivals in a cycle less the arrivals in the other states, i.e., $n_A^{(i)} = \lambda_i E[T_{cycle}] - n_S^{(i)} - n_W^{(i)} - n_L^{(i)} - n_C^{(i)}$. As a result, the corresponding average delay that a packet suffers in direction i is as follows:

$$D_i = \frac{\sum_{\alpha\in\{A,S,L,C,W\}} n_\alpha^{(i)} D_\alpha^{(i)}}{\lambda_i E[T_{cycle}]}, \quad i \in \{1,2\}. \tag{6.28}$$

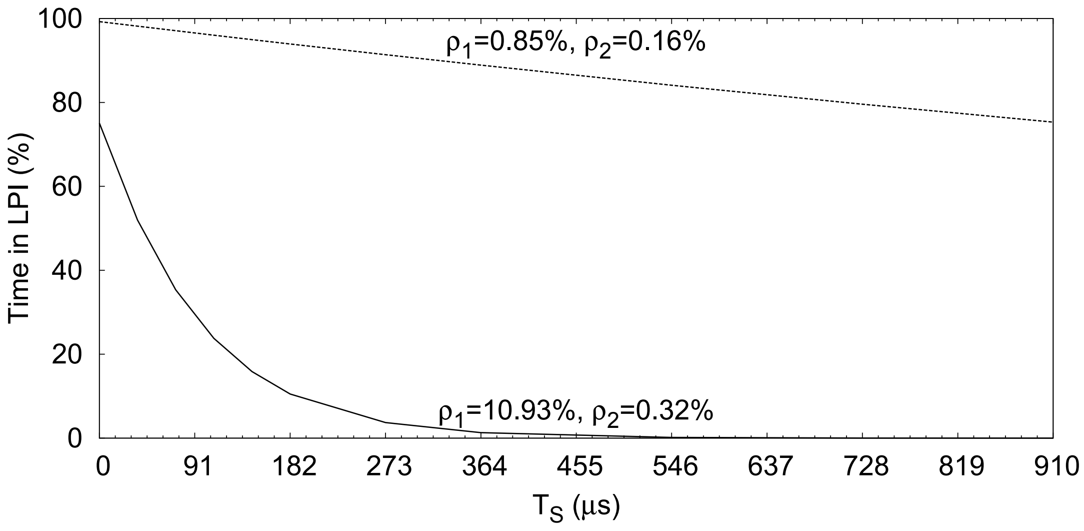
■

6.1.3. Impact of EEE parameters

I use here the model to establish a reference for the evaluation of energy saving in the system. To achieve this goal, consider that an ideal, *energy proportional* system, would allow to switch instantaneously to state L ($A \to L$) as soon as there is no traffic to serve. Similarly, an ideal system would allow to switch back to state A ($C \to A$) as soon as a packet arrives. Therefore, an ideal system would be characterized by $T_w = T_s = 0$. In contrast, a real system has non-zero transition times T_w and T_s and therefore it experiences less energy saving and less chances to enter state L. In Figures 6.2(a) and 6.2(b) I use the model results to depict the impact of T_w and T_s on the energy saving performance stretching their values from 0 to 5 times their standard value. Figures 6.2(a) and 6.2(b) refer to two specific cases, respectively: a typical off-peak hour, during which the load in the most loaded link direction is $\sim 1\%$ of the link capacity, and a typical peak-hour load, during which the link load is $\sim 11\%$ of its capacity. Noticeably, the achievable energy saving (expressed in terms of time spent in states L and C, which is η_{LPI}) is not much affected by the wake-up transition time T_w, while the sleep transition time T_s has a strong impact. From the figures, I notice that small values of T_s would dramatically boost the energy saving performance. However, due to technology constraints, the EEE standard specifies that T_s cannot be smaller that 182 μs. Similarly, the standard imposes $T_w \geq 16$ μs. In the following, when evaluating the energy saving achieved with EEE, I use the minimum standard values $T_w = 16$ μs and $T_s = 182$ μs, and I show that coalescing techniques can be used to reduce the gap between real and ideal energy saving performances without imposing stringent hardware requirements.



(a) Energy saving vs. T_w.



(b) Energy saving vs. T_s.

Figure 6.2: Energy saving versus the transitioning parameters.

6.1.4. Performance dependency on $E[\tau_C]$

With the model described above, it is straightforward to establish a direct relation between average delay and average time spent in coalescing. The resulting expression has the following quadratic form:

$$D = \alpha_1 + \alpha_2 E[\tau_C] + \alpha_3 E^2[\tau_C]; \tag{6.29}$$

where α_1, α_2, and α_3 are non-negative coefficients that do not depend on $E[\tau_C]$. Similarly, it is possible to compute a direct relation between the gain Φ and $E[\tau_C]$:

$$\Phi = (1-m)(\eta_L + \eta_C) = (1-m)\frac{\beta_1 + E[\tau_C]}{\beta_2 + \beta_3 E[\tau_C]}; \tag{6.30}$$

where $m \in [0,1]$, $0 < \beta_1 \ll \beta_2$, and $\beta_3 > 1$ (so that $\beta_3 < \beta_2/\beta_1$, which guarantees that $\frac{d\Phi}{dE[\tau_C]} > 0$ and $\frac{d^2\Phi}{dE^2[\tau_C]} < 0$) are parameters that do not depend on $E[\tau_C]$. As a consequence, both average delay and gain are increasing functions of $E[\tau_C]$. Therefore, to maximize the gain, it is also necessary to maximize the average delay.

The above results, although computed from a model for static coalescing, reveal that changing $E[\tau_C]$ is the only way to tune delay and power saving. In fact, adapting the coalescing parameters online only affects directly $E[\tau_C]$, while the structure of the system cycle remains the same, including the dependencies on $E[\tau_C]$. However, in that case, the averages used in the model should be computed over N_c and/or T_c seen as random variables, which is out of the scope of this work.

Another important observation from Eqs. (6.29) and (6.30) is the tradeoff between $E[\tau_C]$ and the delay, and between $E[\tau_C]$ and $\eta_{LPI} = \eta_L + \eta_C$. The formulas reveal that increasing $E[\tau_C]$ increases the delay D super-linearly, while the gain Φ (and therefore the energy saving η_{LPI}) increases to the asymptotic value $(1-m)/\beta_3$ according to a concave downward function. So, I conclude that the more energy saving I achieve the higher the delay introduced to the packets on average and vice versa, and small energy saving increases can cause large delays when approaching the asymptotic achievable gain.

6.2. Extended EEE model with Coalescing for 1 $Gbps$ links

This section presents an improved version of the analytical model presented in § 6.1. In this new model I modify the way I compute the average time in which the link remains in each of the different states of the system cycle and propose a new and more accurate expression for the computation of the time in which the link remains in the coalescing state. The new model allows to improve the delay estimation, study the impact of EEE parameters on energy saving, and show the intrinsic dependency of achieved gain from average delay. Moreover, I compare the results of the two models.

Let me assume that the cycle starts after the end of state A ("Active"). After the interval T_s elapses the link enters in state L until the reception of the first packet, in either of the two link directions, which triggers the transition to state C ("Coalescing"). After coalescing, the link wakes up and resumes, serving the buffered packets and the ones that arrive during service time. Upon completion of the service the cycle ends and a new cycle starts. Differently from what presented in § 6.1, here I consider renewal cycles that might not include states L, C and W (their duration can be zero).

Indeed, leaving state A represents a renewal point [54] in the system, since at that point the process evolves with no memory of the past activity. Therefore, I can analyze the system in cycles, starting from state S (transition $A \rightarrow S$) instead of starting from state C (transition $L \rightarrow C$) like I presented in § 6.1.

6.2.1. Derivation of $E[\tau_\alpha]$ for $\alpha \in \{A, L, C, S, W\}$

Mean duration of state S***,*** $E[\tau_S]$. State S lasts at most T_s. However, according to the standard, if a packet arrives during state S the link resumes immediately its service without complying the restrictions of the NT policy (i.e., without passing through states L and C) and without any wake-up delay (state W). To estimate the average time spent in state S in a cycle, I consider that arrivals in both link directions are independent Poisson processes, so that the interval between any two packet arrivals in the system is exponential with rate $\lambda_1+\lambda_2$. Therefore, with probability $P_S = e^{-(\lambda_1+\lambda_2)T_s}$ there are no arrivals in T_s, which causes the transition $S \rightarrow L$. Instead, with probability $1 - P_S$ the time spent in state S is a truncated exponential. The resulting average is as follows:

$$\begin{aligned} E[\tau_S] &= P_S T_s + \frac{\int_0^{T_s} I(\lambda_1+\lambda_2)e^{-(\lambda_1+\lambda_2)I}dI}{\int_0^{T_s}(\lambda_1+\lambda_2)e^{-(\lambda_1+\lambda_2)I}dI} \cdot (1-P_S) \\ &= \frac{1}{\lambda_1+\lambda_2}\left(1 - e^{-(\lambda_1+\lambda_2)T_s}\right). \end{aligned} \tag{6.31}$$

Mean duration of state L***,*** $E[\tau_L]$. The link will remain in state L as long as there is no arrival in either of the two link directions. The resulting mean duration, considering that with probability $1 - P_S$ there is no state L at all in a cycle, is:

$$E[\tau_L] = \frac{1}{\lambda_1+\lambda_2}P_S + 0 \cdot (1-P_s) = \frac{1}{\lambda_1+\lambda_2}e^{-(\lambda_1+\lambda_2)T_s}. \tag{6.32}$$

Mean duration of state C***,*** $E[\tau_C]$. To control the duration of the coalescing period I combine a buffer of N_c packets ("N policy" according to [96]) and a timeout T_c counting from the first packet that initiates coalescing ("T policy" according to [46]). I use the expression "NT policy" to refer to this type of coalescing. The duration of the coalescing period is limited by the NT policy according to the following two cases: (i) a coalescing buffer fills up due to $N_c - 1$ arrivals in the link direction that triggered the transition $L \rightarrow C$ or N_c arrivals in the other direction (*N-event*, see bottom of Figure 6.3); (ii) a timeout occurs: no more than $N_c - 2$ packets arrive in the link direction that triggered the transition $L \rightarrow C$ and no more than $N_c - 1$ packets arrive in the other direction (*T-event*, see top of Figure 6.3).

With Poisson arrivals, the probability $F_n^{(i)}(t)$ to have at least n arrivals in queue Q_i in interval

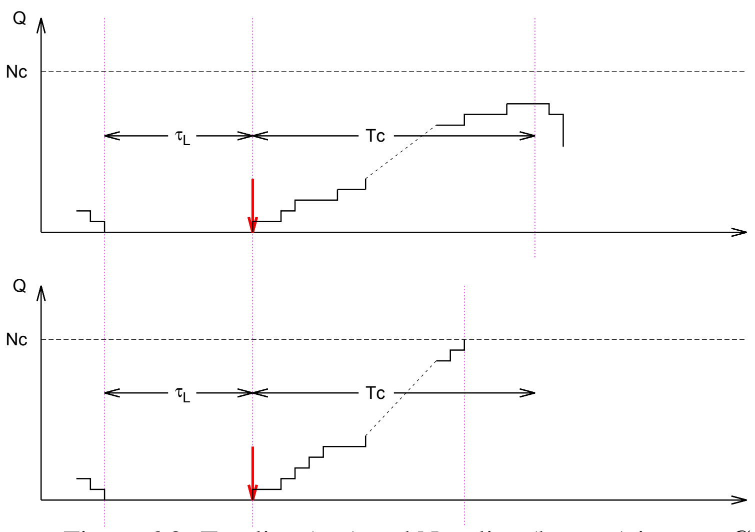


Figure 6.3: T-policy (top) and N-policy (bottom) in state C.

$[0, t \geq 0]$ is given by the following formula:

$$F_n^{(i)}(t) = 1 - \sum_{k=0}^{n-1} \frac{(\lambda_i t)^k}{k!} e^{-\lambda_i t} = Pr(N^{(i)}(t) \geq n), i \in \{1,2\}. \tag{6.33}$$

The probability of having exactly n packets in t seconds is given by $Pr(N^{(i)}(t) = n) = F_n^{(i)}(t) - F_{n+1}^{(i)}(t)$. With the above, as shown in Appendix B, the time spent in state C, given that there is no arrival in T_s (transition $W \to A$), is:

$$\begin{aligned}
E[\tau_C|_{W\to A}] &= \frac{\lambda_1 E[\tau_C|_{W\to A}, Q_1]}{\lambda_1+\lambda_2} + \frac{\lambda_2 E[\tau_C|_{W\to A}, Q_2]}{\lambda_1+\lambda_2} \\
&= \frac{\lambda_1}{\lambda_1+\lambda_2}\Bigg[\sum_{k=0}^{N_c-1}\int_0^{T_c} \frac{\lambda_1^{N_c-1}\lambda_2^k\, t^{N_c-1+k} e^{-(\lambda_1+\lambda_2)t}}{(N_c-2)!\; k!} dt \\
&+ \sum_{k=0}^{N_c-2}\int_0^{T_c} \frac{\lambda_1^k\,\lambda_2^{N_c}\, t^{N_c+k} e^{-(\lambda_1+\lambda_2)t}}{(N_c-1)!\; k!} dt \\
&+ T_c \sum_{n=0}^{N_c-2} \frac{(\lambda_1 t)^n e^{-\lambda_1 t}}{n!} \sum_{m=0}^{N_c-1} \frac{(\lambda_2 t)^m e^{-\lambda_2 t}}{m!}\Bigg] \\
&+ \frac{\lambda_2}{\lambda_1+\lambda_2}\Bigg[\sum_{k=0}^{N_c-1}\int_0^{T_c} \frac{\lambda_1^k\,\lambda_2^{N_c-1}\, t^{N_c-1+k} e^{-(\lambda_1+\lambda_2)t}}{(N_c-2)!\; k!} dt \\
&+ \sum_{k=0}^{N_c-2}\int_0^{T_c} \frac{\lambda_1^{N_c}\,\lambda_2^k\, t^{N_c+k} e^{-(\lambda_1+\lambda_2)t}}{(N_c-1)!\; k!} dt \\
&+ T_c \sum_{n=0}^{N_c-1} \frac{(\lambda_1 t)^n e^{-\lambda_1 t}}{n!} \sum_{m=0}^{N_c-2} \frac{(\lambda_2 t)^m e^{-\lambda_2 t}}{m!}\Bigg],
\end{aligned} \tag{6.34}$$

where $E[\tau_C|_{W\to A}, Q_i]$ is the average duration of τ_C when the arrival that triggers state C occurs in queue Q_i.

To estimate the average coalescing duration during a cycle I consider that the probability of no arrival is P_S and that with probability $1 - P_S$ the duration of state C will be zero:

$$E[\tau_C] = E[\tau_C|_{W\to A}] \cdot e^{-(\lambda_1+\lambda_2)T_s}. \tag{6.35}$$

Mean duration of state W***,*** $E[\tau_W]$**.** With probability P_S the link does not have arrivals in state S and the duration of state W is T_w, otherwise is zero. Thus, on average I have:

$$E[\tau_W] = T_w \cdot e^{-(\lambda_1+\lambda_2)T_s}. \tag{6.36}$$

Mean duration of state A***,*** $E[\tau_A]$**.** The link can transit to state A either from state W ($W \to A$) or from state S ($S \to A$). In both cases, state A is composed by various busy sub-parts in either of the two link directions. The analytic expression for $E[\tau_A]$ is derived in Appendix C, based on the observation that, since arrivals are uncorrelated, once a busy period of Q_1 or Q_2 starts, it evolves like in a legacy $M/G/1$ queue between two *vacations* [59], that is like in normal $M/G/1$ queues. With the above consideration, I can compute the average time to complete the initial busy periods for either Q_1 or Q_2. Assume I start with Q_1, then, if Q_2 is still not empty at the end of the busy period of Q_1, I switch the system observation point to Q_2. At that instant, I assume that Q_2 is busy with probability ρ_2. Using such probability represents an approximation with little impact on the performance of the model. More precisely, as illustrated in Appendix A, the model yields a lower bound of $E[\tau_A]$, although such lower bound is very close to the upper bound for $E[\tau_A]$ and hence to its correct value. Then I keep switching observation point until both queues are empty, which triggers a transition to state S. The resulting expressions for the time spent in state A when there are no arrivals in state S is as follows:

$$\begin{aligned} E[\tau_A|_{W\to A}] = \frac{1}{\lambda_1+\lambda_2} \Bigg\{ & \lambda_1 E\left[B_c^{(1)}\right] + \lambda_2 E\left[B_c^{(2)}\right] \\ & + \frac{\rho_1(\lambda_1\rho_2+\lambda_2)E\left[B^{(1)}\right]}{1-\rho_1\rho_2} + \frac{\rho_2(\lambda_2\rho_1+\lambda_1)E\left[B^{(2)}\right]}{1-\rho_1\rho_2} \Bigg\}; \end{aligned} \tag{6.37}$$

where I have used the following definitions, for $i \in \{1, 2\}$:

$$E\left[B_c^{(i)}\right] \triangleq \frac{1+\lambda_i(E[\tau_C|_{W\to A}, Q_i] + T_w)}{\mu_i - \lambda_i}; \tag{6.38}$$

$$E\left[B^{(i)}\right] \triangleq \rho_i \frac{2-\rho_i}{2\lambda_i(1-\rho_i)^2}. \tag{6.39}$$

Similarly, when there are arrivals in state S, the expression for the time spent in state A is:

$$E[\tau_A|_{S\to A}]=\frac{1}{\lambda_1+\lambda_2}\left\{\lambda_1 E\left[B_s^{(1)}\right]+\lambda_2 E\left[B_s^{(2)}\right]\right.$$
$$\left.+\frac{\rho_1(\lambda_1\rho_2+\lambda_2)E\left[B^{(1)}\right]}{1-\rho_1\rho_2}+\frac{\rho_2(\lambda_2\rho_1+\lambda_1)E\left[B^{(2)}\right]}{1-\rho_1\rho_2}\right\}, \tag{6.40}$$

where $E[B_s^{(i)}]\triangleq(\mu_i-\lambda_i)^{-1},\ i\in\{1,2\}$. Since with probability P_S there is no arrival during T_s, the average state duration is:

$$E[\tau_A]=P_S E[\tau_A|_{W\to A}]+(1-P_S)E[\tau_A|_{\mathrm{S}\to\mathrm{A}}]$$
$$=\frac{1}{\lambda_1+\lambda_2}\left\{\lambda_1 E[B_1^{(1)}]+\lambda_2 E[B_1^{(2)}]\right.$$
$$\left.+\frac{\rho_1(\lambda_1\rho_2+\lambda_2)E[B^{(1)}]}{1-\rho_1\rho_2}+\frac{\rho_2(\lambda_2\rho_1+\lambda_1)E[B^{(2)}]}{1-\rho_1\rho_2}\right\}, \tag{6.41}$$

with $E[B_1^{(i)}]\triangleq E[B_c^{(i)}]e^{-(\lambda_1+\lambda_2)T_s}+E[B_s^{(i)}](1-e^{-(\lambda_1+\lambda_2)T_s})$.

6.2.2. State probabilities η_α

With the model derived in this chapter, the mean cycle duration $E[T_{cycle}]$ is the sum of the above described cycle components, i.e.:

$$E[T_{cycle}]=E[\tau_S]+E[\tau_L]+E[\tau_C]+E[\tau_W]+E[\tau_A], \tag{6.42}$$

while the coefficients η_α for $\alpha\in\{A,L,C,S,W\}$ are given by $\eta_\alpha=E[\tau_\alpha]/E[T_{cycle}]$.

6.2.3. Mean waiting time per packet $D_\alpha^{(i)}$

The mean waiting time for a packet in Q_i is given by Eq. 6.28 also for the model presented in this section.

Next, I identify the mean waiting time of a packet in the queue depending on the state α at arrival time according to this more accurate model presented in this section. Note that the resulting delays are different in the two link directions.

Waiting time $D_A^{(i)}$ for packets arriving in state A. If a packet arrives in state A, I can use results for $M/G/1$ queues. Therefore, the average waiting time can be simply computed by means of the P-K formula [54]:

$$D_A^{(i)}=\frac{\lambda_i E[\sigma_i^2]}{2(1-\rho_i)},\quad i\in\{1,2\}. \tag{6.43}$$

The statistics of σ_i depend on the packet size distribution. For the sake of simplicity, here I assume that packet size is constant and its service time equals to $1/\mu_i$, so that I use $E[\sigma_i^2] = 1/\mu_i^2$, which yields:

$$D_A^{(i)} = \frac{\rho_i}{2\mu_i(1-\rho_i)}, \quad i \in \{1,2\}. \tag{6.44}$$

Waiting time $D_S^{(i)}$ for packets arriving in state S. As it has been described in § 5.1, packets that arrive in state S will immediately cause a transition to state A and the packets will be served with no further delay and thus $D_S^{(i)}$ equals zero:

$$D_S^{(i)} = 0, \quad i \in \{1,2\}. \tag{6.45}$$

Waiting time $D_L^{(i)}$ for packets arriving in state L. In state L only one packet will arrive in either of the two link directions, which causes the transition to state C. The waiting time of the packet received in state L is the sum of the time that the link remains in both state C and state W given by Eq. (6.35) and Eq. (6.36), respectively:

$$D_L^{(i)} = E[\tau_C|_{W\to A}, Q_i] + T_w, \quad i \in \{1,2\}. \tag{6.46}$$

Waiting time $D_C^{(i)}$ for packets arriving in state C. When a packet arrives in state C, it suffers from (i) the residual coalescing interval, (ii) the constant wake-up interval T_w, and (iii) the time to process and transmit packets already present in the queue at the arrival epoch of the new packet.

Poisson arrivals have uniform distribution over time. Therefore, the residual coalescing interval is given as $E[\tau_C]/2$. Similarly, the mean number of packets in the queue depends on the link direction i in which the packet arrival triggered the transition from state L to state C (which is 1 with probability $\frac{\lambda_i}{\lambda_1+\lambda_2}$ and 0 otherwise) and on the average number of packets which arrived afterward, i.e., $\lambda_i E[\tau_C]/2$. The average service time for those packets is therefore $\frac{1}{\mu_i}\left(\frac{\lambda_i E[\tau_C]}{2} + \frac{\lambda_i}{\lambda_1+\lambda_2}\right)$. Consequently, the mean waiting time $D_C^{(i)}$ for packets arriving in state C is:

$$D_C^{(i)} = T_w + \frac{E[\tau_C]}{2} + \rho_i\left(\frac{E[\tau_C]}{2} + \frac{1}{\lambda_1+\lambda_2}\right), i \in \{1,2\}. \tag{6.47}$$

Waiting time $D_W^{(i)}$ for packets arriving in state W. For a packet that arrives when the link is in state W the mean delay depends on two main parameters: the mean residual wake-up time and the time to serve previously received packets (during states L, C and W). Since I assume Poisson arrivals the mean residual wake-up time is $T_w/2$. The time to serve previously received packets is $\left(\frac{\lambda_i}{\lambda_1+\lambda_2} + \lambda_i\left(E[\tau_C] + \frac{T_w}{2}\right)\right)/\mu_1$. So, the mean waiting time $D_W^{(i)}$ for packets received in state W is given by:

$$D_W^{(i)} = \frac{T_w}{2} + \rho_i\left(\frac{1}{\lambda_1+\lambda_2} + E[\tau_C] + \frac{T_w}{2}\right), i \in \{1,2\}. \tag{6.48}$$

Mean number of packets $n_\alpha^{(i)}$ for $\alpha \in \{A, S, L, C, W\}$. To identify the mean waiting time of the packets, I need to model the amount of packets that the link receives in any of the states. Since the arrival rate is different in the two link directions the mean number of packets will be different for the two link directions.

Let me first see the total amount of packets that the link receives over a cycle and then I can calculate the packets received in the separate states. Since arrivals are Poisson, the number of received packets over a cycle in each link direction is simply given by:

$$n_{cycle}^{(i)} = \lambda_i E[T_{cycle}], \quad i \in \{1, 2\}. \tag{6.49}$$

In state A, both link queues behave as independent $M/G/1$ systems with vacations [59] in a generic interval between two vacations.[1] Therefore, the EEE link will receive $\lambda_i E[\tau_A]$ packets in each of the two link directions $i = 1, 2$, i.e.:

$$\begin{aligned} n_A^{(i)} = \lambda_i E[\tau_A] = \frac{\lambda_i}{\lambda_1 + \lambda_2} \Bigg\{ & \lambda_1 E[B_0^{(1)}] + \lambda_2 E[B_0^{(2)}] \\ & + \frac{\rho_1(\lambda_1\rho_2 + \lambda_2)E[B^{(1)}]}{1 - \rho_1\rho_2} + \frac{\rho_2(\lambda_2\rho_1 + \lambda_1)E[B^{(2)}]}{1 - \rho_1\rho_2} \Bigg\}. \end{aligned} \tag{6.50}$$

While the EEE link is in state S there will be at most one arrival in only one link direction $i = 1, 2$, i.e.:

$$\begin{aligned} n_S^{(i)} &= Pr(\text{arrival in link direction } i) \cdot (1 - P_S) \cdot 1 \\ &= \frac{\lambda_i}{\lambda_1 + \lambda_2} \left(1 - e^{-(\lambda_1+\lambda_2)T_s}\right) = \lambda_i E[\tau_S]. \end{aligned} \tag{6.51}$$

Similarly, while the EEE link is in state L it will receive exactly one packet in only one link direction, i.e.:

$$\begin{aligned} n_L^{(i)} &= Pr(\text{arrival in link direction } i) \cdot P_S \cdot 1 \\ &= \frac{\lambda_i}{\lambda_1 + \lambda_2} e^{-(\lambda_1+\lambda_2)T_s} = \lambda_i E[\tau_L], \ i \in \{1, 2\}. \end{aligned} \tag{6.52}$$

In state W, since τ_W is either 0 or T_w, the EEE link will receive either 0 or $\lambda_i T_w$ Poisson arrivals in each of the two link directions, i.e.:

$$n_W^{(i)} = \lambda_i T_w e^{-(\lambda_1+\lambda_2)T_s} = \lambda_i E[\tau_W], \quad i \in \{1, 2\}. \tag{6.53}$$

Eventually, the remaining packets arrive in state C, so that, as it is from Eqs. (6.42) and (6.49)–

[1] Note that vacations are correlated (i.e., they have to coincide), whereas arrivals and service processes are independent. So, between any two vacations, the two link queues behave as normal and independent $M/G/1$ queues.

