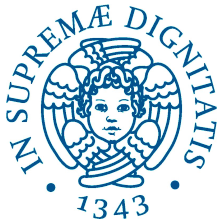


University of Pisa

PhD School of Engineering: "Leonardo Da Vinci"

PhD program in "Applied Electromagnetism in electrical and biomedical engineering, electronics, smart sensors, nano-technologies"

On the Energy Efficiency of Networked Systems

ING-INF/05

Author:
Luca Niccolini

Advisors:
Prof. Giuseppe Iannaccone

Dr. Gianluca Iannaccone

2012

Abstract

Energy is a first-class resource for datacenter operators since its cost is the biggest limiting factor in scaling a large computing facility. The solution embraced by major operators is to build their facilities in strategic geographical locations and to abandon expensive specialized hardware for cheap commodity systems. However, such systems are not efficient when it comes to energy and a considerable amount of research effort has been put in finding a solution to this problem. Furthermore, the need for more programmable and flexible networking devices is pushing the need for hardware commoditization also within the datacenter network.

In this thesis we propose two solutions aimed at improving the overall energy efficiency of a datacenter facility. The first addresses efficiency in computing, by proposing a different hardware architecture for server systems. We propose a hybrid architecture that blends traditional server processors with very-low-power processors from the mobile devices world. The second solution envisions the usage of current server platforms as network switches or routers and provides guidelines for the implementation of power saving algorithms that do not affect peak performance while saving up to 50% power.

This work is based on both theoretical modeling and simulation and experimentation with real-world prototypes.

Acknowledgments

This work would have not been possible without the help, the support and the sympathy of many people.

My advisors first: Giuseppe and Gianluca, they have guided me throughout these three years and they have given me the opportunity to work on the challenging problems exposed in this thesis. Sylvia Ratnasamy for our insightful discussions on the relations between high-speed networking and energy efficiency, and Luigi Rizzo for his very pragmatic approach, always ready to analyze the code and start optimizing it.

Sometimes I wonder how these years would have been like without my old friends and the new ones I met living in Pisa, Berkeley and San Francisco. I have never been lonely and I want to thank them all.

Thanks to Claudio, Caterina and Agata for being my family for almost two years. Thanks to all the guys from the lab in Pisa, to Valentina in particular, without her these pages wouldn't have been printed (literally), and to Martina for being always angry and for creating the chaos in every occasion. Thanks to Massimo, a very good friend.

When I first flew to Berkeley I never expected to meet my "missing brother": Elias, a European Intellectual but mostly a true friend.

I want to thank those who have always been with me even when we were geographically distant, David, Carlo and Tommaso.

A special thanks goes to Chiara who decided to embark with me in this journey.

Needless to say, having a lovely and supportive family has helped more than everything else. I want to thank my mother, my father, my sister and all my relatives. My grandfather in particular who stimulated in me the interest in computing for the first time, perhaps without even knowing it. He loved numbers and statistics in his own way. He kept track of everything he could, from the area of the land he cultivated and the efficiency of his vineyard to the cards that were played during Briscola matches. This thesis is dedicated to him.

Contents

Chapter 1

Introduction

1.1 Motivation

The backbone of today's computing infrastructure is powered by large-scale computing facilities, commonly known as *datacenters*. A typical datacenter hosts tens of thousand commodity servers connected by thousands networking devices and presents challenging scaling problems at all the design levels [90]. However, energy costs constitute the most limiting factor in scaling a datacenter today.

With energy consumption in the order of tens of MegaWatts and electricity bills up to tens of Millions Dollars per year, power-related costs represent a high fraction of the total operating expenses for a datacenter and are growing faster than equipment costs for both computing and networking. Power-related costs represent the second recurring cost, second only to labor. For this reason major datacenter operators like Google, Amazon.com, Microsoft and Yahoo! are building their facilities close to energy sources, where energy is cheap [92].

Scaling-out by adding more and more cheap commodity hardware, and relying on fault-tolerant software to deal with (unavoidable) frequent failures, has been proven a successful approach to achieve large scale while keeping hardware expenses relatively low. The trend toward using cheap commodity hardware is called "commoditization".

Unfortunately, the hardware available today is not *energy-proportional* [60]. An ideal energy-proportional system would consume zero energy while active and waiting for some work to execute. Current servers, instead, consume approximately 50% of their maximum power while running at down to 20% of their max-

imum load. Furthermore, the average utilization of a single server is far from its peak utilization, making the hardware work in its most inefficient operating region.

Commoditization is in progress also for the datacenter network. Similarly to the computing infrastructure the goal is to build a scalable and reliable architecture by using cheap off-the-shelf devices [111]. Inside their own datacenters, operators ask maximum flexibility from the network in order to quickly deploy new network protocols and mechanisms that can efficiently harness the particular network topology [50, 80, 82, 83]. Programmable switches and routers based on the same architecture as servers have been proposed as a means to achieve innovation at speed [33, 71, 74]. However, the move towards commodity servers as network devices suffers from the energy issues introduced above since networks are typically overprovisioned and switching devices operate at low utilization on average.

Energy efficiency improvements at the server and switch/router level have the potential to benefit from the datacenter scale. Even small savings on a single device are amplified by the huge number of devices in a computing facility and by the increasing number of datacenters around the globe.

The contribution of this dissertation is twofold. First, we propose to reduce the energy consumption of computing by using heterogeneous multicore servers. An heterogeneous design includes traditional high-performance and high-energy components coupled with a