(6.53) I have, for $i = 1, 2$:

$$n_C^{(i)} = n_{cycle}^{(i)} - n_S^{(i)} - n_A^{(i)} - n_L^{(i)} - n_W^{(i)} = \lambda_i E[\tau_C]. \tag{6.54}$$

As a *corollary*, note that the computation of n_α reveals that the average number of arrivals in each system state behaves like in the case of Poisson processes not only for the overall cycle, but also for each interval between any two state transitions.

6.3. Sensitivity Analysis of EEE with Coalescing

The partial derivatives with respect to either T_c or N_c for both η_{LPI} and D_i show a dependence on the partial derivative of $E[\tau_C]$ as can be seen, e.g., from the analysis presented in § 6.1.4. Thus, next I report the partial derivative of $E[\tau_C]$ (and $E[T_{cycle}]$), the rest is mere calculation.

6.3.1. Partial derivatives with respect to T_c

The partial derivative of $E[\tau_C]$ with respect to T_c is

$$\frac{\partial E[\tau_C]}{\partial T_c} = \sum_{k=0}^{N_c-2} \sum_{j=0}^{N_c-2} \frac{\lambda_1^k \lambda_2^j}{k!j!} T_c^{k+j} e^{-(\lambda_1+\lambda_2)T_c} > 0, \quad \forall T_c > 0; \tag{6.55}$$

and the partial derivative of $E[T_{cycle}]$ with respect to T_c is given by the following expression:

$$\begin{aligned}\frac{\partial E[T_{cycle}]}{\partial T_c} &= \frac{\partial}{\partial T_c}\left\{E[\tau_C]\left[1 + \frac{1}{\lambda_1+\lambda_2}\left(\frac{\lambda_1\rho_1}{1-\rho_1} + \frac{\lambda_2\rho_2}{1-\rho_2}\right)\right]\right\} \\ &= \left[1 + \frac{1}{\lambda_1+\lambda_2}\left(\frac{\lambda_1\rho_1}{1-\rho_1} + \frac{\lambda_2\rho_2}{1-\rho_2}\right)\right] \frac{\partial E[\tau_C]}{\partial T_c} \qquad (6.56) \\ &= b\,\frac{\partial E[\tau_C]}{\partial T_c} > 0, \quad \forall T_c > 0. \qquad (6.57)\end{aligned}$$

Finally, I get the partial derivative of the energy saving η_{LPI} with respect to T_c as follows:

$$\begin{aligned}\frac{\partial \eta_{LPI}}{\partial T_c} &= \frac{\frac{\partial E[\tau_C]}{\partial T_c} E[T_{cycle}] - \frac{\partial E[T_{cycle}]}{\partial T_c}\left(\frac{1}{\lambda_1+\lambda_2} + E[\tau_C]\right)}{E^2[T_{cycle}]} \\ &= \frac{a - \frac{b}{\lambda_1+\lambda_2}}{(a + b\,E[\tau_C])^2} \frac{\partial E[\tau_C]}{\partial T_c}. \qquad (6.58)\end{aligned}$$

From the above expressions, it is clear that the energy saving is a monotonic function of T_c, and moreover $\frac{\partial \eta_{LPI}}{\partial T_c} > 0, \forall T_c > 0$. Therefore the delay monotonically increases with T_c.

Table 6.1: Maximum η_{LPI} for $D_{target} \leq 1\ ms$ (ρ_1, ρ_2,λ_1, λ_2 are taken from real traffic traces)

ρ_1 [%]	ρ_2 [%]	λ_1 [pkts/s]	λ_2 [pkts/s]	$Max\{\eta'_{LPI}\}$[%]	T'_c [ms]	N'_c [pkts]	$Max\{\eta''_{LPI}\}$[%]	T''_c [ms]	N''_c [pkts]
0.11	5.25	2186	4343	82.09	≥ 3	≥ 32	82.09	=2	100
10.54	0.66	10410	5324	62.84	≥ 7	≥ 22	60.02	=2	100
0.57	32.68	10051	27459	15.34	≥ 9	$= 205$	8.74	≥ 5	100
1.01	40.52	17091	34042	1.80	≥ 10	$= 255$	0.75	≥ 4	100
5.06	0.5	5409	3809	77.50	≥ 5	≥ 15	66.55	$= 1$	100
1.14	17.93	9639	17320	37.59	≥ 7	≥ 75	31.17	$= 3$	100
0.20	0.06	310	268	92.72	$= 1$	≥ 15	92.72	$= 1$	100

Similarly, the partial derivative of the delay D_i with respect to T_c is:

$$\begin{aligned}\frac{\partial D_i}{\partial T_c} =& \frac{\left(\frac{\rho_i}{2\mu_i(1-\rho_i)} - D_i\right)\frac{\partial E[T_{cycle}]}{\partial T_c} - \frac{\rho_i}{2\mu_i(1-\rho_i)}\frac{\partial E[\tau_C]}{\partial T_c}}{E[T_{cycle}]} \\ &+ \frac{\left(\frac{1}{\lambda_1+\lambda_2} + T_w + E[\tau_C]\right)(1+\rho_i)\frac{\partial E[\tau_C]}{\partial T_c}}{E[T_{cycle}]}. \end{aligned} \tag{6.59}$$

Also in this case it is possible to show that $\frac{\partial D_i}{\partial T_c} > 0, \forall T_c > 0$ as far as loads are not extremely high. In practice, high loads prevent any significant EEE benefit [22, 29, 82], and therefore I can safely assume that the delay monotonically increases with T_c under the circumstances in which energy saving can be achieved.

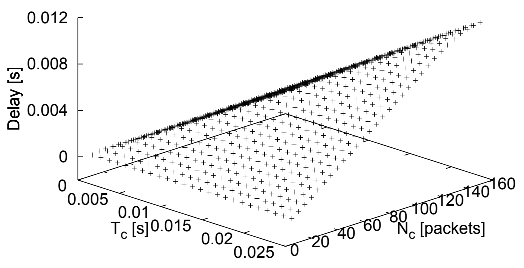
6.3.2. Partial derivative with respect to N_c

Regarding the partial derivative of $E[\tau_C]$ with respect to N_c, since N_c takes only integer values (it refers to packets) I consider the forward difference between $E[\tau_C]$ computed at $N_c + 1$ and at N_c:

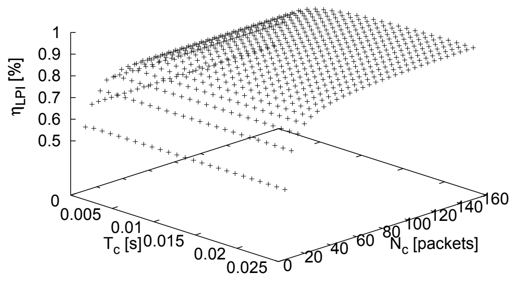
$$\begin{aligned}\frac{\partial E[\tau_C]}{\partial N_c} \approx& \Delta_{N_c}[E[\tau_C]](N_c) = \frac{E[\tau_C](N_c+1) - E[\tau_C](N_c)}{1} \\ =& \sum_{j=0}^{N_c-2} g_{\lambda_1\lambda_2}(N_c-1, j) + \sum_{k=0}^{N_c-2} g_{\lambda_1\lambda_2}(k, N_c-1) \\ &+ g_{\lambda_1\lambda_2}(N_c-1, N_c-1) > 0, \quad \forall N_c \geq 2; \end{aligned} \tag{6.60}$$

where $g_{\lambda_1\lambda_2}(k,j) = \frac{\lambda_1^k\lambda_2^j}{k!j!}\int_{t=0}^{T_c} t^{k+j}e^{-(\lambda_1+\lambda_2)t} > 0, \ \forall T_c > 0.$

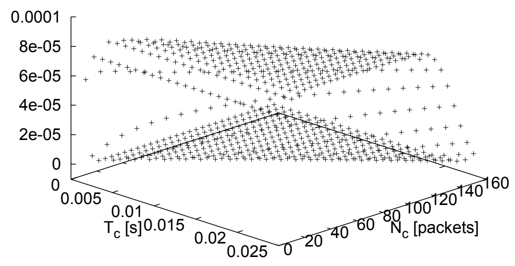
With the above, the partial derivatives of $E[T_{cycle}]$, η_{LPI}, and D_i with respect to N_c have the same form as their partial derivatives with respect to T_c. Therefore, I can conclude that energy saving and delay grow monotonically with N_c as well.



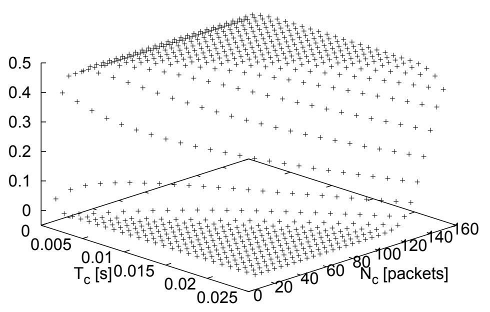
(a) D_2.



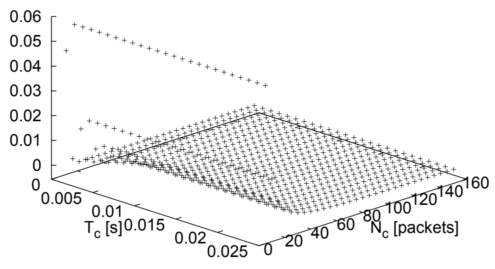
(b) η_{LPI}.



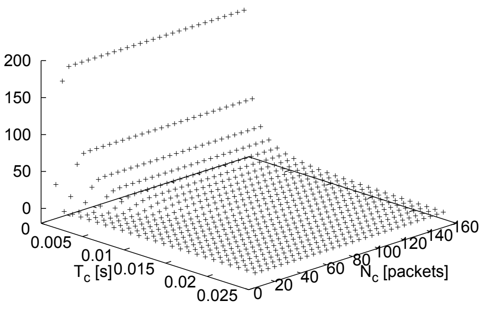
(c) $\frac{\partial D_2}{\partial N_c}$.



(d) $\frac{\partial D_2}{\partial T_c}$.



(e) $\frac{\partial \eta_{LPI}}{\partial N_c}$.



(f) $\frac{\partial \eta_{LPI}}{\partial T_c}$.

Figure 6.4: Coalescing delay, energy saving, and their partial derivatives with respect to T_c and N_c. Since the delay due to coalescing is higher in the least loaded link direction, I show only the delay for packets transmitted in that direction (D_2).

6.3.3. Analysis

The partial derivatives with respect to either T_c or N_c show the strong dependency of D_i and η_{LPI} on $E[\tau_C]$ (and on $E[T_{cycle}]$ but this also depends on $E[\tau_C]$). Furthermore, the value of $E[\tau_C]$ grows with T_c and N_c, and I have shown that both η_{LPI} and D_i monotonically grow with

T_c and N_c.

To graphically see the impact of T_c and N_c on the delay, D_i, and the energy saving, η_{LPI}, I plot in Figure 6.4 an example of partial derivatives, representing the behavior of η_{LPI} and D_i for different T_c and N_c values when the offered load is $\rho_1 = 5.06\%$ and $\rho_2 = 0.5\%$. These loads correspond to a load profile of a traffic trace I collected on a gigabit link in InterHost. Moreover, this is a representative link load since, according to [15], about 80% of the links operate with less than 10% of load, so that the selected case represents a medium load case.

Specifically, Figure 6.4(a) illustrates the behavior of the delay experienced in the most loaded link direction (which is the highest of the two average delays). The figure shows that the delay quickly grows to unacceptable values with both T_c and N_c. The energy saving η_{LPI} also grows, but it does it faster with small values of T_c and N_c, and afterwards it saturates. Overall, the impact of T_c and N_c seems similar. However, the study of the partial derivatives presented in Figure 6.4 unveils that both delay and energy saving are more sensitive to changes in T_c rather than in N_c. Indeed, Figures 6.4(c), 6.4(d), 6.4(e), and 6.4(f) point out that the partial derivatives with respect to T_c are up to three orders of magnitude higher than the ones with respect to N_c. I have observed the same behavior for a large range of load combinations. Therefore, I can say that T_c is more important than N_c in the control of delay and energy saving in EEE. Another important observation is that the impact of N_c saturates for relatively small values of the coalescing buffer size, i.e., implementing buffer sizes of 100 packets allows to achieve the highest possible energy saving.

To validate the above observations, I report in Table 6.1 a few representative study cases corresponding to different combinations of average loads ρ_1 and ρ_2 as observed in real traffic traces for the two link directions. In the table, for each case, I report the maximum energy saving that can be achieved by manually searching the optimal T_c and N_c subject to an average delay below 1 ms (I denote with η'_{LPI} the energy saving factor that can be achieved subject to a given delay contraint). Additionally, I report the values T'_c and N'_c at which η'_{LPI} is maximized. Moreover, for the case without delay constraints but still the delay is below 1 ms, I fix the value of N_c to N''_c =100 packets (larger values do not improve the energy saving gain) and I check again the maximum value of the energy saving factor, which I denote as η''_{LPI}, achievable by varying T_c only. In the table, I report the value T''_c of the coalescing timer which maximizes the energy saving.

From Table 6.1, I can observe that energy saving in the two cases is not far, so that I can think of fixing the size of the coalescing buffer and using an adaptive coalescing algorithm that, by adjusting the sole coalescing timer T_c, is able to achieve near optimal energy savings while keeping bounded the average delay of the packets due to coalescing. Noticeably, Table 6.1 also shows that small values of T_c are needed to achieve optimal (or near-optimal) performance figures, so that the optimal value of T_c can be searched in a small range. In Chapter 7, I use the results of this chapter to design two families of dynamic coalescing control mechanisms: a low complexity family of mechanisms based on coalescing timeouts and buffer fill up events, and another family based on run-time delay measurements. In Chapter 8 I will evaluate the performance of EEE systems with real traffic traces and validate the models against packet-level simulations.

6.4. Summary

In this chapter, I presented an analytical model for $1\ Gbps$ bidirectional EEE links with coalescing. The model is able to estimate both the energy saving and the mean waiting time of the packets in each link direction, due to coalescing, compared to legacy Ethernet links. The model is unique in the literature since only unidirectional EEE links with coalescing were considered so far. Moreover I have used sensitivity analysis to understand the impact of coalescing parameters, such as timer T_c and buffer size N_c, on the energy saving and the delay experienced over Energy Efficient Ethernet (EEE) links with coalescing. The analysis reveals that optimizing energy saving subject to delay constraints is possible by simply adapting T_c.

Chapter 7

Dynamic coalescing strategies

In this chapter I present two families of algorithms that can be used to dynamically match the coalescing parameters to the traffic conditions. These algorithms will allow me to explore the performance of dynamic packet coalescing, which has not been addressed so far in the literature. The first family of algorithms follows the NT policy paradigm and, as in legacy adaptive approaches, it adjusts the coalescing parameters based on the sole occurrence of timeouts and buffer fill-ups [51] (*NT coalescing cotrol* - NTCC). The suitability of the coalescing parameters is judged based on the estimate of average delay and not on the instantaneous variations of delay. In the second family of algorithms, I design and study a *measurement-based coalescing control* solution (MBCC) that tunes the coalescing parameters on-the-fly, according to the instantaneous load and the coalescing delay experienced by the packets.

The rationale behind proposing dynamic coalescing is that static configurations might incur in high delays when the traffic is low. For instance, to increase energy saving, coalescing techniques try to avoid short and frequent transmission bursts by means of large T_c and N_c values. However, as soon as the traffic intensity causes frequent coalescing timeouts, the latency often exceeds T_c. Therefore, in a static configuration, one cannot use very large values of T_c when the traffic intensity can be low with non-negligible probability. Similarly, large values of N_c increase the achievable energy saving, but cause high latency due to queue backlog to be served after the coalescing period. Using a static configuration for all traffic conditions might incur in the following problem: when the traffic is low the coalescing timeout expires frequently, while when the traffic is high one or both coalescing buffers fill up too quickly. In the former case, the experienced delay might be too high, while in the latter case, the achieved energy saving might be far from optimal.

In the following sections I present the two families of dynamic coalescing algorithms for EEE. In the first family, I propose Dynamic Timeout Algorithm which uses tunable coalescing timeouts of duration T_c and Dynamic Queue Size Algorithm which uses coalescing buffers of variable size N_c. In the second family, I design a simple measurement-based delay-controlled distributed adaptive coalescing scheme in which network cards at the edge of the link coordinate

Algorithm 1: Dynamic Timeout Algorithm.

```
Data: $\delta_{up}$ or $\gamma_{up}$, $\delta_{down}$ or $\gamma_{down}$, $T_c^{\min}$, $T_c^{\max}$, $N_c$
if Transition $C \to A$ then
    if Coalescing timeout expired then
        if $T_c < T_c^{\max}$ then
            increase $T_c$;
    else
        if $T_c > T_c^{\min}$ then
            decrease $T_c$;
    restart the coalescing timeout with new $T_c$;
```

by running a simple distributed algorithm to sense the delay incurred by packets. My proposal uses the sensed delay as control signal to trigger the dynamic adaptation of the coalescing timer in the direction identified through the sensitivity analysis derived in § 6.3.

7.1. NT-policy Coalescing Control

In this family of algorithms I use the expression "NT-policy" (see § 6.2 and therein) and therefore the name "NT-policy Coalescing Control" refers to this class of coalescing algorithms. The duration of the coalescing period is limited according to the following two cases: (i) a coalescing buffer fills up due to $N_c - 1$ arrivals in the link direction that triggered the transition $L \to C$ or N_c arrivals in the other direction (*N-event according to [96]*, see bottom of Figure 6.3); (ii) a timeout occurs: no more than $N_c - 2$ packets arrive in the link direction that triggered the transition $L \to C$ and no more than $N_c - 1$ packets arrive in the other direction (*T-event* according to [46], see top of Figure 6.3). In the following subsections I present two classes of algorithms, Dynamic Timeout Algorithm which dynamically tunes the coalescing timeouts T_c and Dynamic Queue Size Algorithm which dynamically tunes the coalescing buffers N_c.

7.1.1. Dynamic Timeout

In this class of algorithms, as indicated by its name, the adjustable parameter is the coalescing timeout T_c, while the coalescing buffers have fixed size N_c. Recall that T_c is defined as the maximum interval that EEE network cards can remain in state C after the arrival of the first packet in state L, thus extending the normal EEE energy saving interval. As stated before, when the coalescing timeout expires, both Ethernet interfaces start transmitting the queued packets to the other link edge. The goal of this first class of algorithms is to keep the system in state C for as long as it is needed to fill up at least one coalescing buffer, i.e., to adjust T_c so that the coalescing operation (i.e., the duration of state C) lasts $\sim T_c$ seconds and the maximum coalescing gain is achieved by filling up the coalescing buffer.

The algorithm's behavior is described by means of pseudocode in Algorithm 1. In this algorithm, the value of T_c keeps changing when the transition $C \to A$ occurs. If the transition occurs

Algorithm 2: Dynamic Queue Size Algorithm.

```
Data: $\delta_{up}$ or $\gamma_{up}$, $\delta_{down}$ or $\gamma_{down}$, $N_c^{\min}$, $N_c^{\max}$, $T_c$
if Transition $C \rightarrow A$ then
    if Coalescing timeout expired then
        if $N_c > N_c^{\min}$ then
            decrease $N_c$;
    else
        if $N_c < N_c^{\max}$ then
            increase $N_c$;
    restart the $T_c$;
```

because of a timeout expiration, then T_c is incremented, unless it reaches a maximum value $T_c^{\max}$. Its maximum value may depend on various factors but the most important is the maximum delay tolerance that is allowed either by applications, or by quality of service constraints. Similarly, if the transition $C \rightarrow A$ occurs because one of the coalescing buffers fills up, then T_c is decremented, unless it reaches its minimum allowable value $T_c^{\min}$. Therefore, the algorithm tries to adjust T_c in a way that state C lasts approximately T_c seconds while trying both to avoid unnecessary long timeouts and to fill up coalescing buffers.

In Algorithm 1, increments and decrements of T_c can follow different strategies. In particular, I consider that both increments and decrements can be either additive or multiplicative. Therefore, to fully specify the behavior of Algorithm 1, I need to specify (i) whether additive or multiplicative increments and decrements are used, (ii) the steps used in case of additive operation (δ_{up} for increments and/or δ_{down} for decrements), or the multiplicative factors used in case of multiplicative operation (γ_{up} for increments and/or γ_{down} for decrements), and (iii) the range $[T_c^{\min},\ T_c^{\max}]$ in which T_c can be adjusted.

The performance of Algorithm 1 depends on the target N_c value adopted. Throughout the experiments presented in Chapter 8, I use $T_c^{\min} = 0.1\ ms$ and $T_c^{\max} = 100\ ms$ which cover a wide range of delays acceptable in practical LAN deployments, while I test all combinations of additive and multiplicative increments and decrements, with various values for δ_{up}, δ_{down}, γ_{up}, and γ_{down}.

7.1.2. Dynamic Queue Size

The second algorithm is similar to the first, but it adjusts the coalescing buffer size N_c instead of the coalescing timeout T_c. In Algorithm 2, N_c is dynamically and automatically tuned in order to adapt the coalescing operation to achieve a target coalescing delay T_c. I.e., the target of Algorithm 2 is to keep the system in state C for about T_c seconds and during this interval accumulate as many packets as possible.

In Algorithm 2, when the traffic intensity is so low that the coalescing timeout expires before N_c packets are queued in any of the two coalescing buffers, the algorithm decreases N_c. The minimum value for N_c is $N_c^{\min} = 2$, since smaller values would not result in any coalescing operation. Conversely, when the traffic increases and at least one coalescing buffer fills up before the coalescing timeout expires, the algorithm increases N_c up to its maximum value $N_c^{\max}$. Similarly to what stated for Algorithm 1, increments and decrements can be either additive or multiplicative, with parameters that I keep calling δ_{up}, δ_{down}, γ_{up}, and γ_{down}, like in the previous algorithm.

The performance of Algorithm 2 depends on the target value of T_c fixed in the system which I keep in the same range as for Algorithm 1. In the experiments, I use $N_c^{\min} = 2$ and $N_c^{\max} = 10,000$, while I tested all combinations of additive and multiplicative increments and decrements, with various values for δ_{up}, δ_{down}, γ_{up}, and γ_{down}.

7.2. Measurement-based Coalescing Control

Differently from what described for NTCC, here I use analytical results on the sensitivity of D_i and η_{LPI} to make *run-time* educated decisions on how to adapt the coalescing parameters to meet a maximum target delay D_{target}.

The analysis tells that T_c and N_c behave qualitatively in a similar way. Specifically, fixing one of the two parameters limits the maximum achievable energy saving, although, by tuning the other parameter, it is possible to adjust the energy saving from zero to the maximum while increasing the delay monotonically. Therefore, to implement an adaptive coalescing algorithm, it is enough to fix one parameter between T_c and N_c to a sufficiently high value (which guarantees that near-maximal energy saving can be achieved), and adapt the remaining parameter.

The analysis also unveils that η_{LPI} and D_i values are more sensitive to T_c rather than toN_c. With the above consideration, jointly to the fact that N_c is limited to integer values, T_c results to be a better candidate for the fine tuning of energy and delay tradeoff when coalescing is adopted.

Therefore, I design an adaptive coalescing algorithm in which only T_c is adjusted. Moreover, in this algorithm, I implement a simple yet effective mechanism to detect when the coalescing is causing excessive delay and timely react. What I include in the algorithm is a low-pass filter to estimate the average coalescing delay D_i. When the link switches to state W, the dynamic timer algorithm tunes $T_c \in [T_c^{\min}, T_c^{\max}]$ based on the experienced (measured) average delay and D_{target}. The way T_c is adapted can folow additive or multiplicative laws, as for NTCC, but trigered by delay measurements rather than by an NT-policy. Thus, I obtain a family of Measurement Based Coalescing Control (MBCC) mechanisms. The pseudocode of an MBCC heuristic using additive increases and multiplicative decreases for T_c is reported in Algorithm 3.

The analysis says that increasing T_c increases both η_{LPI} and delay at any load, so when the average delay is below or above the target, the algorithm increments or decrements the T_c value, respectively. The advantages of my approach are twofold: (i) given that T_c is tuned after exiting state C, the delay adaptation procedure is almost immediate (a few milliseconds), which allows to instantly react to changes in packet delay; (ii) my adaptive algorithm adapts quickly to any changes in traffic load simply by estimating the packet delay. Load variations occur very often in the daily patterns and so my simple T_c adaptation mechanism can produce great benefit for EEE.

Algorithm 3: MBCC: Adaptive Coalescing Timer

```
Input: run-time average estimate of delays $D_1$ and $D_2$
while $C \to W$ do
    if $(D_1 \&\& D_2) \leq D_{target}$ then
        if $T_c < T_c^{\max}$ then
            $T_c = \max\{T_c + \delta, T_c^{\max}\}$
    else
        if $T_c > T_c^{\min}$ then
            $T_c = \max\{T_c - \delta, T_c^{\min}\}$ or $T_c = \max\{(1-\gamma)T_c, T_c^{\min}\}$
```

With the above, I have defined not one but an entire class of delay-controlled MBCC algorithms, which differ in the way the value of T_c is tuned. For example, additive or multiplicative increases and decreases can be used. For what concerns performance evaluation, in Chapter 8, I simply use either (i) an additive increase/decrease approach with fixed step δ or (ii) an additive increase/multiplicative decrease approach with fixed additive step δ and multiplicative decrease percentage γ. I only focus on those two schemes because, first, multiple increase schemes provide less fairness than additive increase schemes and second, multiple increase schemes wildly oscillate and are a source of instability, thus leading to poor performance [27, 51]. Note that, due to the high sensitivity of delay and energy saving with respect to variations of T_c, the possible values of δ and γ have to be small enough to cause small changes in the adaptive timer.

Note that, in gigabit EEE links, the two directions are correlated and therefore the algorithm has to run distributed over the two network cards at the edge of the link, although this requires only a few overhead messages to be transmitted from one card to the other to signal state transition events. However, such messages can be piggybacked by regular EEE state control messages, since each link edge just needs to send one bit to tell the other edge whether the measured delay is exceeding D_{target} or not.

7.3. Summary

In this chapter I presented two families of algorithms that dynamically tune the coalescing parameters to the traffic conditions. The first family of algorithms adapts its coalescing parameters based on the event of timeouts and buffer fill-ups (*NT coalescing control* - NTCC). The **average delay at the end of the process** will judge the effectiveness and suitability of the coalescing parameters. The second family of algorithms is based on the coalescing properties analytically studied: I have designed MBCC, a class of adaptive coalescing algorithms which continuously adapts T_c according to the **delay sensed by the link**.

Chapter 8

Performance evaluation with real traffic traces of EEE with coalescing

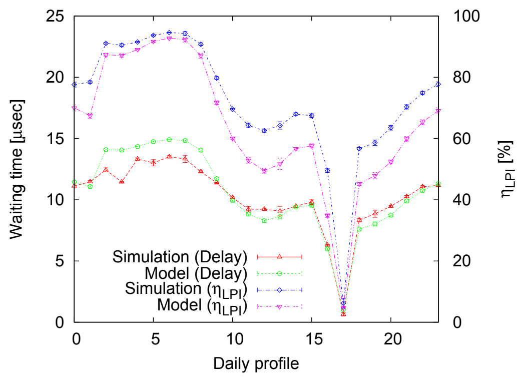
In this chapter I begin with § 8.1-8.3 where I use ns-3 [1] simulations and the models described in § 6.1 and § 6.2 to assess the performance of coalescing techniques over 1 $Gbps$ EEE links. Note that the model cannot be used for dynamic coalescing, so that I will use simulations for the performance assessment of the *Dynamic Timeout* and the *Dynamic Queue Size* algorithms. However, I will show that static coalescing performs almost as well as NTCC algorithms.

In § 8.4 I investigate on the power saving and on the delay of the packets by using different δ_{up}, δ_{down} and γ_{up}, γ_{down} parameters for MBCC algorithms and for various values of N_c and T_c. In the model validation I do not address the properties of MBCC but I try to understand the limits on energy saving and delay by applying dynamic strategies. In fact, we will observe for MBCC algorithm the trade-off between η_{LPI} and delay while adjusting fast or slowly the corresponding tunable parameter (higher or lower values for γ, δ).

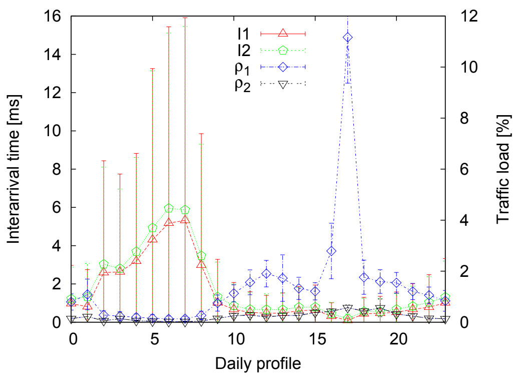
In § 8.5 I present a simple economical analysis using a typical large data center in order to show the imediate economical benefits, achievable only by replacing the legacy Ethernet links with EEE links.

8.1. EEE without Coalescing

I first consider a plain EEE scenario, with no coalescing, for a typical daily profile. In Figure 8.1(a), I depict η_{LPI} and the average packet delay computed via the modified ns-3 simulator and via the analytical model which I presented in § 6.1. Small error bars show the deviation of η_{LPI} and of the average delay since the offered traffic is based on real traffic traces and therefore it is not constant over the sample. Figure 8.1(b) illustrates the measured traffic (ρ_i) sampled once every 60 minutes in each of the directions for a typical day with the corresponding standard deviation for each sample. Moreover, for the same traffic I show the burstiness of the packets as indicated by the interrival time (I_i) and its standard deviation. Although not explicitly reported in



(a) Average delay in the most delayed link direction and energy saving opportunities (η_{LPI}).



(b) Load and interarrival time.

Figure 8.1: EEE model and simulation results with real traces, sampling a one-day traffic pattern (without coalescing).

the figure, when traffic is very low, in the traces I have measured slightly higher burstiness with respect to ideal Poisson interarrivals. This is due to the fact that with low load only a few and bursty TCP flows populate the link.

For those figures I have used the traces of February 8th.The load is very low most of the time, i.e., less than 1%, but I observed an exceptional peak value of medium traffic ($\sim 11\%$) which shows the energy degradation for this load. This traffic behavior is in line with typical data center loads as described in [15].

We can observe that the model I previously presented estimates the energy saving with a good accuracy in either low ($0-1\%$) or medium ($\sim 11\%$) traffic conditions. Indeed we can see that the model clearly captures the trend of energy saving when the traffic load increases/decreases, even though a few discrepancies appear because the model underestimates the effect of burstiness. Neglecting the burstiness of the packets leads the model to give a "pessimistic" estimation about η_{LPI}. Moreover, in Figure 8.1(a), EEE can save most of the time more than 50% of energy. In extreme cases (but not infrequently, since they constitute about 1/3 of the day's samples) the time spent in energy saving, η_{LPI}, exceeds 90%. Remarkably, EEE enables substantial energy saving ($\eta_{LPI}>90\%$) when the traffic is limited to few percents of the link capacity. However, EEE does not appear to be able to save much energy when the traffic surges to relatively low peaks, e.g., to 10% of the link capacity. With higher loads, I can safely assume that a plain EEE approach would bring negligible energy saving. This justifies the research for EEE enhancements that would allow significant savings even when traffic reaches typical utilization levels such as 11% of the link capacity.

In Figure 8.1(a) I also evaluate the accuracy of the model, in terms of delay performance over the same traffic traces studied in Figure 8.1(b). The figure reports the delay suffered in the link direction over which the delay due to EEE state transitions is higher. Results achieved with model and simulation differ by at most 2 μs, which is a negligible quantity. The figure also shows that packet delay is higher when the traffic offered to the link is lower. Moreover, delays due to

Table 8.1: η_{LPI} and average delays computed with the model and via simulation

ρ_1	ρ_2	$E[S_p^{(1)}]$	$E[S_p^{(2)}]$	λ_1	λ_2	T_c	N_c	η_{LPI} [%]			$D^{(1)}$ [ms]			$D^{(2)}$ [ms]		
[%]	[%]	[bytes]	[bytes]	[pkts/s]	[pkts/s]	[ms]	[pkts]	simul.	model-1	model-2	simul.	model-1	model-2	simul.	model-1	model-2
						5	50	97.45	96.84	96.84	3.157	3.067	3.060	3.383	3.063	3.064
						5	100	97.45	96.84	96.84	3.157	3.067	3.060	3.383	3.062	3.064
0.20	0.06	802	281	310	268	10	50	98.29	98.11	98.11	5.925	5.657	5.647	6.140	5.652	5.655
						10	100	98.30	98.11	98.11	5.942	5.660	5.647	6.147	5.652	5.655
						20	100	98.92	98.91	98.91	11.130	10.725	10.700	11.306	10.710	10.714
						5	50	1.69	0.85	0.88	0.002	0.001	0.007	0.002	0.011	0.011
						5	100	1.69	1.74	1.75	0.001	0.026	0.027	0.001	0.039	0.039
0.71	39.69	67	1512	13187	32815	10	50	1.72	0.88	0.89	0.003	0.006	0.007	0.004	0.011	0.011
						10	100	1.79	1.74	1.75	0.005	0.027	0.027	0.005	0.039	0.039
						20	100	1.85	1.75	1.75	0.007	0.027	0.027	0.009	0.039	0.039
						5	50	7.87	4.56	4.59	0.025	0.046	0.046	0.020	0.060	0.061
						5	100	7.97	8.64	8.66	0.028	0.174	0.175	0.023	0.225	0.226
0.87	29.64	83	1477	13048	25084	10	50	7.93	4.57	4.59	0.041	0.046	0.047	0.029	0.060	0.061
						10	100	8.71	8.65	8.67	0.068	0.174	0.175	0.058	0.225	0.226
						20	100	9.20	8.66	8.76	0.101	0.175	0.165	0.082	0.225	0.226
						5	50	0.10	0.03	0.03	$< 1\mu$s	$< 1\mu$s	$< 1\mu$s	0.001	0.004	0.004
						5	100	0.14	0.06	0.06	0.001	$< 1\mu$s	0.001	0.003	0.005	0.005
0.98	53.66	69	1500	17769	44719	10	50	0.10	0.03	0.04	$< 1\mu$s	$< 1\mu$s	$< 1\mu$s	0.001	0.004	0.004
						10	100	0.14	0.06	0.06	0.001	$< 1\mu$s	0.001	0.003	0.005	0.005
						20	100	0.14	0.06	0.06	0.001	$< 1\mu$s	0.001	0.003	0.005	0.005
						5	50	85.03	90.90	90.89	1.896	2.442	2.361	2.123	2.363	2.440
						5	100	85.29	90.90	90.89	1.954	2.442	2.361	2.153	2.364	2.440
3.72	0.37	1180	165	3938	2778	10	50	88.26	94.08	94.08	3.971	4.946	4.944	3.819	4.786	4.784
						10	100	90.10	94.10	94.09	4.331	4.972	4.806	4.453	4.811	4.967
						20	100	91.90	95.82	94.82	8.926	9.031	9.022	8.256	9.707	9.698
						5	50	78.76	88.10	88.09	1.727	2.380	2.377	1.869	2.277	2.274
						5	100	79.21	88.10	88.09	1.815	2.380	2.274	1.922	2.277	2.377
5.06	0.50	1170	165	5408	3809	10	50	82.05	91.63	91.67	3.854	4.334	4.172	3.160	4.146	3.360
						10	100	85.52	92.18	92.18	4.079	4.914	4.696	4.109	4.701	4.909
						20	100	87.54	94.17	94.18	7.700	8.029	9.060	6.910	8.642	8.668
						5	50	17.17	27.31	27.40	0.164	0.361	0.364	0.235	0.443	0.446
						5	100	26.50	39.80	39.81	0.510	0.988	0.988	0.711	1.212	1.212
0.40	23.11	66	1511	7517	19116	10	50	17.73	27. 32	27.40	0.181	0.361	0.364	0.254	0.444	0.446
						10	100	31.54	41.05	41.10	0.745	1.082	1.085	1.002	1.327	1.331
						20	100	31.47	41.07	41.10	0.744	1.084	1.085	1.001	1.329	1.331
						5	50	28.38	30.03	30.16	0.310	0.440	0.445	0.295	0.513	0.519
						5	100	29.90	41.77	41.75	0.398	1.066	1.065	0.433	1.243	1.242
1.14	17.93	148	1294	9639	17320	10	50	29.90	30.04	30.16	0.428	0.440	0.445	0.384	0.514	0.519
						10	100	35.38	44.77	44.84	0.819	1.310	1.316	0.818	1.528	1.535
						20	100	36.42	44.79	44.84	0.988	1.312	1.316	1.029	1.530	1.535

EEE are never relevant, since they are comparable to or shorter than the duration a single packet transmission time.

In general, I have shown that my model can be safely used to estimate the EEE performance in terms of both energy saving and delay when no coalescing is adopted. In the next section I will show the correctness of my model also for the case of EEE with static coalescing. Before that, in light of the traffic measurements and energy saving estimates, I enlighten the potential economical benefit of EEE.

8.2. EEE links with Static Coalescing

In contrast to other works, I am the first to consider packet coalescing for EEE 1000BASE-T links taking into account that a simple energy saving mechanism regulates both link directions in a coordinated way. In Table 8.1 I list some representative results achieved with model and simulation. Since Chapter 6 is based on two models I report the results computed using the model in § 6.1 as "model-1" and the results of § 6.2 as "model-2".

For each trace I report loads, arrival rates and average packet sizes for each link direction. I simulate the coalescing algorithm for multiple combinations of N_c and T_c, and report the achieved results for the fraction of energy saving, η_{LPI}, and for the mean packet waiting time in the two link directions, $D^{(i)}$. I also use the average traffic descriptors reported in Table 8.1 to evaluate η_{LPI}, and $D^{(i)}$ through my model.

We observe that there is high traffic in one link direction and low traffic in the other link direction in all traces. Note that high loads correspond to large packet sizes, which, in turn, correspond to the typical behavior of TCP traffic, i.e., large packets in one link direction and small acknowledgments in the other link direction. From Table 8.1, we can also observe that the model approximates very well both the time spent in Low Power Idle, η_{LPI}, and the average delay in the two link directions. η_{LPI} is estimated with $\pm 5\%$ deviation in most of the cases (90%), while average delay estimations are subject to an error which is of the order of 10% for high delay cases (less than half ms), and a few μs for the case of low delays (tens of μs or less). Moreover, observing the table, we can say that model-2 performs slightly better than model-1 in most of the cases, while model-2 yields more accurate results when the error of model-1 is large.

Overall, under any of the tested traffic conditions and coalescing configurations, the results achieved through the model are accurate enough with respect to simulations. However, running simulations requires much more time and computational resources than the model. Hence, my model results in a suitable tool for the quick evaluation of EEE potentials and coalescing effectiveness, under any traffic condition.

8.3. Static vs. NT-policy Coalescing Control

I now compare the performance of static and dynamic coalescing. I use simulation only, since the model was not designed to predict the behavior of dynamic coalescing schemes. However, I will show that static and dynamic coalescing achieve very similar results, so that developing a model for EEE with dynamic coalescing in which the tuning of the parameters is based on NT policy events (like timeouts and buffer fill-ups) is unnecessary.

8.3.1. Static Coalescing

Table 8.1 reveals that coalescing enables high energy saving opportunities in a variety of load conditions. In particular, even in presence of loads of the order of 10%, energy saving ranges between 60% and 80%. High values of N_c and T_c would even allow for relevant energy saving ($\sim 10\%$) under high traffic ($\sim 30\%$). The table also reveals that delays grow fast with N_c and T_c. However, under high load, the achieved average delay assumes values not greater than a few hundreds of μs. High delays can be suffered only when traffic is low and coalescing parameters are high.

Notably, using $N_c \in [50, 100]$ packets and $T_c = 10\,ms$ yields high energy saving under all reported cases, with quite limited and acceptable delays. Hence, static coalescing appears to be near-optimal under a wide range of traffic conditions.

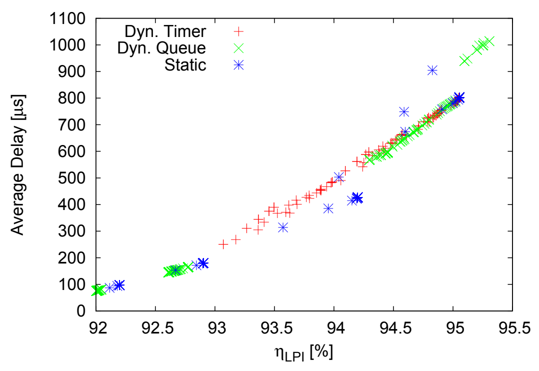
In addition to what reported in Table 8.1, I tested many possible combinations for many traffic traces, with N_c ranging from 2 to 10,000 packets, and T_c ranging from 0.1 to 100 ms. I omit here a complete description of the results with static coalescing and - since configurations resulting in high delay are undesirable - I limit the discussion only to cases with average delay below 1 ms. Specifically, I selected four different traffic traces, corresponding to the most typical traffic loads. Figure 8.2 reports the results for both static and dynamic algorithms under the following traffic conditions: Figure 8.2(a) is for $\rho_1 = 0.2\%$ and $\rho_2 = 0.06\%$, Figure 8.2(b) for $\rho_1 = 5.1\%$ and $\rho_2 = 0.1\%$, Figure 8.2(c) for $\rho_1 = 10.9\%$ and $\rho_2 = 0.2\%$ and Figure 8.2(d) for $\rho_1 = 39.7\%$ and $\rho_2 = 0.7\%$. Since we are interested in the potential performance of coalescing in terms of energy saving and delay, Figure 8.2 plots the values for the highest delay over the two link directions as a function of the achieved η_{LPI}. Each blue "$*$" marker in the figure is obtained under a different coalescing configuration for N_c and T_c.

Figures 8.2(a) to 8.2(d) show that higher energy saving corresponds to higher delay. However, delay can be minimized in exchange of small energy saving reduction. Under low loads, as in Figure 8.2(a), the value of η_{LPI} stays well above 90% in all cases, which means that coalescing of a very few packets (e.g., $N_c = 2$), combined with a short coalescing timeout (e.g., $T_c = 1\ ms$), is more than enough to achieve high energy saving with very limited packet delay. Conversely, under high load (e.g., Figure 8.2(d)), there is no static configuration that can bring high energy saving with bounded delay ($\eta_{LPI} < 1.52\%$).

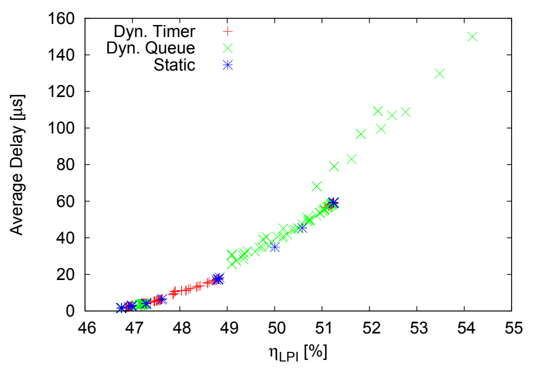
8.3.2. Dynamic Timeout

The Dynamic Timeout Algorithm "helps" the EEE coalescer to adapt its T_c value to the traffic conditions. In Figure 8.2, in addition to the results for static coalescing, I plot the average delay (the maximum in the two link directions) versus η_{LPI} for $N_c \in [2, 10000]$, δ_{up} and δ_{down} from 0.1 to 10 ms, and γ_{up} and γ_{down} from 1 to 100%. Adopted traffic traces are the same as for the evaluation of static coalescing. Similarly to the static coalescing case, I report results for the cases in which the average added delay per packet is below 1 ms only.

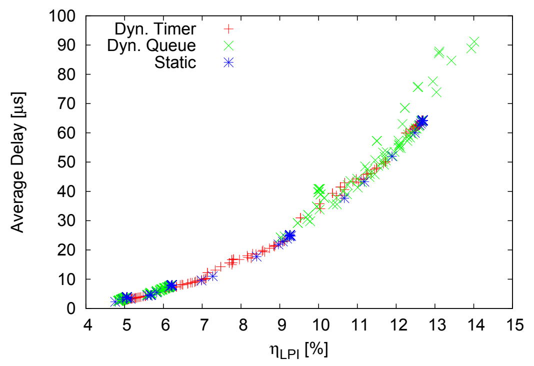
For all four traces used in Figure 8.2, we can see that Dynamic Timeout achieves almost the same results as for the static coalescing. Under low load conditions η_{LPI} is slightly higher than 90% while for loads equal to $\sim 5\%$, $\sim 10\%$ and $\sim 40\%$ η_{LPI} is $\sim 50\%$, $\sim 10\%$ and $\sim 1.5\%$, respectively. Within each figure, it is possible to observe that there are no huge differences in the



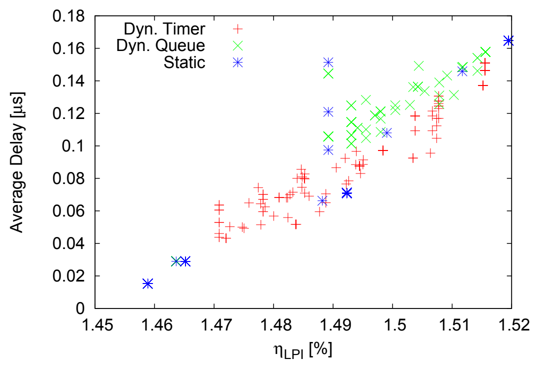
(a) $\rho_1 = 0.2\%$, $\rho_2 = 0.06\%$.



(b) $\rho_1 = 5.1\%$, $\rho_2 = 0.1\%$.



(c) $\rho_1 = 10.9\%$, $\rho_2 = 0.2\%$.



(d) $\rho_1 = 39.7\%$, $\rho_2 = 0.7\%$.

Figure 8.2: η_{LPI} vs. delay performance using different coalescing algorithms under various traffic conditions ($\delta \in [1, 10]$ packets or $\delta \in [0.1, 10]$ ms, $\gamma \in [1, 100]$ %, $N_c \in [2, 10000]$ packets, $T_c \in [0.1, 100]$ ms).

energy saving and delay performance achieved under different configurations for the same traffic trace analysis.

I conclude that dynamically adjusting the coalescing timeout does not outperform static coalescing when the coalescing delay has to be kept low under all traffic conditions.

8.3.3. Dynamic Queue Size

The Dynamic Queue Size algorithm adapts N_c to the traffic conditions and slightly outperforms the other tested algorithms in terms of energy saving. This behavior is shown in Figure 8.2, which also includes results for the Dynamic Queue Size algorithm, tested for $T_c \in [0.1, 100]\,ms$, additive parameter δ_{up} and δ_{down} from 1 to 10 packets, and multiplicative parameter γ_{up} and γ_{down} from 1 to 100%. Similarly to what I did for the other algorithms, I simulate EEE with coalescing using the Dynamic Queue Size algorithm over the same traffic traces as before. Each green "$\times$" marker in Figures 8.2(a) to 8.2(d) corresponds to a different configuration for the Dynamic Queue Size algorithm.

In terms of achievable energy saving, results for the Dynamic Queue Size case are, in general, slightly better than for static coalescing and for the Dynamic Timeout algorithm. Specifically, in Figure 8.2(a), under very low load, η_{LPI} improvements due to Dynamic Queue Size are relatively low. Instead, under medium/high loads (see Figures 8.2(b) and 8.2(c)) we observe an improvement of a few percents in η_{LPI}, with respect to the other algorithms. Finally, under very high loads (e.g., see Figure 8.2(d)), η_{LPI} is very small under any algorithm and configuration. In all cases, the (slightly) higher energy saving is achieved at the expenses of slightly higher delay.

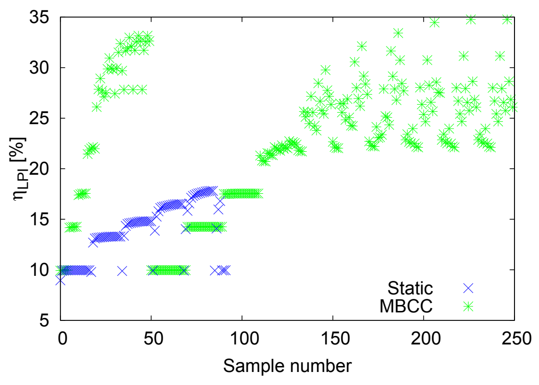
Summing up I can say that the differences in both energy saving and delay can be huge if I compare the different configurations in static or dynamic coalescing strategies. Nevertheless, if I compare the best configurations for static and dynamic coalescing I can safely infer that dynamically adjusting the coalescing parameters does not allow to achieve significantly better performance with respect to static coalescing. This result can be explained by recalling Eqs. (6.29) and (6.30), which reveal that the performance only depends on the average duration of the coalescing state. Therefore, the performance of any algorithm that attempts to boost the power saving gain is bounded by the achieved value of $E[\tau_C]$, which, in turn, is bounded by the maximum allowable average delay. Indeed, having shown that static coalescing and dynamic coalescing algorithms achieve similar energy saving and delay tradeoffs under a variety of traffic configurations, I conclude that static coalescers are preferable with respect to NTCC schemes for real implementation due to their low complexity. In fact, dynamic coalescing requires extra tuning (for γ and/or δ).

8.4. Measurement-based Coalescing Control

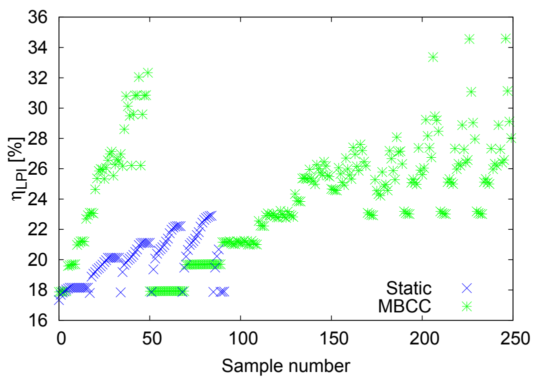
In this section I evaluate MBCC by implementing a delay-controlled adaptive coalescing timer algorithm. I benchmark MBCC against static coalescing algorithms, since as known in § 8.3, NTCC does not improve performance if a target delay has to be guaranteed [22]. In the following, I first evaluate the achievable energy savings obtained by different configurations of static coalescing and MBCC for a set of representative traffic traces. Afterwards, I illustrate the behavior of energy savings and delays over time, when the load keeps changing. For the experiments, I use the real traffic traces I have been allowed to collect in Satec, a large web hosting center in Madrid, Spain.

8.4.1. Experimental setup

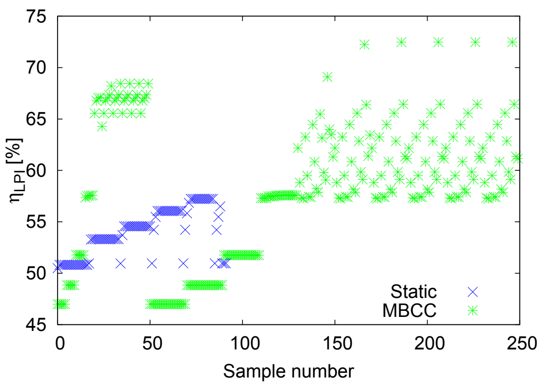
In this study, static coalescing schemes require the calibration of T_c and N_c based on the expected traffic characteristics or based on, e.g., the peak traffic. However, to guarantee low delay under low load conditions, both T_c and N_c have to be tuned to values well below the ones that achieve the best energy performance under medium or high traffic. In particular, since the criterion is to regulate the average coalescing delay of the packets crossing the EEE link under all traffic conditions, a (T_c, N_c) combination with small values has to be universally adopted to cope with the delay under scarce traffic conditions ($\sim$0.1% in the less loaded direction). Therefore, static coalescing has the disadvantage that it needs to be tuned on the *off-peak* traffic conditions.



(a) $\rho_1 = 1.14\%$, $\rho_2 = 18\%$.



(b) $\rho_1 = 0.57\%$, $\rho_2 = 32.7\%$.



(c) $\rho_1 = 0.11\%$, $\rho_2 = 5.3\%$.

Figure 8.3: Achievable energy saving for MBCC vs. Static coalescing with $D_{target} \leq 1\ ms$.

Apparently, so far, this has not been considered a great disadvantage for EEE links. In fact, energy savings are expected to be harvested only under low to medium traffic conditions. However, I argue that even though low loaded links represent about 40% of a data center links, there is still another 60% of the links from which additional energy savings could be potentially obtained.

To evaluate static coalescing, I test a range of values for T_c and N_c, as reported in Table 8.2, under different traffic conditions. In contrast, for MBCC with the delay-controlled adaptive timer heuristic (Algorithm 3), I consider a fixed N_c value, such as the one selected based on the results reported in Table 6.1 (i.e., we could select the value N_c =100 packets), whereas the T_c value is automatically adapted according to the traffic. I assign D_{target} as initial value for the timer T_c. Other configuration parameters for MBCC are the adaptation coefficients δ and γ. The range of values for N_c, δ and γ can be read in Table 8.3.

All tested parameters span over large intervals, to thoroughly explore their impact by means of simulations.

Table 8.2: Static Coalescing: list of T_c and N_c combinations

Parameter	Value
T_c [μs]	200/500/700/1000/1200/**1300**/1400/1500/1700/2000
N_c [packets]	2/5/**10**/11/13/15/17/20/25/30/40/50/60/70/80/90/100

Table 8.3: MBCC with Adaptive Coalescing Timer (Algorithm 3): list of parameters

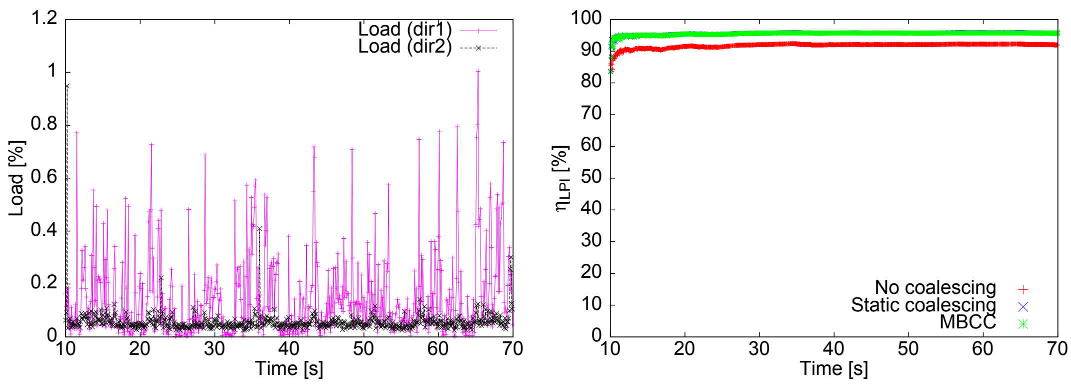
Parameter	Value
δ [μs]	10/30/100/300/**1000**
γ [%]	**10**/25/50/75
N_c [packets]	2/5/10/20/50/75/100/200/500/**1000**

8.4.2. Achievable energy saving

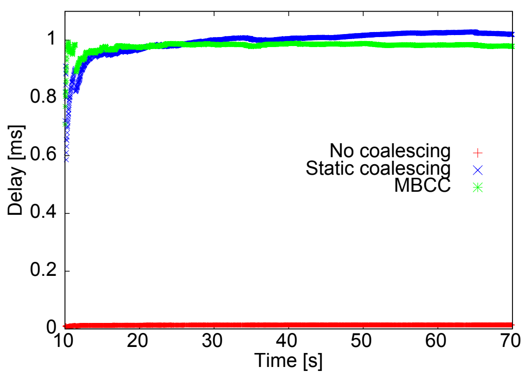
Here I compare static coalescing and MBCC under a variety of configuration choices, as reported in Tables 8.2 and 8.3. In particular, I report the results for energy saving subject to average coalescing delay, D_{target}, not exceeding 1 ms. I think that this is a reasonable upper bound for the average delay in a point-to-point link. Indeed, according to [28] a connection between East and West coast in the US has at least four hops that create 28.4 ms of average delay. Thus, I consider that adding 1 ms due to the use of EEE in a data center connected to such a network is acceptable.

In Figure 8.3 I plot η_{LPI} for three different load combinations (ρ_1, ρ_2). For static coalescing I run the simulation of a trace with a given combination (T_c, N_c) and I get the average delay of the packets in both directions and η_{LPI}. The delay can be higher or lower than D_{target} but I report the energy saving only for those combinations that give average coalescing delay below D_{target} (Table 8.2). For MBCC, since it guarantees that the delay is below D_{target}, I report all points corresponding to all the tested combinations of parameters (Table 8.3). Notably, the best results achieved with static coalescing in any of the depicted scenarios are very far from the best results of MBCC. Indeed, MBCC practically doubles the gain achieved by static coalescing.

Moreover, in the experiments I have observed that a particular combination performs best for static coalescing under any of the tested load combinations, i.e., (T_c =1300 μs, N_c =10 packets), reported in boldface in Table 8.2. In contrast, for the case of MBCC, I have observed high variability in the configuration that achieves the best results in the various cases. In particular, considering that in each subfigure of Figure 8.3 the first 50 samples for MBCC use only the parameter δ to adapt T_c (additive increase/decrease), while the remaining 200 samples use both δ and γ (additive increase, multiplicative decrease) for the adaptation of T_c, I can conclude that using both δ and γ is slightly more convenient. However, I have also observed that many configurations are equivalent, in particular when N_c is small (below 20), the performance is determined by N_c only, and changing δ and γ does not affect the results. In contrast, with higher N_c values δ and γ can be responsible for a fluctuation of 10-15% of energy saving. More in general, the results indicate that bigger values of δ and N_c allow bigger energy saving. Instead, a bigger value of γ reduces the energy benefit. The topmost points in all the cases correspond to the combination



(a) Load. (b) Energy saving (Static and MBCC overlap).

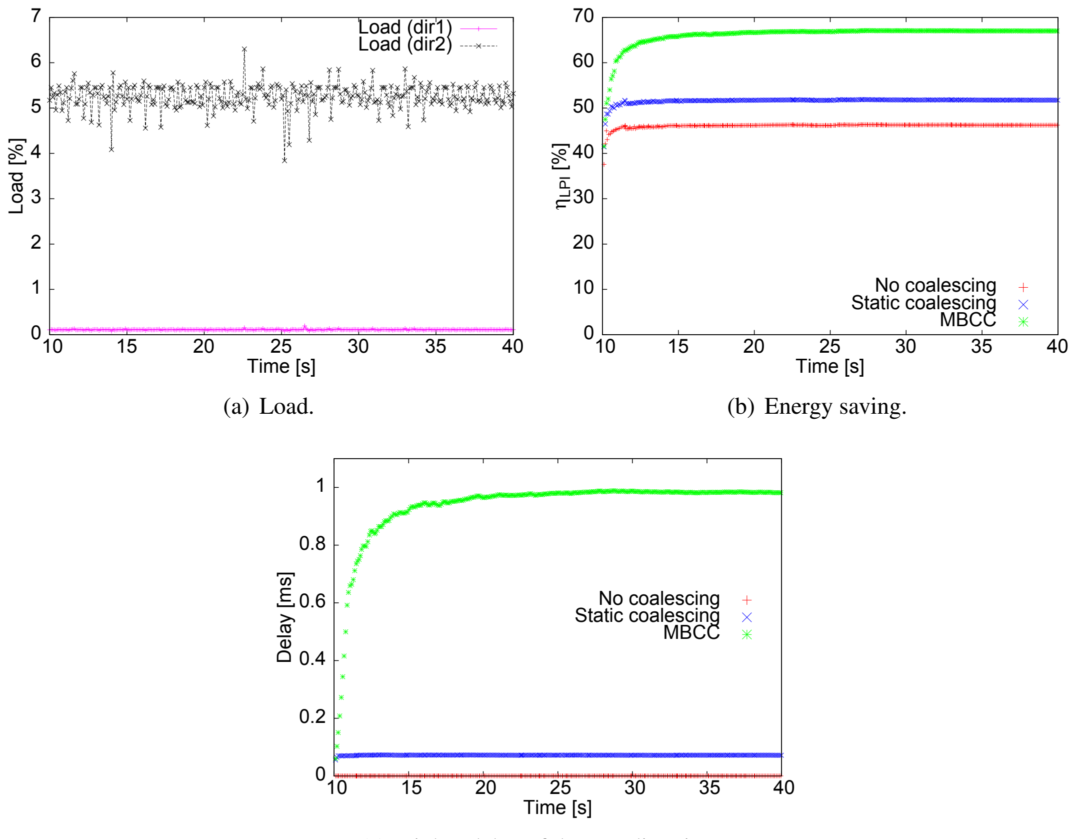


(c) Higher delay of the two directions.

Figure 8.4: Low load ($\rho_1 = 0.2\%$, $\rho_2 = 0.06\%$). MBCC and static coalescing practically save the same amount of energy.

(N_c = 1000 packets, δ = 1000 μs, γ = 10%), which is reported in boldface in Table 8.3.

Now I select a near-optimal configuration for MBCC, and I compare its performance with the best configuration of the static coalescing scheme. I use an additive increase, additive decrease scheme with δ =100 μs, and N_c =100 packets for MBCC, and T_c = 1300 μs, and N_c =10 packets for static coalescing. With those configurations, in Figures 8.4, 8.5, and 8.6 I plot the behavior over time of η_{LPI} and the higher of the two delays D_i for three different load combinations. In these figures, in addition to the performance MBCC and static coalescing, I also report the performance of EEE links without coalescing. Figure 8.4 illustrates the case of low load. Specifically, as shown in Figure 8.4(a), the load in either link direction does not exceed 1%, and energy saving of 90-95% can be achieved with or without coalescing (see Figure 8.4(b)). As concerns delay, Figure 8.4(c) shows that coalescing introduces considerable delay with respect to the case of plain EEE without coalescing. However, the delay, D_{target}, is below 1 ms. The medium load case of Figure 8.5 shows how MBCC manages to tradeoff delay for energy saving, while keeping the delay below 1 ms. Indeed, Figure 8.5(b) shows the huge energy saving gain due to the delay-controlled coalescing operation of my proposal. In Figure 8.6 I show a very dynamic case which combines high load with frequent and rapid load changes. We can still observe



(a) Load.

(b) Energy saving.

(c) Higher delay of the two directions.

Figure 8.5: Medium load ($\rho_1 = 0.11\%$, $\rho_2 = 5.3\%$). MBCC largely outperforms static coalescing at the expenses of delay (without exceeding the available delay budget).

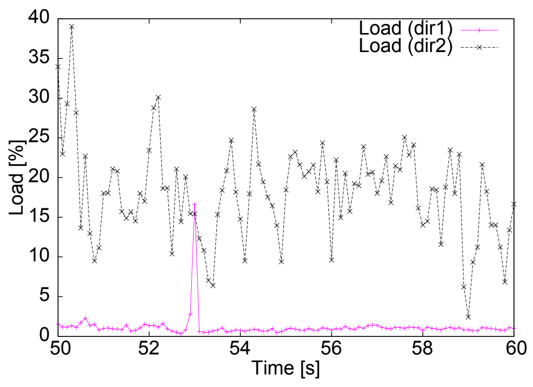
that my MBCC approach achieves a sevenfold gain with respect to plain EEE and a twofold gain with respect to static coalescing, while retaining the caused delay well below 1 ms. The impact of traffic variability is clear in the behavior of η_{LPI} and in the experienced delay. Interestingly, the performance comparison shows that MBCC is able to maintain a constant gain over time with respect to the other schemes.

In conclusion, the energy benefit due to delay-aware MBCC is remarkable under any traffic condition, including under quickly variable traffic conditions. Configuring MBCC schemes is easy, since it only requires to make reasonably simple decisions on the maximum size of the coalescing buffer (in the order of 100 packets) and on the δ parameter (in the order of milliseconds). The γ parameter is optional and, if used, has to be chosen as a small factor (in the order of 10%).

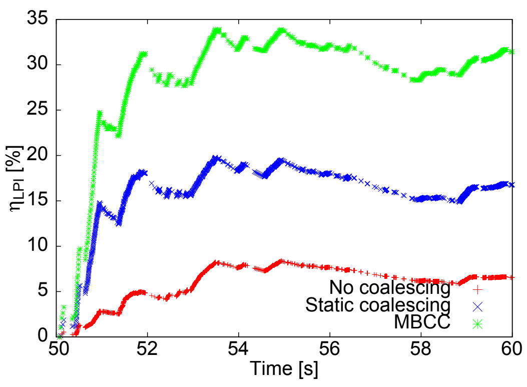
8.5. Economical Impact

The importance of EEE with MBCC can be seen using the following simple economical analysis. Let me consider a large data center, e.g., the one of OVH[1]. This data center contains 360,000

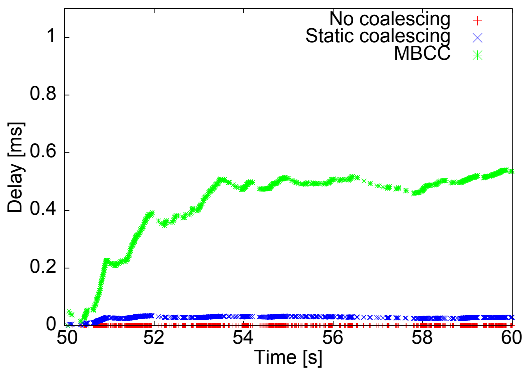
[1]OVH.com presentation: `http://www.youtube.com/watch?v=4e97g7_qSxA`



(a) Load.



(b) Energy saving.



(c) Higher delay of the two directions.

Figure 8.6: Highly variable load ($\rho_1 = 1.44\%$, $\rho_2 = 18.0\%$). MBCC doubles energy savings with respect to static coalescing.

physical servers, and each server has on average 3 connected network ports [11]. Assuming that all network ports have gigabit links, each port may consume between 2 W and 13 W using legacy Ethernet [85]. Typical load distributions are $\sim$ 40% of the links at almost zero load ($\leq 0.1\%$), $\sim$ 40% between 0.1% and 10% of load, and the rest of the links operate at higher loads [15]. Therefore I can use the results of Figures 8.4, 8.5 and 8.6 for an approximated economical analysis. Moreover, considering that the average cost of electricity in USA is about \$0.1/KWh, I can roughly estimate the cost of electricity for the network equipment of the servers of the aforementioned data center, using legacy Ethernet, plain EEE, EEE with static coalescing, or EEE with MBCC.

Thus, I will consider that on average an Ethernet card consumes 5 W and I further consider as averaged load values the ones I have in Figures 8.4, 8.5 and 8.6. With my calculations, the annual electricity bill of data center servers just due to the network would be ~\$4.73M using legacy Ethernet. This amount could be reduced almost by half accounting to ~\$2.23M by adopting EEE. EEE with static coalescing could further deduct another ~\$133K from the bill and, finally, MBCC could allow to save another ~\$400K resulting in a final bill of ~\$1.7M per year. Therefore, the adoption of MBCC could potentially reduce the electricity cost of a data center by $\sim$ 65% if compared with legacy Ethernet and by $\sim 25\%$ if compared with plain EEE. Practically, MBCC would quadruplicate the cost saving attainable with static coalescing.

In this simple estimate I exclude switches and other equipment such as air conditioning, CPU processing or server fans which could further contribute to the electricity cost reduction. Moreover faster Ethernet cards, i.e., 10, 40 and 100 $Gbps$, consume even more energy (at least two, three and five times more, respectively), so that the potential for energy saving is greater for higher data rates.

The cost of implementing coalescing is just adding a buffer to the NIC to support the packet aggregation but this might not be a problem since NICs have already integrated memory buffers and thus all we need is to reserve some space for coalescing. The cost of measurement-based coalescing control is negligible since it only requires software modifications on the driver side in order to apply the timer adaptation. Therefore, I believe that EEE with MBCC adjusting the coalescing timer is worth further research interest.

8.6. Summary

In this Chapter I estimated the potential savings that can be achieved by adopting the recently released IEEE Standard 802.3az and its enhancement (coalescing) instead of the legacy Ethernet. Overall, I observed traffic patterns yielding the possibility to save at least 40%, and up to more than 90%, in each gigabit link. Such a power saving would represent a non-negligible operational cost reduction for a data center, in the order of several millions of Dollars per year. Moreover, I evaluated my analytical model using a modified version of ns-3 simulator which integrates EEE compliant links. I further proposed an exhaustive performance evaluation to observe the impact of packet coalescing techniques to the energy saving and the packet delay. Specifically, I tested legacy EEE and coalescing algorithms using as input the collected packet traces. My study showed that EEE needs to be endowed with packet coalescing to achieve significant energy saving. Surprisingly, the first family of dynamic coalescing - NTCC - does not outperform static coalescing. However, MBCC achieves dramatic gain with respect to both static and NTCC coalescing algorithms.Specifically, I validated the superiority of MBCC using the real traffic traces I have captured, and I showed that my proposal can even double the energy saving benefit with respect to static and NTCC coalescing schemes. Moreover, from a purely economical point of view, MBCC can reduce the electricity cost of a data center by 65%. Notably, if compared to EEE with static coalescing, MBCC would quadruplicate cost savings on large data centers' electricity bill.

Part III

Energy Efficiency in Data Centers

Chapter 9

Measurements and Empirical Modeling

In this chapter, I focus on the characterization of data center servers' energy consumption. Indeed, in order to obtain full benefit of energy efficient techniques proposed in the literature [56, 67], it is crucial to profile the utilization of the data center servers' components. Moreover, it is necessary to understand the energy consumption of servers and how it is affected by different load configurations. There is a large body of work on characterizing servers' energy consumption. However, the existing literature does not jointly consider phenomena like the irruption of multicore servers and dynamic voltage and frequency scaling (DVFS) [93], which are key to achieve scalability and flexibility in the architecture of a server. Therefore, more complex/complete models which study the energy consumed by a server are needed. To be consistent, these models have to be based on empirical values. However, I found that there is a lack of empirical works studying servers' energy behavior.

In particular, I present an exhaustive empirical characterization of the power requirements of multiple components of data center servers. To do so, I devise different experiments to stress these components, taking into account the multiple available frequencies and the fact that I am working with multicore servers. In these experiments, I measure energy consumption of server components and identify their optimal operational points. My study proves that the curve defining the minimal CPU power utilization, as a function of the load in Active Cycles Per Second, is neither concave nor purely convex. Instead, it definitively shows a super-linear dependence on the load. Similarly, I present results on how to improve the efficiency of network cards and disks.

9.1. Baseline and CPU

As I mentioned in Chapter 4, for each server I have measured the power it uses with neither disk accesses nor network traffic. I assume that the power utilization observed is the sum of the baseline consumption plus the power used by the CPU. I have obtained samples of the power consumed under different configurations that vary in the number of active cores used, the frequency at which the CPU operates (all cores operate at the same frequency), and the active cores load (all active cores are equally loaded). The list of available and tested CPU frequencies and cores can be found in Table 4.1. I tune the total load ρ by using `lookbusy`, as described in Chapter 4. Each experiment lasts 30 s and it is repeated 10 times. Results are summarized in terms of average and standard deviation. Specifically, in the figures reported in this section, the power utilization for each tested configuration is depicted by means of a vertical segment centered on the average power utilization measured, and with segment size equal to two times the standard deviation of the samples.

The results of these experiments for each of the 3 servers analyzed (`Nemesis`, `Survivor` and `Erdos`) are presented in Figure 9.1 (the measurements for some frequencies and some number of cores are omitted for clarity). Here, for each configuration of number of active cores, frequency, and load in ACPS, the mean and standard deviation of all the experiments with that configuration are presented. Also the least squares polynomial fitting curve for the samples is shown for each number of cores and frequency. The curves shown are for polynomials of degree 7, but I observed that using a degree 3 polynomial instead does not reduce drastically the quality of the fit (e.g., the relative average error of the fitting increases from 0.7% with 7-th degree polynomials to 1.5% with degree equal to 3 for `Erdos`, while it remains practically stable and below 0.7% for `Nemesis`). In general, I can use an expression like the following to characterize the CPU power utilization:

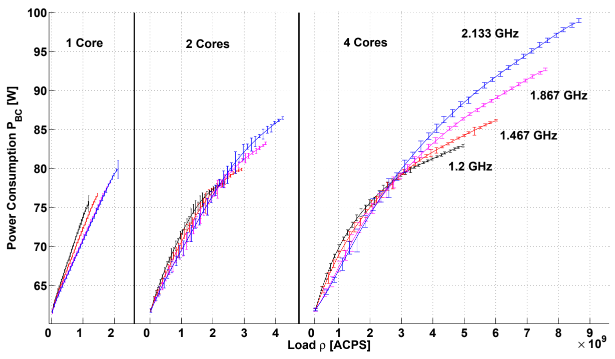
$$P_{BC}(\rho) = \sum_{k=0}^{n} \alpha_k \rho^k, \quad n \leq 7, \tag{9.1}$$

where P_{BC} includes both the baseline power utilization of the servers and the power used by the CPU, and ρ is the load expressed in active cycles per second. Therefore, coefficient α_0 in Eq. 9.1 represents the consumption of the system when the CPU activity tends to 0, and I can thereby interpret α_0 as the baseline power utilization of the system. Note that the polynomial fitting, and hence the baseline power utilization α_0, depends on the particular combination of number of cores and frequency adopted. However, for sake of readability, I do not explicitly account for such a dependency in the notation.

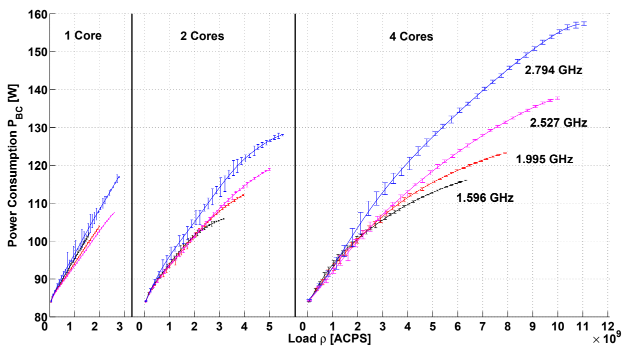
A first observation of the fitting curves for each particular server in Figure 9.1 reveals that the power for near-zero load is almost the same in all curves (e.g., for `Nemesis` this value is between 84 and 85 W). Observe that it is impossible to run an experiment in which the load of the CPU is actually zero to obtain the baseline power utilization of a server. However, all the fitting curves converge to a similar value for $\rho \to 0$, which can be assumed to represent the baseline power utilization.

A second observation is that for one core the curves grow linearly with the load. However, as soon as two or more cores are used, the curves are clearly concave, which implies that for a fixed frequency the efficiency grows with the load (I will discuss later the efficiency in terms of number of active cycles per energy unit).

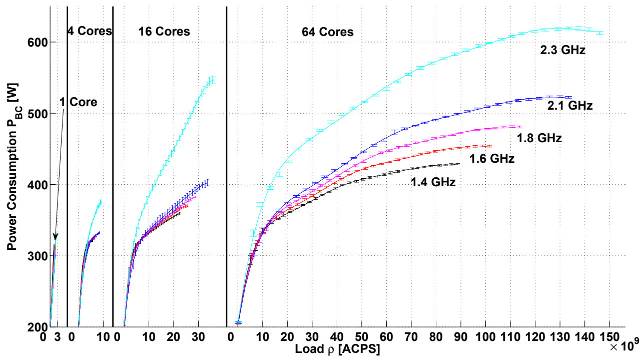
A third observation is that frequency does not significantly impact the power utilization when



(a) Survivor

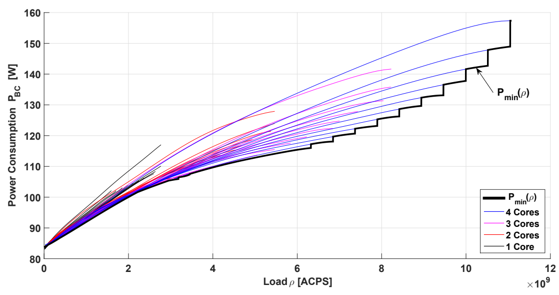


(b) Nemesis

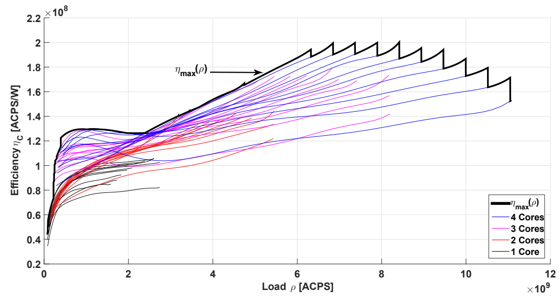


(c) Erdos

Figure 9.1: Power utilization of 3 servers (Survivor, Nemesis, and Erdos) for baseline and CPU characterization experiments. In each figure, the color of the curve identifies the frequency used.



(a) Minimal power.

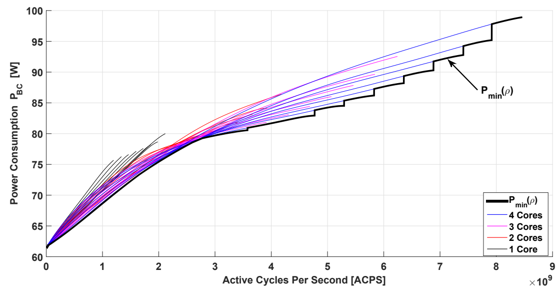


(b) Maximal efficiency.

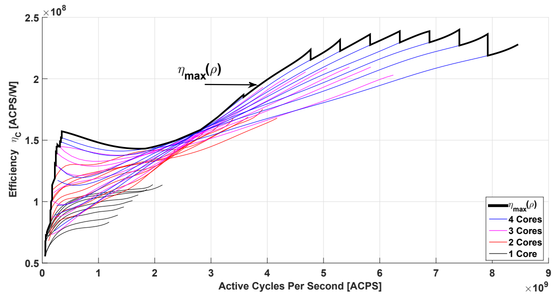
Figure 9.2: CPU performance bounds of `Nemesis`.

the load is low. In contrast, at high load, the power clearly increases with the CPU frequency. More precisely, the power grows superlinearly with the frequency, for a fixed load and number of cores. This is particularly evident in the curves characterizing `Erdos`, which is the most powerful among all three servers.

From the previous figures it emerges that the power utilization due to CPU and baseline can be minimized by selecting the right number of active cores and a suitable CPU frequency. Similarly, I can expect that the energy efficiency, defined as number of active cycles per energy unit, can be maximized by tuning the same operational parameters. I graphically represent the impact of operation parameters on power utilization and energy efficiency in Figures 9.2, 9.3 and 9.4 respectively for `Nemesis`, `Survivor` and `Erdos`. In particular, Figures 9.2(a), 9.3(a) and 9.4(a) report all possible fitting curves for the power measurements, plus a curve marking the lowest achievable power utilization at a given load. I name such a curve "minimal power curve" $P_{\min}(\rho)$, and I observe that (i) it only depends on the load ρ, and (ii) it is a piecewise concave function, which makes it suitable to formulate power optimization problems. Finally, to evaluate the energy efficiency of the CPU, I report in Figures 9.2(b), 9.3(b) and 9.4(b) the number of active cycles per energy unit obtained from the measurements respectively for `Nemesis`, `Survivor` and `Erdos`. I compute the power due to active cycles as the power $P_{BC} - \alpha_0$, i.e., by subtracting the baseline consumption from P_{BC}, and I obtain the efficiency η_C by dividing the load (in active cycles per second) by the power due to active cycles, i.e., $\eta_C = \frac{\rho}{P_{BC}(\rho)-\alpha_0}$. Also in this case I show the curve that maximizes the efficiency at a given load, which I name "Maximal efficiency curve" $\eta_{\max}(\rho)$. Interestingly, I observe that (i) $\eta_{\max}(\rho)$ presents multiple local maxima, (ii) for

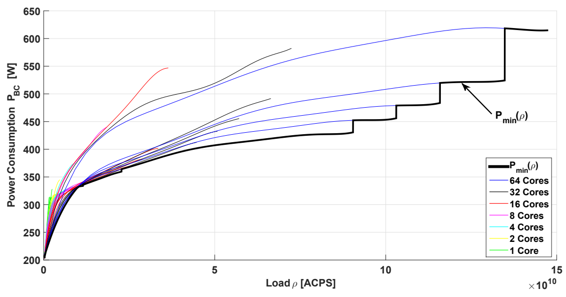


(a) Minimal power.

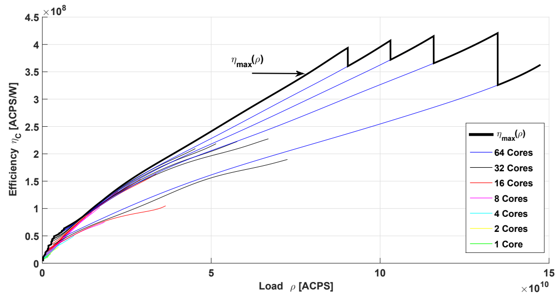


(b) Maximal efficiency.

Figure 9.3: CPU performance bounds of Survivor.



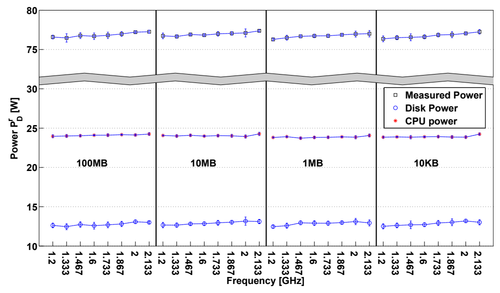
(a) Minimal power.



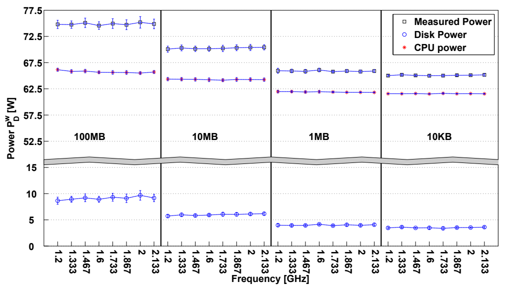
(b) Maximal efficiency.

Figure 9.4: CPU performance bounds of Erdos.

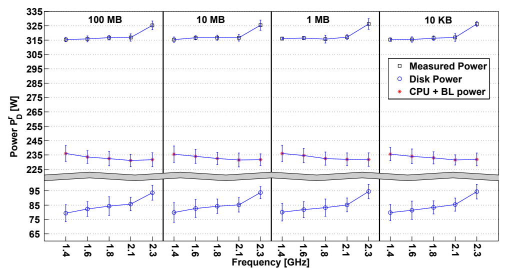
a given configuration of frequency and number of active cores, the efficiency is maximized at the highest achievable load, (iii) all local maxima corresponds to the use of all available active cores,



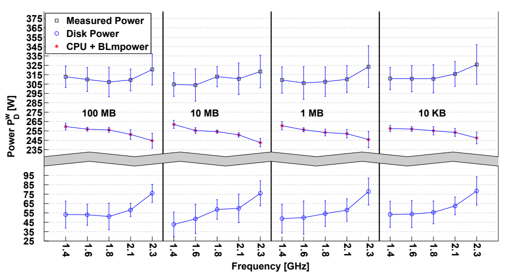
(a) Power utilization during reading (`Survivor`).



(b) Power utilization during writing (`Survivor`).



(c) Power utilization during reading (`Erdos`).



(d) Power utilization during writing (`Erdos`).

Figure 9.5: Instantaneous power utilization for a reading/writing operations. Results are presented for every frequency and for 4 different block sizes for each one of the servers.

but (iv) the absolute maximum is *not* achieved neither at the highest CPU frequency nor at the lowest.

9.2. Disks

I now characterize the power and energy consumption of disk I/O operations. During the experiments, I continuously commit either read or write operations, while keeping the CPU load ρ as low as possible (i.e., disconnecting the network and not running other tasks). Still, the power measurements obtained during the disk experiments contain both the power used by the disk and power due to CPU and baseline. Indeed, Figure 9.5 shows, for each experiment, the total measured power P_t, the power P_{BC} computed according to Eq. 9.1 at the load ρ measured during the experiment, and the power due to disk operations, computed as $P^x_D = P_t - P_{BC}(\rho), \quad x \in \{r, w\}$, where r and w refer to reading and writing operations, respectively. I test sequentially all the available frequencies for each server (see Table 4.1), and I/O block sizes ranging from 10 *KB* to 100 *MB*. Figure 9.5 shows average and standard deviation of the measures over 10 experiment repetitions. Results for `Nemesis` are omitted since they are like `Survivor`' results. Indeed, `Survivor` and `Nemesis` have similar disks and file systems, while `Erdos` is equipped with SAS disks with RAID. In all cases shown in the figure, the disk power is small but not negligible with respect to the baseline consumption. Furthermore, I can observe that the two servers presented behave differently. Indeed, while the power utilization due to writing is affected

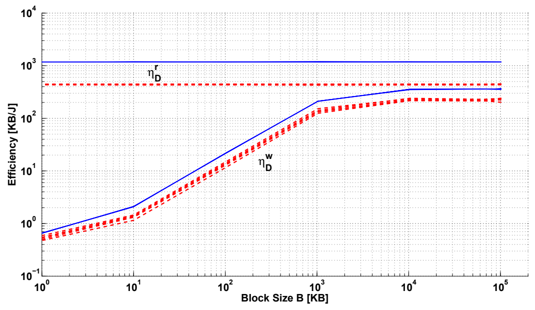


Figure 9.6: Disk reading and writing efficiencies for `Erdos` (red dotted lines) and `Nemesis` (blue solid lines).

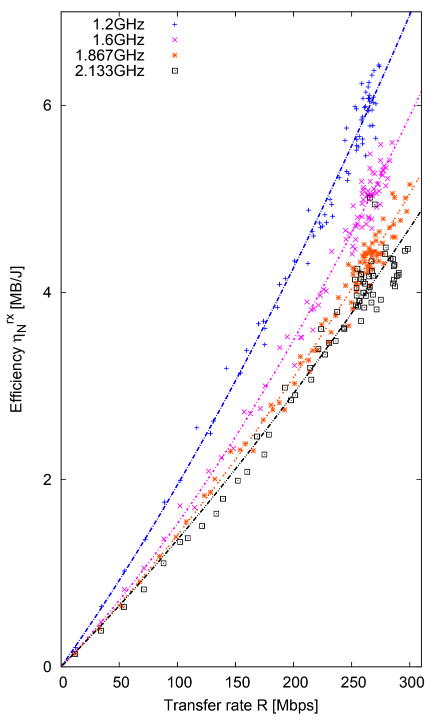
by the block size B for both machines, I observe that `Survivor`' disk writing power P_D^w is not affected by the CPU frequency, while `Erdos`' results show an increase with the frequency. A similar behavior is also observed for the reading power of the disk. The main difference is that reading is a more costly operation since the power consumption in reading is approximately 30% higher as can be observed in Figures 9.5(a),9.5(b) and Figures 9.5(c),9.5(d).

Moreover, the results obtained with `Erdos` are affected by a substantial amount of variability in the measurements, which I believe is due to the caching operations enforced by the RAID mechanism in `Erdos`. Furthermore, `Erdos` shows a baseline plus CPU power decrease for both reading and writing. This behavior is because `Erdos` is very powerful and higher CPU frequencies finish the workload faster (keep in mind that disks are the bottleneck for disk-intensive tasks) and therefore in accordance with Figure 9.1 the average load ρ will be lower when higher frequencies are used.

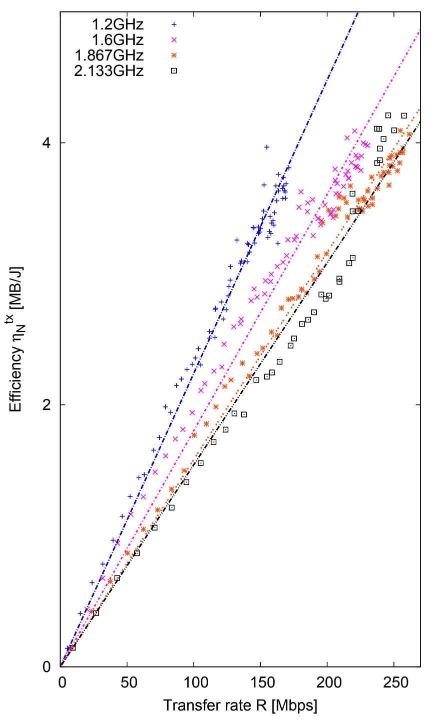
Similarly to what was described for the CPU, I can also compute the energy efficiencies η_D^r and η_D^w of disk reading and writing operations, respectively. Figure 9.6 reports efficiency as a function of the I/O block size, and shows one line per each CPU frequency. This efficiency can be computed by subtracting the baseline power from the total power, and by measuring the volume V of data read or written in an interval T as $\eta_D^x = \frac{V}{P_D^x T}, \quad x \in \{r, w\}$. I can observe that results are similar for all the servers. Specifically, reading efficiency is almost constant at any frequency and for each block size, while writing is more efficient with large block sizes. Also, the efficiency changes very little with the adopted CPU frequency. Efficiency, however, saturates to a disk-dependent asymptotic value, which is due to the mechanical constraints of the disk (e.g., due to the non-negligible *seek* time, the number of read/write operations per second is limited). In addition, although not visible in the figure due to the log-scale adopted, η_D^w is a concave function of the block size B.

9.3. Network

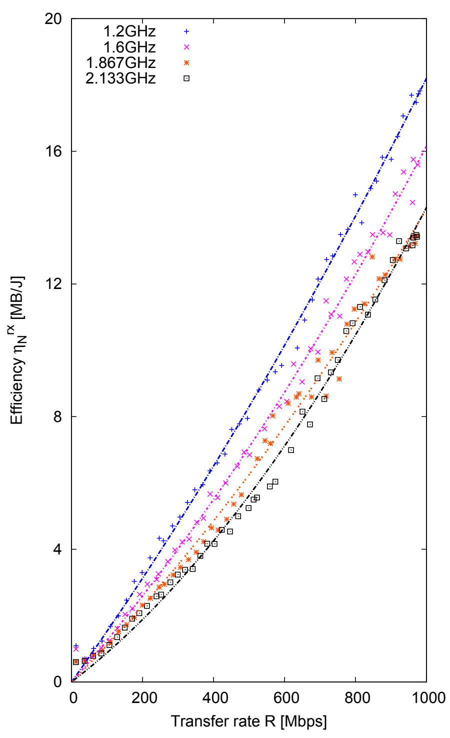
The last server component that I characterize via measurements is the NIC. Similarly to CPU and disk, I run experiments in which only the operating system and the test scripts are active. For the network, I run the scripts described in § 4.3.5 to transmit (receive) traffic over a gigabit



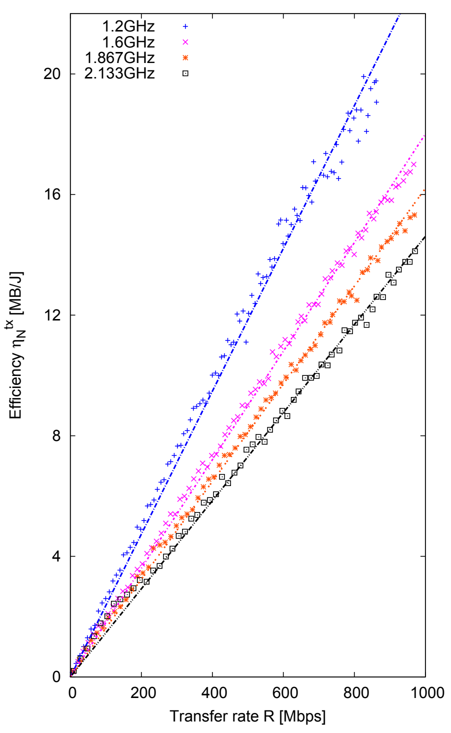
(a) Receiver Efficiency (64-B packets).



(b) Sender efficiency (64-B packets).

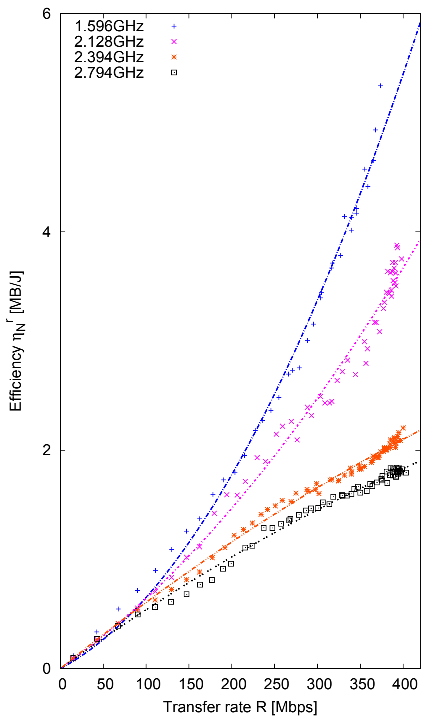


(c) Receiver efficiency (1470-B packets).

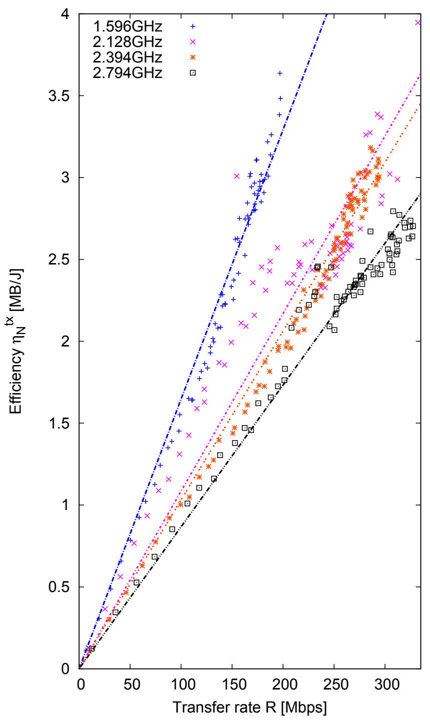


(d) Sender efficiency (1470-B packets).

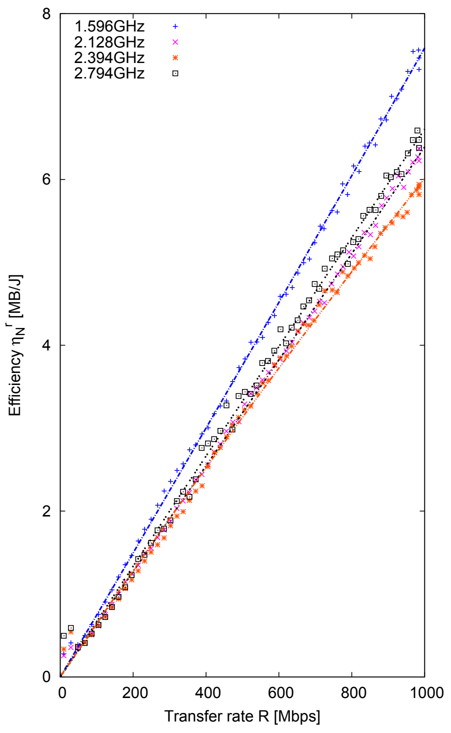
Figure 9.7: Network efficiencies for different frequencies and 64-B (upper) and 1470-B (bottom) packets (`Survivor`).



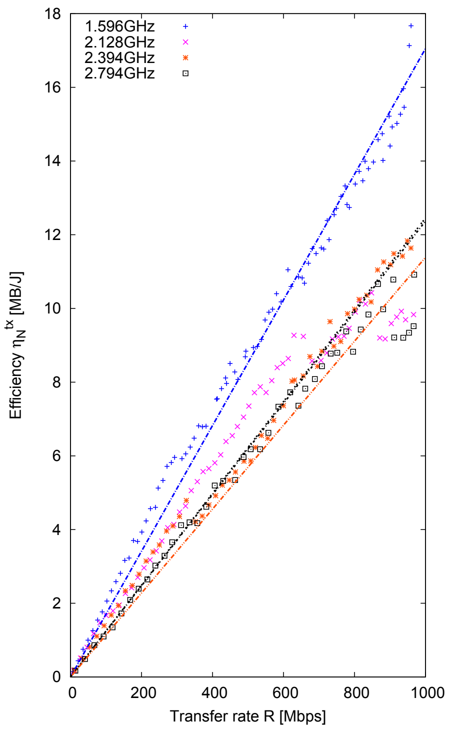
(a) Receiver efficiency (64-B packets).



(b) Sender efficiency (64-B packets).

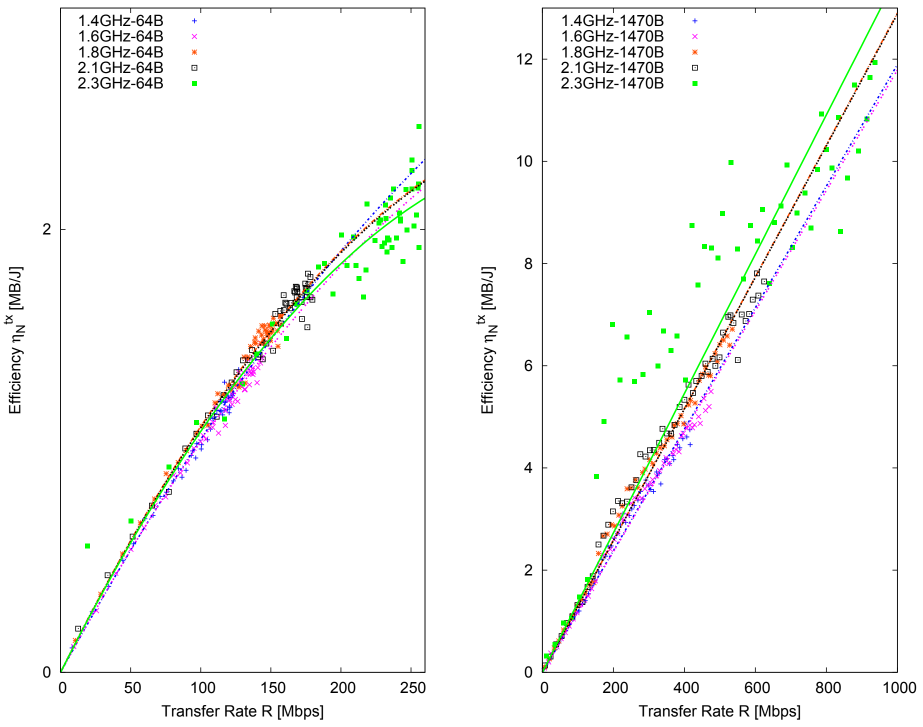


(c) Receiver efficiency (1470-B packets).



(d) Sender efficiency (1470-B packets).

Figure 9.8: Network efficiencies for different frequencies and 64-B (upper) and 1470-B (bottom) packets (`Nemesis`).



(a) Sender efficiency (Erdos). (b) Sender efficiency (Erdos).

Figure 9.9: Network efficiencies for different frequencies and 64-B (upper) and 1470-B (bottom) packets (Erdos).

Ethernet connection and count the system active cycles ρ. I measure the total power utilization P_t during the experiment, so that the power due to network activity can be then estimated as $P_N^x = P_t - P_{BC}(\rho)$, $x \in \{tx, rx\}$, where P_N^{tx} and P_N^{rx} refer to the power consumed when acting as a sender and as a receiver, respectively.

In the experiments, I sequentially test all the available frequencies for each server (see Table 4.1), and fix the packet size and transmission rate within the achievable set of rates (which depends on the packet size, e.g., < 950 *Mbps* for 1470-B packets). I report the results for the network energy in terms of efficiencies η_N^{tx} and η_N^{rx} (volume of data transferred per unit of energy). These efficiencies are computed as $\eta_N^x = \frac{R}{P_N^x}$, $x \in \{tx, rx\}$, where R is the transmission rate during the experiment.

Figures 9.7, 9.8 and 9.9 show the network efficiencies of Survivor, Nemesis and Erdos averaged over 5 samples per transmission rate R[1]. For the sake of readability, the figures only shows results for the biggest and smallest packet sizes, i.e., 64-B and 1470-B packets. For Nemesis and Survivor I report four CPU frequencies: the lowest, the highest, the most efficient (according to Figures 9.2(b) and 9.3(b)) and an intermediate one, while all five available frequencies for Erdos are shown. The figure also reports the polynomial fitting curves for efficiency, which I found to be at most of second order. Since the efficiency is represented in terms

[1]Network results are obtained by using a point-to-point Ethernet connection between two controlled servers.

of network activity only, in the fitting I force the zero-order coefficient of the polynomials to be 0. Therefore, I can characterize the network efficiencies of the servers as $\eta_N^x = \beta_1 R + \beta_2 R^2$, $x \in \{tx, rx\}$, where the β_i coefficients are computed by minimizing the least square error of the fitting.

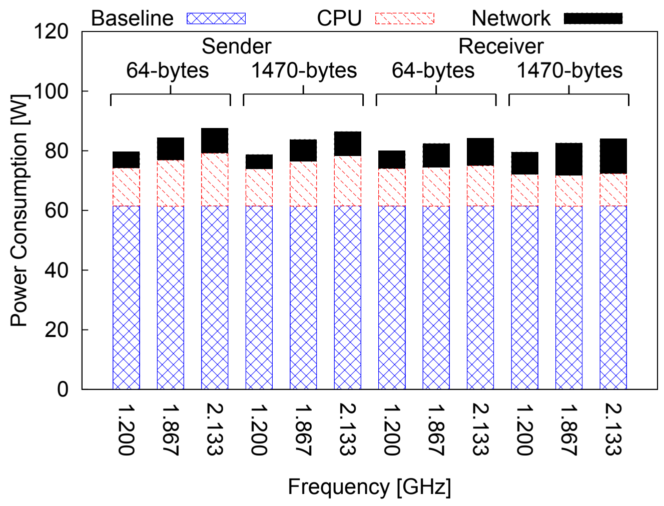
It can be observed in Figures 9.7, 9.8 and 9.9 that efficiencies are almost linear or slightly superlinear with the transfer rate, e.g., the receiving efficiency of `Survivor` exhibits an evident quadratic behavior (Figures 9.7(a) and 9.7(c)). Indeed, my measurements show that the network power utilization is independent from the throughput, which is a well known result for legacy Ethernet devices. In fact, the NICs of the servers are not equipped with power saving features like, e.g., the recently standardized IEEE 802.3az I have studied in Part II of this thesis.

In all cases, the efficiency is strongly affected by the selected CPU frequency. Moreover, efficiency is also affected by packet size, although the impact of packet size changes from server to server, e.g., `Survivor` sending efficiency is only slightly affected by it.

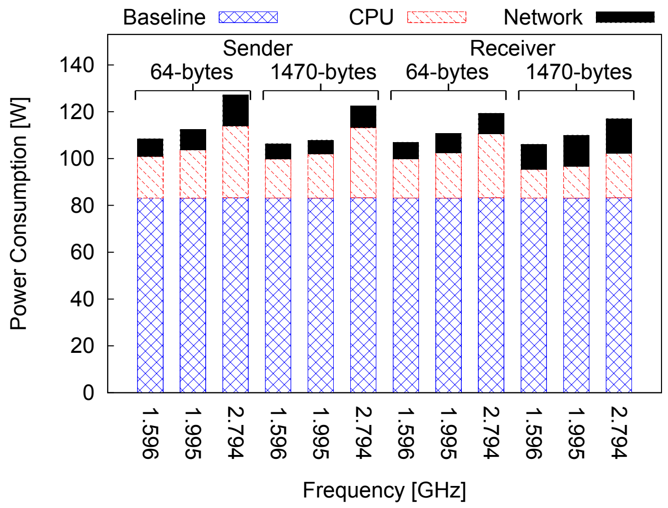
Another observation is that, depending on the packet size and frequency used, sending can be more energy efficient than receiving at a given transmission rate, and using the highest CPU frequency is never the most efficient solution. Note also that the efficiency decreases with the packet size, although this effect is particularly evident at the receiver side, while it only slightly impacts the efficiency of the packet sender. However, network activity causes non-negligible CPU activity, as shown in Figure 9.10 for a few experiment configurations for all three servers. Overall, the lowest CPU frequency yields the lowest total power utilization during network activity periods.

9.4. Summary

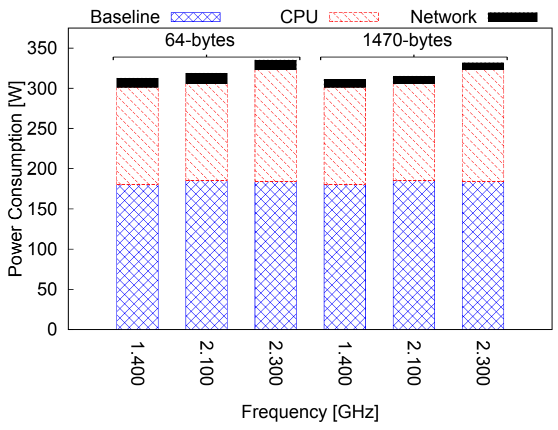
In this chapter I have reported my measurement-based characterization of energy and energy consumption in a server. I have exhaustively measured the energy consumed by CPU, disk, and NIC under different configurations, identifying the optimal operational levels, which usually do not correspond to the static system configurations commonly adopted. I found that, besides the *baseline component*, which does not changes significantly with the operational parameters, the CPU has the largest impact on energy consumption among all the three components. I observe that CPU consumption is neither linear nor concave with the load, i.e., the systems are not *energy proportional*. Disk I/O is the second larger contributor to energy consumption, although performance changes sensibly with the I/O block size used by the applications. Finally, the NIC activity is responsible for a small but not negligible fraction of energy consumption, which scales almost linearly with the network transmission rate. In general, most of the energy/power performance figures do not scale linearly with the utilization, in contrast to what is commonly assumed in the literature. In the next chapter, I first show how to use the characterization described in this chapter to predict power consumption for real applications running on the servers. Afterwards, I experimentally validate the derived model.



(a) `Survivor` (sender-receiver).



(b) `Nemesis` (sender-receiver).



(c) `Erdos` (sender).

Figure 9.10: Power utilization with network activity for `Survivor`, `Nemesis`, `Erdos` and (64-B experiments were run with a transmission rate $R = 150$ *Mbps*, while $R = 400$ *Mbps* for the experiments with 1470-B packets).

Chapter 10

Energy Consumption Estimation and Model Validation

While the results presented in the previous chapter are useful to understand the energy consumption pattern of CPU, disk and network, I believe that a much more important use of these results is being able to estimate the energy consumption of applications. In this chapter I describe how this can be done from simple data about the application. Moreover, I validate the accuracy of the model derived from my characterization by comparing the real energy consumed by two Hadoop applications - PageRank and WordCount - with the estimation from my model, obtaining errors below 4.1% on average.

10.1. Energy Estimation Hypothesis

The approach I propose to estimate the energy E_{app} consumed by an application lays on the basic assumption that the energy is essentially the sum of the baseline energy E_B (baseline power times application running time), the energy consumed by the CPU E_C, the energy consumed by the disk E_D, and the energy consumed by the network interface E_N:

$$E_{app} = E_B + E_C + E_D + E_N. \tag{10.1}$$

Hence, the process of estimating E_{app} is reduced to estimating these four terms. In order to estimate the first two terms, I need to know the total number of active cycles that the application will execute, C_{app}, and the load ρ_{app} (in ACPS) that the execution will incur in the CPU. From this, the total running time T_{app} can be computed as $T_{app} = C_{app}/\rho_{app}$. Then, once the number of cores and the frequency that will be used have been defined, it is also possible to estimate the baseline plus the CPU energy consumption $E_B + E_C$. For this estimation, I use the fitting curves in Figure 9.1 to extract the power utilization, P_{BC}, and multiplying by the execution time of the application, T_{app}, I get the corresponding energy consumption:

$$E_B + E_C = P_{BC}T_{app} = P_{BC}C_{app}/\rho_{app}. \tag{10.2}$$

The energy consumed by the disk is simply the energy consumed while reading and writing, i.e., $E_D = E_D^r + E_D^w$. To estimate these latter values, the block size to be used has to be decided, from which I can obtain an estimate of the efficiency of reading, η_D^r, and writing, η_D^w. These, combined with the total volume of data read and written by the application, denoted as V_D^r and V_D^w respectively, allow to obtain the estimate energy as

$$E_D = \frac{V_D^r}{\eta_D^r} + \frac{V_D^w}{\eta_D^w}. \tag{10.3}$$

Finally, to estimate E_N, the transfer rate R and the packet size S have to be chosen, which combined with the frequency used, yield sending and receiving efficiencies η_N^{tx} and η_N^{rx} (see Figures 9.7, 9.8 and 9.9). Then, if the total volumes of data to be sent and received are V_N^{tx} and V_N^{rx}, respectively, the energy spent due to network is as follows:

$$E_N = \frac{V_N^{tx}}{\eta_N^{tx}} + \frac{V_N^{rx}}{\eta_N^{rx}}. \tag{10.4}$$

Summing up Eqs. 10.2, 10.3, and 10.4 I obtain the estimated E_{app}.

10.1.1. Applications and Scenarios for Validation

In this subsection I present the applications and scenarios I experimented with in order to validate the model presented in § 10.1. My goal was to be able to estimate the energy consumed by an application deployed on a data center based on the usage of its different components. For that, I executed two different Hadoop applications, PageRank and WordCount, in three different scenarios: first with no network, second with a server connected to the network, and finally with a two-server cloud. For the first two scenarios I used `Nemesis`, whereas, for the cloud case, I used both `Nemesis` and `Survivor`. I describe the applications and the scenarios in detail below.

My first application is a **Hadoop Map-Reduce PageRank** based application that follows the approach from Castagna [20]. This application, that I denote PageRank for simplicity, computes several iterations of the pagerank algorithm on an Erdos-Renyi random (directed) graph with 1 million nodes and average degree 5[1]. The execution of the PageRank application has three phases: preprocessing, map-reduce, and postprocessing. On its side, the map-reduce phase is a sequence of several homogeneous iterations of the PageRank algorithm that runs until a certain threshold is met. For simplicity, I only estimate the energy consumed during the map-reduce phase of the pagerank algorithm, which I force to run 10 times.

[1]My PageRank algorithm assigns one input graph to each mapper so, in order to have one map task in each machine, two instances of this graph had to be used in the cluster scenario.

My second application is the **Hadoop Map-Reduce WordCount**. This is a simple program that reads text files and counts how often words occur. For WordCount I use a few hundreds of books as input and estimate the energy consumed for the whole map-reduce process.

As I have mentioned above, these applications are run in 3 different scenarios. In the first scenario, denoted as **Isolated Server**, I run Hadoop in `Nemesis` keeping it disconnected from the network. When I run my applications in this scenario I am basically measuring the impact on the energy consumption of the baseline, CPU and hard disk.

In the second scenario, denoted as **Connected Server**, I run Hadoop in `Nemesis` while it exchanges data on a gigabit LAN. In order to measure the effect of the network on the energy consumption, I evaluate 4 different cases for each application. These cases result from combining 2 different behaviors, depending on whether `Nemesis` acts as a sender or as a receiver of data, with 2 different packet sizes, 64 and 1470 bytes. To do so, I run Iperf, as a server or as a client according to the case, in parallel with Hadoop.

Finally, in the third scenario, denoted as **Cloud**, I set up a two-server Hadoop cluster with `Nemesis` and `Survivor`. In this scenario `Nemesis` is configured as the master node of the cluster and `Survivor` as a slave node. The execution of the applications is shared by both nodes so Hadoop itself exchanges traffic between both servers, and I do not insert additional network traffic in this case. Finally, in order to have a better control of the experiment, I force the reduce tasks to be mandatorily run in `Nemesis`, which also conditions the way the data is exchanged between `Nemesis` and `Survivor`.

Observe that all 3 scenarios are based on Hadoop. This implies that, apart from the map and reduce tasks due to the applications being run, there are some extra processes executed in the servers I am using. The most important processes that I can find in `Nemesis` are *NameNode* (the process that keeps the directory tree of all files in the file system, and tracks where across the cluster the file data is kept), *Secondary NameNode* (that performs periodic checkpoints of the *NameNode*), *DataNode* (the process that is in charge of storing data in the Hadoop File System (HDFS)), *JobTracker* (that receives the jobs and submits MapReduce tasks to the cluster nodes) and *TaskTracker* (a per node process that can accept a determined number of MapReduce tasks). On its side, `Survivor` runs, in the cloud scenario, DataNode and TaskTracker.

10.2. Experiments and Observed Results

For the sake of consistency in the results, I ran both applications 10 times per frequency for each one of the considered scenarios and averaged the results.

I start by describing the **Isolated Server** scenario. For each run i I record the total number of active cycles executed C^i_{app}, the time spent T^i_{app} and the volume of data read (written), $V_D^{r,i}$ ($V_D^{w,i}$). Since I cannot measure the instantaneous CPU load, I assume that the CPU load is the same during the run for a given frequency. Hence, it can be estimated as $\rho^i_{app} = C^i_{app}/T^i_{app}$. Then, from ρ^i_{app} I obtain the estimate of the instantaneous power P^i_{BC} using the fitting curves

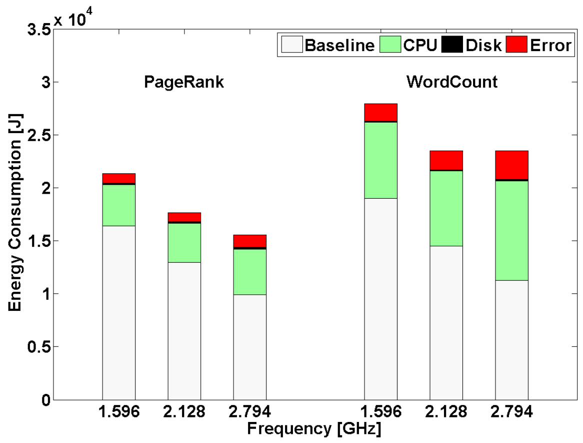


Figure 10.1: Energy consumption of `Nemesis` in the **Isolated Server** scenario.

as described in § 4.1. Finally, using Eq. 10.2 I compute the estimate $E_B^i + E_C^i$. In order to estimate the energy consumed by the disk operations, I use the fact that Hadoop uses a block size of 64 MB. This allows to estimate the reading (writing) efficiencies, $\eta_D^{r,i}$ ($\eta_D^{w,i}$) that I compute, in Joules per byte. Combining these values with the measured volume of data read and written ($V_D^{r,i}$ and $V_D^{w,i}$), as described in Eq. 10.3, I obtain E_D^i.

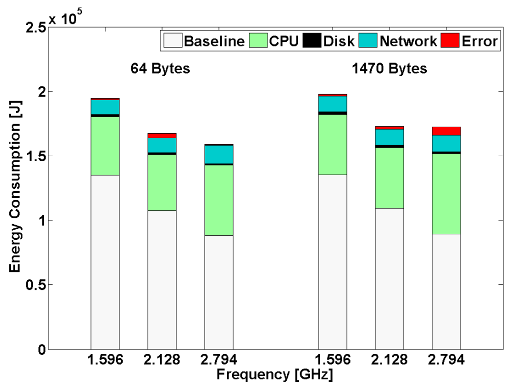
The total estimated energy of the application, E_{app}^i, is obtained by summing up the energy of the different components used in run i, as stated in Eq. 10.1 (remember that, in the **Isolated Server** the network is not used). I sum the values of the ten runs of an experiment and I get the estimated $E_{app} = \sum_{i=1}^{10} E_{app}^i$. The (approximated) total *real* energy $\hat{E}_{app}^i$ consumed in run i is computed by the average value of the power samples which I registered with the power analyzer during the run, and I multiply it with the run time T_{app}. Then, the total energy consumed by the experiment is obtained as $\hat{E}_{app} = \sum_{i=1}^{10} \hat{E}_{app}^i$. The estimation error for each experiment is then computed as $\hat{E}_{app} - E_{app}$.

I show the results obtained for the **Isolated Server** scenario with the minimum, the maximum, and the most efficient[2] frequencies (the results for the remaining frequencies are similar) in Figure 10.1. The figure shows the results for both PageRank and WordCount. As can be seen, the error is relatively small, except for the case when I run WordCount at the maximum frequency. Errors are of 4%, 4%, 7%, 5%, 7% and 10% respectively, following the same order as in Figure 10.1.

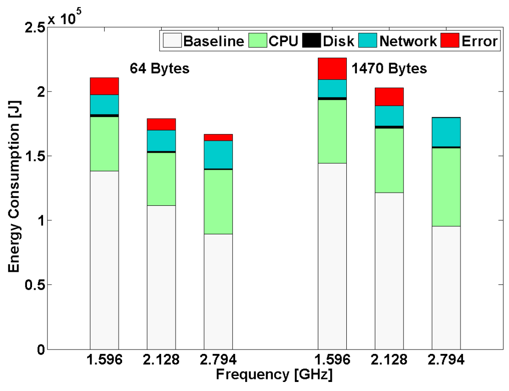
I move now to the **Connected Server** scenario. As I described in the previous section, this scenario is studied in 4 different cases depending on whether `Nemesis` acts as sender or receiver and whether the size of the packets is of 64 or 1470 bytes. Of course, another relevant parameter is the rate at which these packets are sent. The rates used are 150 and 400 Mbps when using packets of 64 or 1470 bytes, respectively.

The total energy consumed in these cases is computed in the same way as I did for the **Isolated Server** scenario but adding the contribution of the network. In order to estimate the network consumption in one run with `Nemesis` sending traffic (resp., receiving traffic), the sending ef-

[2]Respectively 1.596, 2.128 and 2.794 GHz, according to the results shown in § 4.1.

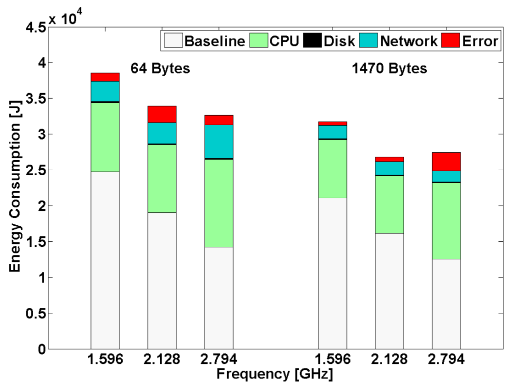


(a) PageRank, sender side.

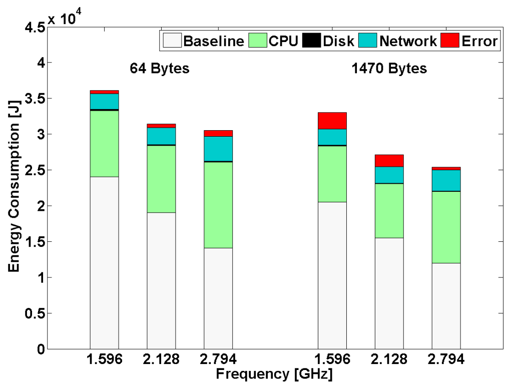


(b) PageRank, receiver side.

Figure 10.2: Energy consumption of `Nemesis` running PageRank in the **Connected Server** scenario, with either small or big packets.



(a) WordCount, sender side.



(b) WordCount, receiver side.

Figure 10.3: Energy consumption of `Nemesis` running WordCount in the **Connected Server** scenario, with either small or big packets.

ficiency η_N^{tx}, (resp., receiving efficiency η_N^{rx}) is obtained from the transfer rate R, the frequency and packet size used (see Figures 9.7, 9.8 and 9.9). The amount of data sent (resp., received) can be obtained from the server itself by consulting the OS registers[3]. Therefore, the energy of the network for an run i, E_N^i, is obtained using Eq. 10.4. Then, including E_N^i for each run in the computation of E_{app}^i I can obtain the total energy consumed by the application. Following the same steps as in the previous scenario, I get the results shown in Figure 10.2 and 10.3. The error measured is again relatively smaller for PageRank than for WordCount. The error measured for each of the cases can be found in Table 10.1.

I finally analyze the **Cloud** scenario. In this scenario I set up a cluster with two servers, `Nemesis` and `Survivor`, and run the 2 aforementioned Hadoop applications in it. This scenario may seem relatively similar to the **Connected Server** scenario, but it has is a major differ-

[3]We can read the registers rx_bytes, rx_packets, tx_bytes or tx_packets from `/sys/class/net/eth0/statistics`.

Table 10.1: Error measured in the different cases of the **Connected Server** scenario.

Packet Size	Freq	Cases			
		PR - Send	PR - Rec	WC - Send	WC - Rec
64-B	1.596	0.5%	6.0%	2.9%	2.7%
	2.128	2.0%	4.7%	6.4%	1.5%
	2.794	0.5%	2.9%	4.0%	2.9%
1470-B	1.596	0.7%	6.9%	1.6%	6.5%
	2.128	1.1%	6.5%	5.8%	5.8%
	2.794	3.8%	0.3%	0.9%	1.5%

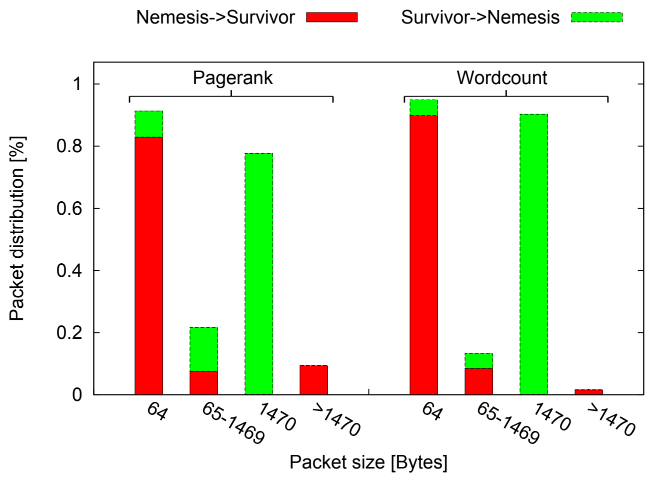
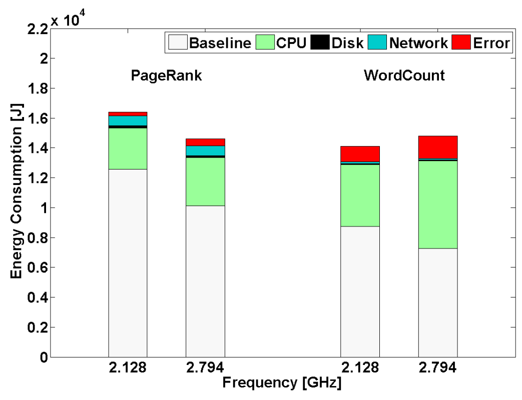


Figure 10.4: Distribution of the sizes of the packets exchanged between `Nemesis` and `Survivor` for both PageRank and WordCount in the **Cloud** scenario.

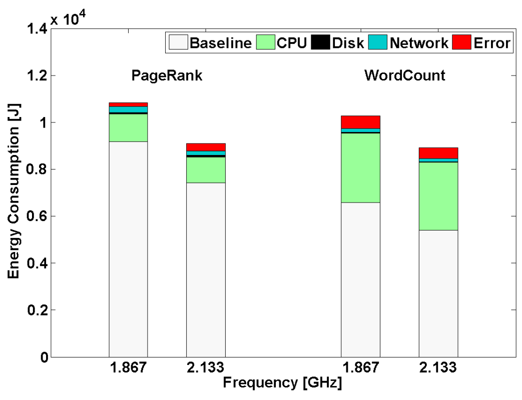
ence. While in the previous scenario I was the one controlling the network traffic, here the traffic is controlled by Hadoop. Specifically, I know that, in this scenario, there are two main sources of traffic: requesting input data when it is not present in a server, and sending the mapper tasks outputs to the reducer tasks. The only condition I impose in the server to have some control over the traffic is related to this later aspect, I force the reducers to be always in `Nemesis`.

Although I am able to retrieve the total amount of data received or sent by each server, I know neither the size of the packets used nor the rate. Therefore, I can compute neither the sending efficiency η_N^{tx} nor the receiving efficiency η_N^{rx}. In order to be able to compute both the sending and receiving efficiencies I analyze the traffic exchanged by both servers for each one of the applications. Figure 10.4 shows the amount of packets of each size that were exchanged by both servers (and the direction of the exchange) for both applications. The results show the vast majority of packets are either small (64 bytes) or big (1470 bytes). Moreover, it shows that most of the packets sent from `Nemesis` to `Survivor` are small packets for both applications, while big packets are sent in the opposite direction.

Given these results, I approximate the energy consumed by the network assuming that all the packets exchanged are of the same size and that the rate is the maximum achievable rate for each packet size according to the results from § 4.1. For instance, I consider roughly 30 Mbps when Survivor receives 64-Byte packets and roughly 970 Mbps if it sends 1470-Byte packets. These assumptions allow me to compute now η_N^{tx} and η_N^{rx}. The remaining parameters are computed as



(a) Nemesis



(b) Survivor

Figure 10.5: Energy consumption of `Nemesis` and `Survivor` in the **Cloud** scenario.

for the other scenarios, so to determine $\hat{E}_{app}$ and E_{app}. The results are shown in Figure 10.5. As in the previous scenarios, errors are relatively low. In particular, the error in `Nemesis` when running PageRank is 3.1% and 1.4% for 2.128 GHz and 2.794 GHz, respectively, and of a 9.7% and a 6.5% for 2.128 GHz and 2.794 GHz when running WordCount. On the other hand, the measured errors for `Survivor` are 3.3% and 3.6% for 1.867 GHz and 2.133 GHz when running PageRank and 5.1% and 5.2%, respectively, when running WordCount.

10.3. Discussion

I discuss now some of the implications of those results. I start with consolidation as a technique for energy saving. It has been often assumed that the best way of saving energy is by using the highest frequency available and applying consolidation (which is to fill servers as much as possible). This reduces the total number of servers being used, allowing to switch off the rest. This assumption has led to proposing bin-packing based solutions [4, 65, 71, 92]. However, the results presented in Figures 9.2(b), 9.3(b) and 9.4(b) show that the highest frequency is not always the most efficient one, and this has been found to be true for two different architectures (Intel and AMD). This implies that, by running servers at the optimal amount of load, and the right frequency, a considerable amount of energy could be saved.

A second relevant aspect is the baseline consumption of servers. The results presented for all three servers show that their baselines are within a 30-50% of the maximum consumption. Then, it is obvious that more effort has to be done for reducing baseline consumption. For instance, a solution could consist in switching off cores in real time, not just disabling them, or in introducing very fast transitions between active and lower energy states, i.e., to achieve real *suspension* in idle state.

There is another relevant issue related to the CPU load associated to disk and network activity. It can be observed in Figure 9.5 that disks do not incur much CPU overhead. In fact, the power used by the CPU plus baseline does not change much across the experiments. Instead, the energy consumed by the CPU due to network operations is even larger than the energy consumed by the NIC (see Figure 9.10). Some works [38] have already pointed out that the way packets are handled by the protocol stack is not energy efficient. The results reinforce this feeling and point out that building a more efficient protocol stack would certainly reduce the amount of energy consumed due to the network.

Finally, it is worth to mention that in this work I have assumed that the power utilization of the RAM memory is included in the *baseline*. The characterization experiments have been run in such a way that there were few memory accesses, so its power utilization did not affect my measurements. However, RAM memory became an uncontrolled source of power utilization in §10.2 when I validated my proposed model. In fact, all the Hadoop processes that run in the servers consume significant RAM memory. This impacts more significantly the memory used by the cluster's master node, since it runs internal Hadoop processes (such as the NameNode or the JobTracker) whose memory requirement increases with the number of mappers and reducers. This cost is, therefore, paid only in `Nemesis`, the master node of the cluster, and not in `Survivor`, which explains the different accuracy of the model for the two servers. This error is particularly evident although small when WordCount is run, due to the fact that the required number of mappers for WordCount is larger than for PageRank and, therefore, the RAM required in `Nemesis` increases and so does the uncontrolled energy consumption.

10.4. Summary

In this chapter I have shown how to predict and optimize the energy consumed by an application via a concrete example using network activity plus PageRank computation in Hadoop. Moreover, I validated the accuracy of the model derived from the per component characterization by comparing the real energy consumed by two Hadoop applications - PageRank and WordCount - with the estimation from my model, achieving very accurate energy estimates with errors below 4.1% on average from the measured total energy consumption.

Chapter 11

Summary and Conclusions

In this thesis, I have carried out a detailed analysis of the energy saving in data and web hosting centers. I have focused on the energy optimization of the networking part of data centers and also I developped a model to estimate and optimize the energy consumption of the data center servers. Performance evaluation shows that my approaches provide substantial improvements over existing state-of-the-art solutions.

In the first part of this thesis, I outlined all the background information which is needed for the rest of the thesis. In particular, in Chapter 1 I provide data on current CPU technologies, VM techniques and energy consumptions of the various components of data center servers. Moreover, I highlight my main results and the contribution of this thesis explosing my list of publications. In Chapter 2 I explained IEEE Standard 802.3az which is the energy efficient alternative of Ethernet, namely Energy Efficient Ethernet (EEE), and its energy efficient enhancement packet coalescing. In a nutshell, EEE has been recently introduced to reduce the power consumed in LANs. Since then, researchers have proposed various traffic shaping techniques to leverage EEE in order to boost power saving. Packet coalescing is a promising mechanism which can be used on top of EEE to tradeoff power saving and packet delay. In Chapter 3 I gave a background on existing methods for resource provisioning in data centers and the most common assumptions/mistakes that research community makes regarding VM migration. Moreover, I challenged common evaluation practices employed in past VM consolidation studies, such as simulation and small testbeds, which fail to capture the fundamental properties of real systems. Specifically, I identified a series of over-simplifying assumptions regarding energy consumption and performance characteristics with respect to virtualized infrastructures. Therefore, more complex/complete models which study the energy consumed by a server are needed. To be consistent, these models have to be based on empirical values. However, I found that there is a lack of empirical works studying servers' energy behavior. In Chapter 4 I analyse the methodology which I follow in this thesis, from gathering real data traces (devices, topology, data capturing) and power measurements, to a break down analysis of the measured server components and the benchmarks used for each of the components. Finally, in Chapter 5 I gave the details of existing literature on EEE and on energy efficiency in data centers.

In the second part of this thesis, I have focused exclusively on EEE. In Chapters 6-8, I analyzed the interesting and special case of 1000Base-T EEE links, in which power saving operations are triggered only when links are inactive in both transmission directions. I am the first to provide an analytical model for EEE 1000Base-T which accounts for the bidirectional nature of LAN traffic. My model allows to compute the power saving achieved by EEE, with and without packet coalescing, by using a few significant traffic descriptors. Furthermore, I used real traffic traces to investigate on the performance of static as well as two families of dynamic coalescing schemes. My results showed that the first family of algorithms - NTCC - which adapts its coalescing parameters based on the event of timeouts and buffer fill-ups does not significantly outperform static coalescing in terms of power save and delay. The second family of algorithms - MBCC - adapts its coalescing timer according to the delay sensed by packets in the link and my evaluations showed that the energy saving benefit can be doubled with respect to static coalescing schemes and therefore with respect to NTCC.

Specifically, in Chapter 6, I have presented a model for bidirectional EEE links with static or no coalescing which can be used to accurately estimate power saving and packet delay over EEE gigabit links. This model is unique in the existing literature, in which only unidirectional EEE links are accounted for. Although this study can be extended to EEE links operating at different speeds, I focused on $1\ Gbps$ links since they are the most commonly adopted links in nowadays data centers. Moreover I have used sensitivity analysis to understand the impact of coalescing parameters, such as timer T_c and buffer size N_c, on the energy saving and the delay experienced over EEE links with coalescing. The analysis reveals that optimizing energy saving subject to delay constraints is possible by simply adapting T_c.

In Chapter 7 I proposed two families of dynamic algorithms. In the first family I adapt the coalescing parameters based on timeouts and buffer overflows during coalescing. Notably, I have shown that static coalescing algorithms, in which the coalescing queue size N_c and the coalescing timeout T_c are fixed, can achieve results as good as dynamic coalescing algorithms, in which either N_c and T_c can be dynamically adapted to the traffic characteristics. However in the second family of dynamic algorithms the energy saving that can be achieved is doubled while keeping the coalescing delay within specific bounds.

In Chapter 8 I performed a performance evaluation of my analytic model and my dynamic algorithms. In particular, I have modified the ns-3 simulator to implement (i) the EEE standard and (ii) static as well as dynamic coalescing algorithms, and thus validate my model. Furthermore, I have proposed an exhaustive performance evaluation on the impact of packet coalescing techniques over EEE power saving and delay performances. Specifically, I have tested EEE and coalescing algorithms by means of real packet traces I collected at the firewall interfaces of a large web hosting center in Madrid, Spain. Moreover, my model can be used to estimate the potential power saving vs. delay tradeoff of EEE with static or dynamic coalescing. This study has shown that the sole EEE standard (without coalescing) works fine under scarce traffic (1%). In contrast, as soon as the traffic exceeds a few percents of the link capacity, EEE needs to be endowed with packet coalescing to achieve significant power saving. Thanks to coalescing, significant economy can be achieved with link loads as high as 40-50%, while plain EEE would not allow to achieve detectable power saving with loads higher than a few percents of the link capacity.

In the third part of the thesis, i.e. Chapters 9- 10, I concentrate on the characterization of data center servers' energy consumption. Indeed, in order to obtain full benefit of energy efficient techniques proposed in the literature [56, 67], it is crucial to profile the utilization of the data center servers' components. Moreover, it is necessary to understand the energy consumption of servers and how it is affected by different load configurations. There is a large body of work on characterizing servers' energy consumption. However, the existing literature does not jointly consider phenomena like the irruption of multicore servers and dynamic voltage and frequency scaling (DVFS) [93], which are key to achieve scalability and flexibility in the architecture of a server. To address this problem, I designed an evaluation framework which incorporates more accurate models for data center systems and their available resources. In particular, I proposed a measurement-based power characterization methodology for servers, which accepts as input the load of individual hardware components and estimates the energy consumption for different server configurations. The integration of the two solutions, allowed me to achieve the envisioned goal of exploring the energy-performance trade-off in data centers. Finally, I proposed an accurate technique to estimate the energy consumption of cloud applications.

Specifically, in Chapter 9 I empirically characterized the power and energy consumed by different types of servers. In particular, in order to understand the behavior of their energy and power consumption, I performed measurements in different servers. In each of them, I exhaustively measured the power consumed by the CPU, the disk, and the network interface under different configurations, identifying the optimal operational levels. One interesting conclusion of this study was that the curve that defines the minimal CPU power as a function of the load is neither linear nor purely convex as has been previously assumed. Moreover, I found that the efficiency of the various server components can be maximized by tuning the CPU frequency and the number of active cores as a function of the system and network load, while the block size of I/O operations should be always maximized by applications.

In Chapter 10 I showed how to estimate the energy consumed by an application as a function of some simple parameters, like the CPU load, and the disk and network activity. Moreover, I validated the proposed approach by estimating the energy consumed by several *map-reduce Hadoop* computations. My model achieved very accurate energy estimates, below 4.1% on average from the measured total energy consumption.

The research performed in this dissertation resulted in four conference papers [8, 22, 25, 26], three journal articles [9, 24, 63], and one poster [23].

Appendices

Appendix A

Approximation accuracy for $E[\tau_A]$ in the model of § 6.2

In § 6.2, I have implicitly assumed that the "memory" of the coalescing operation is lost at the end of the first busy interval after entering state A. To be more rigorous, coalescing memory consists in packets cumulated during τ_c and T_W at *both* link edges, and so, for the properties of regular $M/G/1$ queues, it is lost only after the initial busy intervals in *either* link directions terminate. At that point, the probability to check a queue at random and find it busy is *exactly* ρ_i. In contrast, before that point, the probability to find a queue busy is *at least* ρ_i, due to the fact that packets coalesced during states C and W cumulate with fresh arrivals. Therefore, my approximation is *conservative* with respect to the time spent in state A.

Specifically, let me indicate with $\rho'_j \geq \rho_j$ the probability to find queue Q_j busy ad the end of $B_c^{(i)}$, $i \neq j$, to evaluate the difference between my approximation $\theta_A^{(i)}$ and the exact analytical value of the average:

$$\begin{aligned}
&E[\tau_A|_{W\to A}, Q_i] \\
&= E[B_c^{(i)}] + \rho'_j E[B^{(j)}] + \rho_i \rho'_j E[B^{(i)}] + \rho_i \rho_j \rho'_j E[B^{(j)}] + \dots \\
&= E[B_c^{(i)}] + \rho'_j \frac{E[B^{(j)}] + \rho_i E[B^{(i)}]}{1 - \rho_1 \rho_2} \geq \theta_A^{(i)},
\end{aligned} \tag{A.1}$$

in which I remark that $\theta_A^{(i)}$ is the approximation yielded by my model. Of course, the above lower bound can be used for the unconditional duration of state A:

$$\theta_A \triangleq \frac{\lambda_1 \theta_A^{(1)}}{\lambda_1 + \lambda_2} + \frac{\lambda_2 \theta_A^{(2)}}{\lambda_1 + \lambda_2} \leq E[\tau_A|_{W\to A}], \tag{A.2}$$

where θ_A is the approximation used in this paper.

With the above, it is clear that the more packets are coalesced—and the higher the loads—the higher the difference between my approximation for the time spent in state A and the value returned by an exact model.

Moreover, an upper bound for the time spent in state A can be computed as follows. Denote by $B_c^{(j,i)}$ the initial busy period of queue Q_j after a transition $W \to A$ with an arrival to Q_i triggering coalescing. State A lasts at least the maximum between $B_c^{(i)}$ and $B_c^{(j,i)}$, after which memory of coalescing vanishes, plus a random alternation of total duration ψ of busy periods of the two queues in steady state. For the average, using the relation $E[\max_i\{A_i\}] \leq \sum_i E[A_i]$ (valid for non-negative processes A_i), I have the following upper bound:

$$\begin{aligned} E[\tau_A|_{W\to A}, Q_i] &= E\left[\max\left\{B_c^{(i)}, B_c^{(j,i)}\right\} + \psi\right] \\ &\leq E\left[B_c^{(i)}\right] + E\left[B_c^{(j,i)}\right] + E[\psi], \end{aligned} \tag{A.3}$$

in which the average of $B_c^{(j,i)}$, which is similar to $B_c^{(i)}$, is given by the following expression:

$$E[B_c^{(j,i)}] = \frac{\lambda_j\left(E[\tau_C|_{W\to A}, Q_i] + T_W\right)}{\mu_j - \lambda_j},\ i \neq j. \tag{A.4}$$

Since ψ is an alternation of busy periods in steady state, which can start from either observing Q_1 or Q_2, its average can be upper bounded by the following expression, obtained as in the derivation of $\theta_A^{(i)}$ in Eq. (C.4):

$$\begin{aligned} E[\psi] &\leq \max\left\{\rho_1\frac{E[B^{(1)}]+\rho_2 E[B^{(2)}]}{1-\rho_1\rho_2}, \rho_2\frac{E[B^{(2)}]+\rho_1 E[B^{(1)}]}{1-\rho_1\rho_2}\right\} \\ &\leq \frac{\rho_1 E[B^{(1)}] + \rho_2 E[B^{(2)}]}{1-\rho_1\rho_2}. \end{aligned} \tag{A.5}$$

Therefore, a practical upper bound for the time spent in state A after a coalescing period triggered by queue Q_i is:

$$\begin{aligned} E[\tau_A|_{W\to A}, Q_i] &\leq E\left[B_c^{(i)}\right] + \frac{\lambda_j\left(E[\tau_C|_{W\to A}, Q_i] + T_W\right)}{\mu_j - \lambda_j} \\ &+ \frac{\rho_1 E[B^{(1)}]+\rho_2 E[B^{(2)}]}{1-\rho_1\rho_2} \\ &= \theta_A^{(i)} + \frac{\lambda_j\left(E[\tau_C|_{W\to A}, Q_i] + T_W\right)}{\mu_j - \lambda_j} + \frac{(\rho_i - \rho_1\rho_2)E[B^{(i)}]}{1-\rho_1\rho_2}, \end{aligned} \tag{A.6}$$

where all terms in the R.H.S. are positive due to the fact that $\rho_i \in [0,1)$, $i \in \{1,2\}$, under stable system conditions.

Note that my approximation $\theta_A^{(i)}$ approaches the upper bound (and hence, being $\theta_A^{(i)}$ a lower bound, it approaches the correct value) in extremely low load conditions. Instead, the upper bound becomes much greater than $\theta_A^{(i)}$ under high load conditions.

Finally, after some algebraic manipulation to remove the conditions on the queue that started the coalescing, I obtain the following expression for the upper bound:

$$E[\tau_A|_{W\to A}] \leq \frac{1}{\lambda_1+\lambda_2}\left(\frac{\lambda_1}{\mu_1-\lambda_1}+\frac{\lambda_2}{\mu_2-\lambda_2}\right) + \frac{\lambda_1\left(E[\tau_C|_{W\to A}]+T_W\right)}{\mu_1-\lambda_1}+\frac{\lambda_2\left(E[\tau_C|_{W\to A}]+T_W\right)}{\mu_2-\lambda_2} + \frac{\rho_1\frac{2\mu_1-\lambda_1}{2(\mu_1-\lambda_1)^2}+\rho_2\frac{2\mu_2-\lambda_2}{2(\mu_2-\lambda_2)^2}}{1-\rho_1\rho_2}. \tag{A.7}$$

Note that, the probability to stay in state LPI, which is the power saving factor $\eta_{LPI} = \eta_L + \eta_C$, is inversely proportional to $E[\tau_A]$. Therefore, the lower bound of $E[\tau_A]$ corresponds to the upper bound of η_{LPI} and the upper bound of $E[\tau_A]$ corresponds to the lower bound of η_{LPI}. As concerns the delay, as it is easy to compute, the partial derivative of $D^{(i)}$ with respect to $E[\tau_A]$ is a constant divided by a function of $E[\tau_A]$ and other parameters. However, since such function is always positive, the delay is monotone w.r.t. $E[\tau_A]$, whose bounds can be therefore straightforwardly used to compute bounds for $D^{(i)}$ as well.

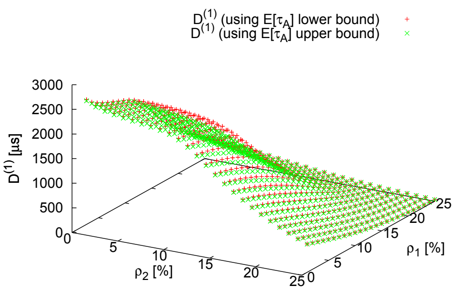
The analysis presented in this appendix reveals that my approximation in the calculation of $E[\tau_A|_{W\to A}]$ can be rough only if loads are high. In that case, θ_A is a very conservative approximation of the average. However, high loads make the transition $W \to A$ unlikely to happen since the weight of $E[\tau_A|_{W\to A}]$ in the average expressed in Eq. (6.41) is $P_S = e^{-(\lambda_1+\lambda_2)T_S}$. Therefore, potentially large errors on $E[\tau_A|_{W\to A}]$ are discounted with an exponential function of the load and do not significantly affect $E[\tau_A]$. In conclusion, the approximation on $E[\tau_A]$ I have introduced is impacted by two contrasting factors: (i) the uncertainty on $E[\tau_A|_{W\to A}, Q_i]$, which grows with the load, and (ii) the probability to enter state L, and hence have a transition $W \to A$, which exponentially decreases with the load. Thereby, although there is practically no uncertainty at very low and high loads, my approximation might yield inaccurate values for $E[\tau_A]$ at low-medium loads (e.g., loads below 10-15% in either link direction, with only large packets, would yield $P_S \leq 0.1$). Next I show by means of numerical results that the uncertainty introduced in the model by the approximation on $E[\tau_A]$ is small under a large range of realistic operational parameters.

In Figures A.1-A.6 I plot the upper and the lower bound for delays and η_{LPI} for different combinations of (N_c, T_c) and average packet sizes. Here I simulate loads increasing by 1% steps. Moreover Figures A.1-A.3 correspond to combination ($N_c = 50$ packets, $T_c = 5\mu$s) while Figures A.4-A.6 correspond to combination ($N_c = 100$ packets, $T_c = 20\mu$s). Finally, Figures A.1 and A.4 correspond to traffic with big packets (1500 bytes) in both directions, Figures A.2 and A.5 correspond to traffic with small packets (150 bytes) in both directions, and Figures A.3 and A.6 correspond to traffic with big packets in direction 1 and small packets in the direction 2. In all plots, I report loads below 25% because delays and η_{LPI} converge to their asymptotic values (practically to 0) very quickly. In all cases I observe little differences between the values computed based on lower and upper bounds of $E[\tau_A]$. As expected, there is no observable error for very

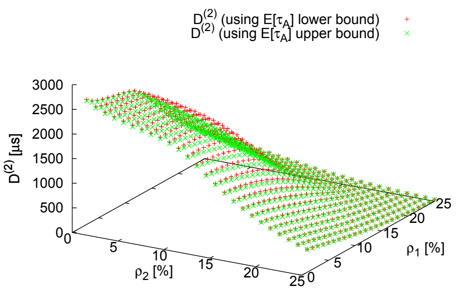
Table A.1: Model uncertainty computed on η_{LPI} and average delays

ρ_1	ρ_2	$E[S_p^{(1)}]$	$E[S_p^{(2)}]$	λ_1	λ_2	T_c	N_c	η_{LPI} [%]		$D^{(1)}$ [ms]		$D^{(2)}$ [ms]	
[%]	[%]	[bytes]	[bytes]	[pkts/s]	[pkts/s]	[ms]	[pkts]	lower bound	upper bound	lower bound	upper bound	lower bound	upper bound
						5	50	96.75	96.83	3.057	3.060	3.061	3.064
						5	100	96.75	96.84	3.057	3.060	3.062	3.064
0.20	0.06	802	281	310	268	10	50	98.00	98.11	5.641	5.647	5.649	5.655
						10	100	98.00	98.11	5.641	5.647	5.649	5.655
						20	100	98.79	98.91	10.687	10.699	10.702	10.715
						5	50	0.88	0.88	0.007	0.007	0.011	0.011
						5	100	1.74	1.74	0.026	0.027	0.039	0.039
0.71	39.69	67	1512	13187	32815	10	50	0.88	0.88	0.007	0.007	0.011	0.011
						10	100	1.74	1.75	0.026	0.027	0.039	0.039
						20	100	1.74	1.75	0.027	0.027	0.039	0.039
						5	50	4.56	4.59	0.046	0.047	0.060	0.061
						5	100	8.55	8.67	0.173	0.176	0.223	0.226
0.87	29.64	83	1477	13048	25084	10	50	4.56	4.59	0.046	0.047	0.060	0.061
						10	100	8.56	8.67	0.173	0.176	0.224	0.227
						20	100	8.55	8.67	0.173	0.176	0.224	0.227
						5	50	0.03	0.03	$< 1\mu$s	$< 1\mu$s	0.004	0.004
						5	100	0.06	0.06	$< 1\mu$s	$< 1\mu$s	0.005	0.005
0.98	53.66	69	1500	17769	44719	10	50	0.03	0.03	$< 1\mu$s	$< 1\mu$s	0.004	0.004
						10	100	0.06	0.06	$< 1\mu$s	$< 1\mu$s	0.005	0.005
						20	100	0.06	0.06	$< 1\mu$s	$< 1\mu$s	0.005	0.005
						5	50	89.45	90.89	2.323	2.361	2.401	2.440
						5	100	89.45	90.89	2.323	2.361	2.401	2.440
3.72	0.37	1180	165	3938	2778	10	50	92.52	94.08	4.705	4.784	4.862	4.944
						10	100	92.54	94.10	4.726	4.807	4.884	4.967
						20	100	94.20	95.82	9.534	9.698	9.852	10.022
						5	50	86.22	88.09	2.327	2.377	2.226	2.274
						5	100	86.22	88.09	2.327	2.377	2.226	2.274
5.06	0.50	1170	165	5408	3809	10	50	89.62	91.67	4.079	4.172	4.262	4.360
						10	100	90.12	92.18	4.591	4.697	4.800	4.909
						20	100	92.02	94.19	8.468	8.667	8.852	9.060
						5	50	26.75	27.40	0.355	0.364	0.436	0.446
						5	100	38.46	39.81	0.955	0.988	1.171	1.212
0.40	23.11	66	1511	7517	19116	10	50	26.75	27.40	0.355	0.364	0.436	0.447
						10	100	39.66	41.10	1.048	1.086	1.285	1.331
						20	100	39.66	41.10	1.048	1.086	1.285	1.332
						5	50	29.39	30.16	0.433	0.445	0.505	0.519
						5	100	40.32	41.76	1.028	1.066	1.200	1.243
1.14	17.93	148	1294	9639	17320	10	50	29.39	30.16	0.433	0.445	0.505	0.519
						10	100	43.18	44.84	1.267	1.316	1.478	1.535
						20	100	43.18	44.84	1.268	1.317	1.478	1.535

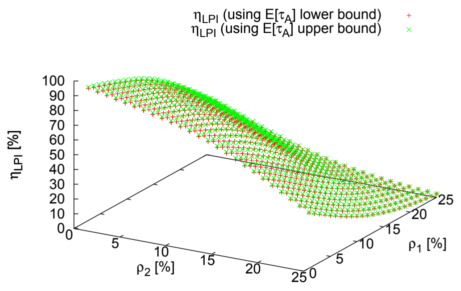
low load combinations and for medium-high loads, for which the coefficient $P_S \ll 1$ and so the uncertainty disappears. The figures show that some minor differences can be noticed for the case of big packets (1500-byte packets in both link directions) for loads comprised between 5% to 15%. For what concerns the cases previously presented in Table 8.1, which reports a trace-driven evaluation of EEE with coalescing, the model uncertainty due to my approximation is negligible, as shown in Table A.1.



(a) Upper and lower bound for Delay in direction 1.

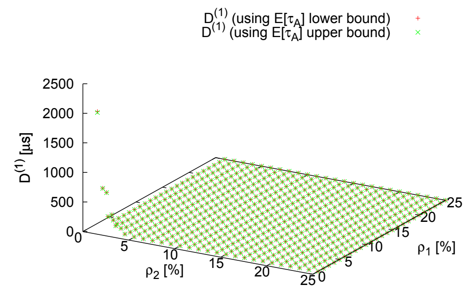


(b) Upper and lower bound for Delay in direction 2.

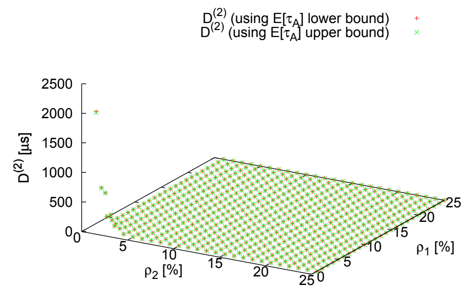


(c) Upper and lower bound for η_{LPI}.

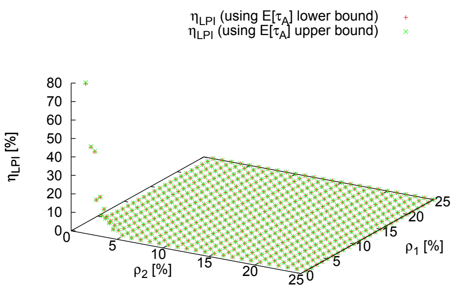
Figure A.1: 1500-byte packets in both directions and $N_c = 50$ packets, $T_c = 5$ ms.



(a) Upper and lower bound for Delay in direction 1.

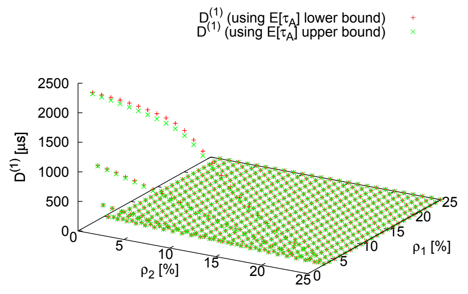


(b) Upper and lower bound for Delay in direction 2.

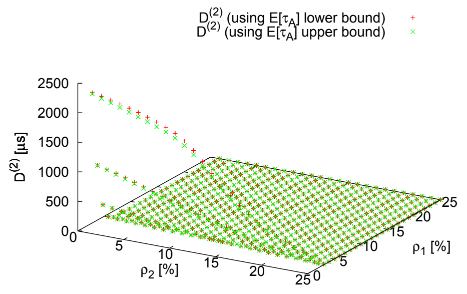


(c) Upper and lower bound for η_{LPI}.

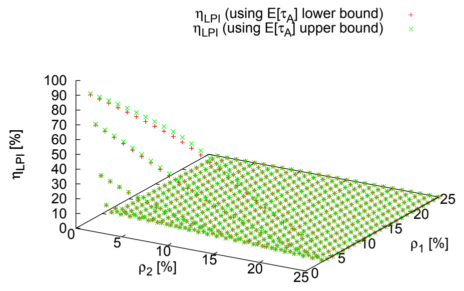
Figure A.2: 150-byte packets in both directions and $N_c = 50$ packets, $T_c = 5$ ms.



(a) Upper and lower bound for Delay in direction 1.

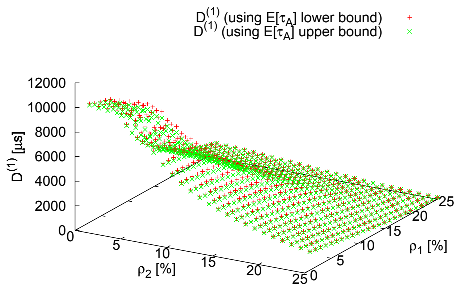


(b) Upper and lower bound for Delay in direction 2.

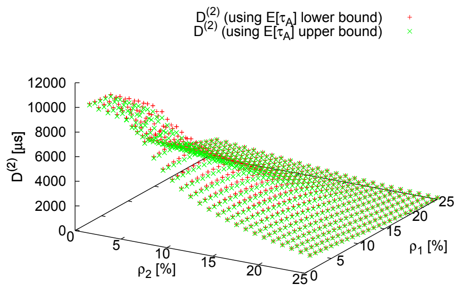


(c) Upper and lower bound for η_{LPI}.

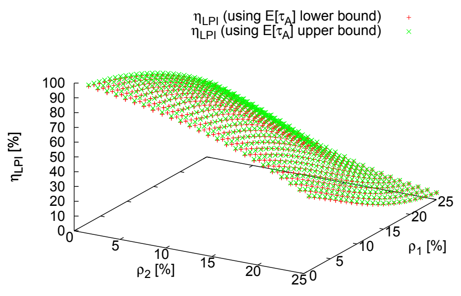
Figure A.3: 1500-byte packets in direction 1 and 150-byte packets in direction 2 and $N_c = 50$ packets, $T_c = 5$ ms.



(a) Upper and lower bound for Delay in direction 1.

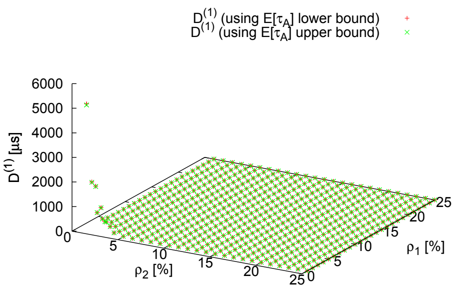


(b) Upper and lower bound for Delay in direction 2.

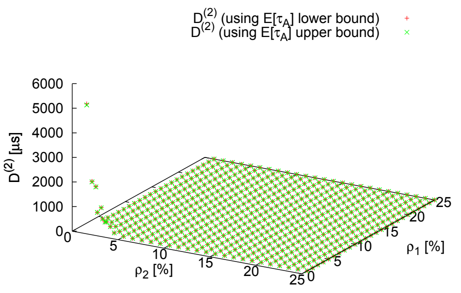


(c) Upper and lower bound for η_{LPI}.

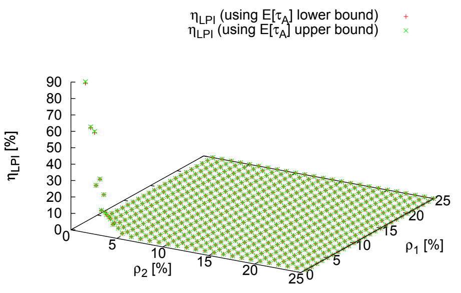
Figure A.4: 1500-byte packets in both directions and $N_c = 100$ packets, $T_c = 20$ ms.



(a) Upper and lower bound for Delay in direction 1.

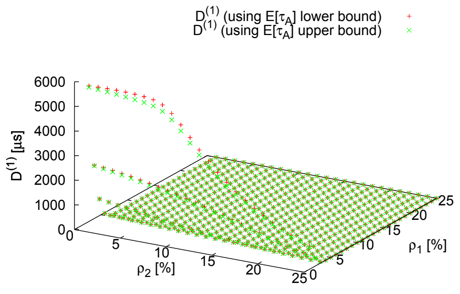


(b) Upper and lower bound for Delay in direction 2.

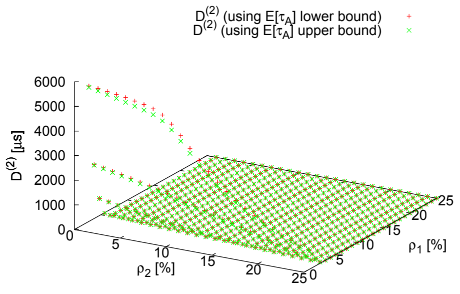


(c) Upper and lower bound for η_{LPI}.

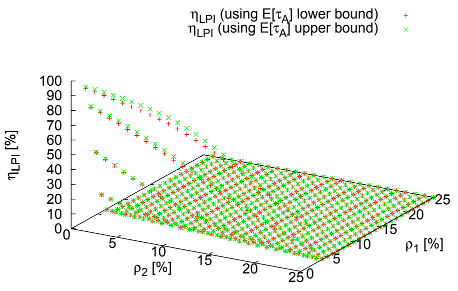
Figure A.5: 150-byte packets in both directions and $N_c = 100$ packets, $T_c = 20$ ms.



(a) Upper and lower bound for Delay in direction 1.



(b) Upper and lower bound for Delay in direction 2.



(c) Upper and lower bound for η_{LPI}.

Figure A.6: 1500-byte packets in direction 1 and 150-byte packets in direction 2 and $N_c = 100$ packets, $T_c = 20$ ms.

Appendix B

Computation of the duration of state C in the model of § 6.2

The duration of state C is a r.v. $Y = \min\{X_1, X_2, T_c\}$, i.e., the minimum between T_c and two other independent r.v.'s X_1 and X_2 modeling the time needed for $N_c - 1$ arrivals for the queue that triggered coalescing and N_c arrivals for the other queue. Therefore, X_1 and X_2 are Erlang random variables. Assuming that Q_1 triggers transition $L \to C$, then $X_1 \sim Erl(\lambda_1, N_c - 1)$ and $X_2 \sim Erl(\lambda_2, N_c)$. The cumulative distribution function (CDF) of Y can be written as a function of the CDFs of X_1 and X_2 as follows:

$$F_Y(t) = 1 - [1 - F_{X_1}(t)]\,[1 - F_{X_2}(t)]\,u\,(T_c - t)\,,\; t \geq 0, \tag{B.1}$$

where $u(\cdot)$ is the unit step function, which guarantees a jump to 1 at time $t = T_c$ in the CDF of Y (the state duration cannot exceed the timeout). From Eq. (6.33) and from the definition of Erlang distribution, $F_{X_1}(t) = F^{(1)}_{N_c - 1}(t)$, and $F_{X_2}(t) = F^{(2)}_{N_c}(t)$, which yields the following expression:

$$\begin{aligned} E[\tau_C|_{W \to A}, Q_1] = E[Y] &= \int_0^\infty t dF_Y(t) \\ &= \int_0^{T_c} \frac{(\lambda_1 t)^{N_c - 1} e^{-\lambda_1 t}}{(N_c - 2)!} \cdot \left(1 - F^{(2)}_{N_c}(t)\right) dt \\ &+ \int_0^{T_c} \frac{(\lambda_2 t)^{N_c} e^{-\lambda_2 t}}{(N_c - 1)!} \left(1 - F^{(1)}_{N_c - 1}(t)\right) dt \\ &+ \left(1 - F^{(1)}_{N_c - 1}(T_c)\right) \left(1 - F^{(2)}_{N_c}(T_c)\right) T_c. \end{aligned} \tag{B.2}$$

The first term in the above expression corresponds to Q_1 finishing coalescing first, the second term corresponds to Q_2 finishing first, and the last term is due to timeouts. The above expression could be also re-written in explicit form (no integrals and cumulative functions), although not compactly. The expression for $E[\tau_c|Q_2]$ has the same structure and the average duration of state

C is eventually computed as $E[\tau_C] = \frac{\lambda_1}{\lambda_1+\lambda_2} E[\tau_C|_{W\to A}, Q_1] + \frac{\lambda_2}{\lambda_1+\lambda_2} E[\tau_C|_{W\to A}, Q_2]$, from which I obtain Eq. (6.34).

Appendix C

Computation of the duration of state A in the model of § 6.2

Observe that, although the occurrence of a busy interval in a link direction depends on the activity of the other link direction, the evolution of each individual busy interval only depends on the arrival processes. In particular, once a busy interval starts, it evolves like in a legacy $M/G/1$ queue with initial state given by the queue backlog at the beginning of the busy interval [54].

Entering state A from state W. Assume, with no loss of generality, that Q_i triggers the coalescing. I denote with $B_c^{(i)}$, $i \in \{1, 2\}$, the first busy period seen at Q_i only. Hence, on average I have the following duration for the busy period:

$$E[B_c^{(i)}] = \frac{E\left[Z_c^{(i)}\right] E\left[S_p^{(i)}\right]/R}{1-\rho_i} = \frac{E\left[Z_c^{(i)}\right]}{\mu_i - \lambda_i}, \tag{C.1}$$

where $E[S_p^{(i)}]/R$ is the mean packet service time, and $Z_c^{(i)}$ is the queue size at the beginning of the busy period. The expression is equivalent to Eq. (6.38) because Q_i is the queue that received the packet that triggered the transition $L \to C$, so that $E\left[Z_c^{(i)}\right] = 1 + \lambda_i(E[\tau_C|_{W\to A}, Q_i] + T_W)$.

At the end of $B_c^{(i)}$, Q_i is empty with probability 1, while I assume that the other queue Q_j is non-empty with a probability that is equal to its utilization factor ρ_j. This assumption represents an approximation with little impact on the performance of the model, as it is discussed in Appendix A. Then I can observe the followup in the evolution of the system from the viewpoint of the other queue, although this time the busy period $B^{(i)}$ depends on the backlog of an $M/G/1$ queue seen at a random observation instant, given that the queue is serving a packet. Therefore, using the Pollaczek-Khinchin mean formula [16], and using the condition that queue has to be non-empty at the observation instant, I have:

$$E\left[Z^{(i)}\right] = \frac{1}{\rho_i}\cdot\left(\frac{\lambda_i\, E\left[S_p^{(i)}\right]}{R} + \frac{\lambda_i^2\, E\left[S_p^{(i)^2}\right]/R^2}{2(1-\rho_i)}\right)$$
$$= 1 + \frac{\rho_i}{2(1-\rho_i)}; \tag{C.2}$$
$$E\left[B^{(i)}\right] = \frac{E\left[Z^{(i)}\right]}{\mu_i-\lambda_i} = \frac{2\mu_i-\lambda_i}{2(\mu_i-\lambda_i)^2}; \tag{C.3}$$

which is equivalent to Eq. (6.39).

Busy periods $B^{(1)}$ and $B^{(2)}$ alternate until the observation of the queue status at the end of a busy period reveals that both queues are empty. At that point I have a transition $A \to S$. The duration of state A, assuming that Q_i triggered coalescing, is then computed as follows:

$$E[\tau_A|_{W\to A}, Q_i]$$
$$\simeq E[B_c^{(i)}] + \rho_j E[B^{(j)}] + \rho_i\rho_j E[B^{(i)}] + \rho_i\rho_j^2 E[B^{(j)}] + \ldots$$
$$= E[B_c^{(i)}] + \rho_j \frac{E[B^{(j)}] + \rho_i E[B^{(i)}]}{1-\rho_1\rho_2} \triangleq \theta_A^{(i)}. \tag{C.4}$$

In the above expression, I have defined $\theta_A^{(i)}$ as the result of my approximation.

Thereby, since queue Q_i triggers coalescing with probability $\frac{\lambda_i}{\lambda_1+\lambda_2}$, from the above I obtain the expression for $E[\tau_A|_{W\to A}]$ presented in Eq. (6.37).

Entering state A from state S. In this case, there is an arrival in state S, which is immediately served. The first busy period is then $B_s^{(i)}$ on either Q_1 or Q_2, with initial backlog equal to 1. For the rest, the computation of $B_s^{(i)}$ is analogue to the one of $B_c^{(i)}$. Moreover, due to the independence of the considered Poisson arrival processes, the probability of restarting the service with a busy period in direction i, $B_s^{(i)}$, is $\frac{\lambda_i}{\lambda_1+\lambda_2}$. After that, the following alternating busy intervals are exactly like in the case of busy periods after the transition $W \to A$ discussed above, i.e., I have intervals $B^{(1)}$ and $B^{(2)}$. Thereby, the computation of $E[\tau_A|_{S\to A}]$ is similar to the one of $E[\tau_A|_{W\to A}]$, and yields the result of Eq. (6.40).

Average time spent in state A. So far I have shown how to derive $E[\tau_A|_{W\to A}]$ and $E[\tau_A|_{W\to S}]$. The result presented in Eq. (6.41) is therefore obtained by averaging these results with the probability to have or not arrivals in state S, that is: $E[\tau_A] = E[\tau_A|_{W\to A}]P_S + E[\tau_A|_{W\to S}](1-P_S)$.

References

[1] NS-3 website: `http://www.nsnam.org/`.

[2] NetOptics website: `http://www.netoptics.com/products/network-taps`.

[3] Endace website: `http://www.endace.com`.

[4] A. Beloglazov *et al.*. Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. *Future Generation Computer Systems*, 28(5), 2012.

[5] A. Gandhi *et al.*. Optimal power allocation in server farms. In *SIGMETRICS Performance Evaluation Review*. ACM, 2009.

[6] A. Verma *et al.*. pMapper: power and migration cost aware application placement in virtualized systems. In *Middleware*. Springer, 2008.

[7] M. Andrews, S. Antonakopoulos, and L. Zhang. Minimum-cost network design with (dis)economies of scale. In *IEEE FOCS*, pages 585–592, 2010.

[8] J. Arjona Aroca, A. Chatzipapas, A. Fernández Anta, and V. Mancuso. A measurement-based analysis of the energy consumption of data center servers. In *Proceedings of ACM e-Energy '14*, pages 63–74. ACM, June 2014.

[9] J. Arjona Aroca, A. Chatzipapas, A. Fernández Anta, and V. Mancuso. A measurement-based characterization of the energy consumption in data center servers. *IEEE Journal on Selected Areas in Communications*, 33(12):2863–2877, Dec 2015.

[10] J. Baliga, R. W. A. Ayre, K. Hinton, and R. S. Tucker. Green cloud computing: Balancing energy in processing, storage, and transport. *Proceedings of the IEEE*, 99(1):149–167, 2011.

[11] S. Bapat. The Future of Data Centers (... and the Stuff That Goes In Them). In *1st Berkeley Symposium on Energy Efficient Electronic Systems*, June 2009.

[12] R. Basmadjian, N. Ali, F. Niedermeier, H. de Meer, and G. Giuliani. A methodology to predict the power consumption of servers in data centres. In *ACM e-Energy*, pages 1–10, 2011.

[13] R. Basmadjian and H. de Meer. Evaluating and modeling power consumption of multi-core processors. In *IEEE e-Energy*, pages 1–10, 2012.

[14] A. Beloglazov and R. Buyya. Energy efficient resource management in virtualized cloud data centers. In *MGC*. IEEE, 2010.

[15] T. Benson, A. Anand, A. Akella, and M. Zhang. Understanding data center traffic characteristics. *ACM SIGCOMM Computer Communication Review*, 40(1):92–99, January 2010.

[16] D. Bertsekas and R Gallager. *Data Network*. Prentice Hall, 2 edition, 1992.

[17] R. Bolla, R. Bruschi, A. Carrega, F. Davoli, and P. Lago. A Closed-Form Model for the IEEE 802.3az Network and Power Performance. *IEEE Journal on Selected Areas in Communications*, 32(1):16–27, 2014.

[18] A. Brihi and W. Dargie. Dynamic voltage and frequency scaling in multimedia servers. In *IEEE AINA*, 2013.

[19] C. Pettey. Gartner estimates for ICT industry CO_2 emissions. `http://goo.gl/4KuOAi`, 2007.

[20] P Castagna. Having fun with pagerank and mapreduce. *Hadoop User Group UK talk. Available: http://static.last.fm/johan/huguk-20090414/paolo_castagna-pagerank.pdf*, 2009.

[21] J. S. Chase, D. C. Anderson, P. N. Thakar, A. M. Vahdat, and R. P. Doyle. Managing energy and server resources in hosting centers. *ACM SIGOPS Operating Systems Review*, 35(5):103–116, 2001.

[22] A. Chatzipapas and V. Mancuso. Modelling and real-trace-based evaluation of static and dynamic coalescing for energy efficient ethernet. In *Proceedings of ACM e-Energy '13*, pages 161–172, May 2013.

[23] A. Chatzipapas and V. Mancuso. Improving the energy benefit for 802.3az using dynamic coalescing techniques. In *Distributed Computing Systems (ICDCS), 2015 IEEE 35th International Conference on*, pages 728–729, June 2015.

[24] A. Chatzipapas and V. Mancuso. An m/g/1 model for gigabit energy efficient ethernet links with coalescing and real-trace-based evaluation. *IEEE/ACM Transactions on Networking*, PP(99):1–1, Sep 2015.

[25] A. Chatzipapas and V. Mancuso. Measurement-based coalescing control for 802.3az. In *2016 IFIP Networking Conference (IFIP Networking) and Workshops*, pages 270–278, May 2016.

[26] A. Chatzipapas, D. Pediaditakis, C. Rotsos, V. Mancuso, J. Crowcroft, and A. W. Moore. Challenge: Resolving data center power bill disputes: The energy-performance trade-offs of consolidation. In *Proceedings of the 2015 ACM Sixth International Conference on Future Energy Systems*, e-Energy '15, pages 89–94, New York, NY, USA, 2015. ACM.

[27] D. M. Chiu and R. Jain. Analysis of the increase and decrease algorithms for congestion avoidance in computer networks. *Computer Networks and ISDN systems*, 17(1):1–14, 1989.

[28] B. Y. Choi, S. Moon, Z.-L. Zhang, K. Papagiannaki, and C. Diot. Analysis of point-to-point packet delay in an operational network. *Computer networks*, 51(13):3812–3827, September 2007.

[29] K. Christensen, P. Reviriego, B. Nordman, M. Bennett, M. Mostowfi, and J. A. Maestro. IEEE 802.3az: The Road to Energy Efficient Ethernet. *IEEE Communications Magazine*, 48(11):50–56, November 2010.

[30] D. Abts *et al.*. Energy proportional datacenter networks. In *SIGARCH Computer Architecture News*, volume 38. ACM, 2010.

[31] D. Kliazovich *et al.*. DENS: Data center energy-efficient network-aware scheduling. In *GreenCom, CPSCom, IEEE/ACM*, 2010.

[32] D. Meisner *et al.*. PowerNap: eliminating server idle power. *SIGARCH Comp. Architecture News*, 2009.

[33] D. Economou, S. Rivoire, C. Kozyrakis, and P. Ranganathan. Full-system power analysis and modeling for server environments. In *Proceedings of Workshop on Modeling, Benchmarking, and Simulation*, pages 70–77, 2006.

[34] European Commission - (DG INSFO). Study of the impacts of ICT on energy efficiency, September 2008.

[35] F. Hermenier *et al.*. Entropy: a consolidation manager for clusters. In *VEE*. ACM, 2009.

[36] G. Chen *et al.*. Energy-aware server provisioning and load dispatching for connection-intensive Internet services. In *NSDI*. USENIX, 2008.

[37] G. Wang and T. S. E. Ng. The impact of virtualization on network performance of amazon ec2 data center. In *INFOCOM*. IEEE, 2010.

[38] A. Garcia-Saavedra, P. Serrano, A. Banchs, and G. Bianchi. Energy consumption anatomy of 802.11 devices and its implication on modeling and design. In *ACM CoNEXT*, pages 169–180, 2012.

[39] R. Ge, X. Feng, S. Song, H.-C. Chang, D. Li, and K. W. Cameron. Powerpack: Energy profiling and analysis of high-performance systems and applications. *IEEE TPDS*, 21(5):658–671, 2010.

[40] C. Gunaratne, K. Christensen, B. Nordman, and S. Suen. Reducing the Energy Consumption of Ethernet with Adaptive Link Rate (ALR). *Computers, IEEE Transactions on*, 57(4):448–461, April 2008.

[41] H. Goudarzi *et al.*. SLA-based optimization of power and migration cost in cloud computing. In *CCGrid*. IEEE, 2012.

[42] S. Herrería Alonso, M. Rodríguez Pérez, M. Fernández Veiga, and C. López García. How efficient is energy-efficient ethernet? In *Proceedings of ICUMT*, October 2011.

[43] S. Herrería Alonso, M. Rodríguez Pérez, M. Fernández Veiga, and C. López García. A GI/G/1 Model for 10Gb/s Energy Efficient Ethernet Links. *IEEE Transactions on Communications*, 60(11):3386–3395, November 2012.

[44] S. Herrería Alonso, M. Rodríguez Pérez, M. Fernández Veiga, and C. López García. Bounded energy consumption with dynamic packet coalescing. In *Proceedings of IEEE NOC*, pages 1–5, June 2012.

[45] S. Herrería Alonso, M. Rodríguez Pérez, M. Fernández Veiga, and C. López García. Optimal configuration of Energy-Efficient Ethernet. *Computer Networks*, 56(10):2456–2467, 2012.

[46] D. P. Heyman. The T-policy for the M/G/1 queue. *Management Science*, 23(7):775–778, 1977.

[47] IEEE Std. 802.3az. Energy Efficient Ethernet, 2010.

[48] S. Irani, S. Shukla, and R. Gupta. Algorithms for power savings. *ACM Trans. Algorithms*, 3(4), November 2007.

[49] J. Dean and L. A. Barroso. The tail at scale. *Commun. ACM*, 56(2), February 2013.

[50] J. Xu and J. AB Fortes. Multi-objective virtual machine placement in virtualized data center environments. In *GreenCom*. IEEE, 2010.

[51] V. Jacobson. Congestion avoidance and control. *ACM SIGCOMM Computer Communication Review*, 18(4):314–329, August 1988.

[52] T. Karagiannis, M. Molle, M. Faloutsos, and A. Broido. A nonstationary poisson view of internet traffic. In *Proceedings of IEEE INFOCOM*, volume 3, pages 1558–1569. IEEE, 2004.

[53] K. J. Kim, S. Jin, N. Tian, and B. D. Choi. Mathematical Analysis of Burst Transmission Scheme for IEEE 802.3Az Energy Efficient Ethernet. *Performance Evaluation*, 70(5):350–363, May 2013.

[54] L. Kleinrock. *Queueing Systems: Theory*, volume 1. J. Wiley & Sons, 1975.

[55] B. Krishnan, H. Amur, A. Gavrilovska, and K. Schwan. VM power metering: feasibility and challenges. *ACM SIGMETRICS Performance Evaluation Review*, 38(3):56–60, 2011.

[56] D. Kusic, J. O. Kephart, J. E. Hanson, N. Kandasamy, and G. Jiang. Power and performance management of virtualized computing environments via lookahead control. *Cluster computing*, 12(1):1–15, 2009.

[57] L. A. Barroso *et al.*. The case for energy-proportional computing. *IEEE computer*, 2007.

[58] D. Larrabeiti, P. Reviriego, J. A. Hernández, J. A. Maestro, and M. Uruena. Towards an Energy Efficient 10 Gb/s optical Ethernet: Performance analysis and viability. *Optical Switching and Networking*, 8(3):131–138, March 2011.

[59] H. Levy and L. Kleinrock. A queue with starter and a queue with vacations: Delay analysis by decomposition. *Operations Research*, 34(3):426–436, 1986.

[60] A. W. Lewis, S. Ghosh, and N.-F. Tzeng. Run-time energy consumption estimation based on workload in server systems. *HotPower'08*, pages 17–21, 2008.

[61] C. Liu, J. Huang, Q. Cao, S. Wan, and C. Xie. Evaluating energy and performance for server-class hardware configurations. In *IEEE NAS*, pages 339–347, 2011.

[62] F. Malandrino, M. Kurant, A. Markopoulou, C. Westphal, and U. C. Kozat. Proactive Seeding for Information Cascades in Cellular Networks. In *Proceedings of IEEE INFOCOM'12*, March 2012.

[63] V. Mancuso and A. Chatzipapas. On IEEE 802.3az Energy Efficiency in Web Hosting Centers. *IEEE Communications Letters*, 16(11):1880–1883, November 2012.

[64] Marco Ajmone Marsan, Antonio Fernández Anta, Vincenzo Mancuso, Balaji Rengarajan, Pedro Reviriego Vasallo, and Gianluca Rizzo. A simple analytical model for energy efficient ethernet. *IEEE Communications Letters*, 15(7):773–775, 2011.

[65] M. Mishra and A. Sahoo. On theory of vm placement: Anomalies in existing methodologies and their mitigation using a novel vector based approach. In *Proceedings of the 2011 IEEE 4th International Conference on Cloud Computing*, CLOUD '11, pages 275–282, Washington, DC, USA, 2011. IEEE Computer Society.

[66] A. Miyoshi, C. Lefurgy, E. Van Hensbergen, R. Rajamony, and R. Rajkumar. Critical power slope: understanding the runtime effects of frequency scaling. In *ACM ICS'02*, pages 35–44, 2002.

[67] J. Moore, J. Chase, P. Ranganathan, and R. Sharma. Making scheduling "cool": Temperature-aware workload placement in data centers. In *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, ATEC '05, pages 61–75, Berkeley, CA, USA, 2005. USENIX Association.

[68] M. Mostowfi and K. Christensen. Saving Energy in LAN Switches: New Methods of Packet Coalescing for Energy Efficient Ethernet. In *Proceedings of IGCC*, July 2011.

[69] M. Mostowfi and K. Christensen. An Energy-Delay Model for a packet coalescer. In *Proceedings of IEEE Southeastcon*, March 2012.

[70] N. Bobroff *et al.*. Dynamic placement of VM for managing SLA violations. In *IM*. IEEE, 2007.

[71] L. Nonde, T. E. H. El-Gorashi, and J. M. H. Elmirghani. Energy efficient virtual network embedding for cloud networks. *Journal on Lightwave Technology*, 33(9):1828–1849, 2015.

[72] B. Nordman. EEE Savings Estimates. IEEE 802.3 Energy Efficient Ethernet Study Group, May 2007.

[73] P. Delforge. America's data centers consuming and wasting growing amounts of energy. `http://goo.gl/HOLLBx`, 2014.

[74] P. Patel and D. Bansal *et al.*. Ananta: Cloud scale load balancing. In *ACM SIGCOMM CCR*, 2013.

[75] D. Pavlov, J. Soeurt, P. Grosso, Z. Zhao, K. van der Veldt, H. Zhu, and C. de Laat. Towards energy efficient data intensive computing using ieee 802.3 az. In *High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion:*, pages 806–810. IEEE, 2012.

[76] R. Nathuji *et al.*. VPM tokens: virtual machine-aware power budgeting in datacenters. In *Cluster comp.* Springer, 2009.

[77] R. Wang *et al.*. Openflow-based server load balancing gone wild. In *Hot-ICE*. USENIX, 2011.

[78] R. Raghavendra, P. Ranganathan, V. Talwar, Z. Wang, and X. Zhu. No power struggles: Coordinated multi-level power management for the data center. *ACM SIGARCH Computer Architecture News*, 36(1):48–59, 2008.

[79] P. Reviriego, K. Christensen, J. Rabanillo, and J. A. Maestro. Initial Evaluation of Energy Efficient Ethernet. *IEEE Communications Letters*, 15(5):578–580, May 2011.

[80] P. Reviriego, K. Christensen, and A. Sánchez-Macian. Using Coordinated Transmission with Energy Efficient Ethernet. In *Proceedings of IFIP Networking*, May 2011.

[81] P. Reviriego, J. A. Hernández, D. Larrabeiti, and J. A. Maestro. Performance Evaluation of Energy Efficient Ethernet. *IEEE Communications Letters*, 13(9):697–699, September 2009.

[82] P. Reviriego, J. A. Maestro, J. A. Hernández, and D. Larrabeiti. Burst Transmission for Energy Efficient Ethernet. *IEEE Computer Society*, 14(4):50–57, July 2010.

[83] P. Reviriego, V. Sivaraman, Z. Zhao, J.A. Maestro, A. Vishwanath, A. Sánchez-Macian, and C. Russell. An energy consumption model for energy efficient ethernet switches. In *High Performance Computing and Simulation (HPCS), 2012 International Conference on*, pages 98–104, jul 2012.

[84] S. Lee *et al.*. Validating heuristics for virtual machines consolidation. *Microsoft Research TR*, 2011.

[85] R. Sohan, A. Rice, A.W. Moore, and K. Mansley. Characterizing 10 Gbps Network Interface Energy Consumption. In *Proceedings of IEEE LCN 2010*, October 2010.

[86] S. Srikantaiah, A. Kansal, and F. Zhao. Energy aware consolidation for cloud computing. In *Proceedings of the 2008 conference on Power aware computing and systems*, volume 10. San Diego, California, 2008.

[87] E. Le Sueur and G. Heiser. Dynamic voltage and frequency scaling: The laws of diminishing returns. In *Proceedings of 2010 Usenix HotPower*, pages 1–8, 2010.

[88] T. C. Ferreto *et al.*. Server consolidation with migration control for virtualized data centers. *Future Generation Computer Systems*, 27(8), 2011.

[89] T. Wood *et al.*. Black-box and gray-box strategies for virtual machine migration. In *NSDI*. USENIX, 2007.

[90] W. van Heddeghem, S. Lambert, B. Lannoo, D. Colle, M. Pickavet, and P. Demeester. Trends in worldwide ict electricity consumption from 2007 to 2012. *Computer Communications*, 50:64–76, September 2014.

[91] A. Vasan, A. Sivasubramaniam, V. Shimpi, T. Sivabalan, and R. Subbiah. Worth their Watts? - An empirical study of datacenter servers. In *IEEE HPCA*, pages 1–10, 2010.

[92] M. Wang, X. Meng, and L. Zhang. Consolidating virtual machines with dynamic bandwidth demand in data centers. In *IEEE INFOCOM*, pages 71–75, 2011.

[93] Mark Weiser, Brent Welch, Alan Demers, and Scott Shenker. Scheduling for reduced cpu energy. In *Mobile Computing*, pages 449–471. Springer, 1994.

[94] Q. Wu. HAProxy: The reliable, high performance TCP/HTTP load balancer, http://www.haproxy.org.

[95] Q. Wu. Making Facebook's software infrastructure more energy efficient with Autoscale. `https://goo.gl/69aZbd`.

[96] M. Yadin and P. Naor. Queueing systems with a removable service station. *OR*, pages 393–405, 1963.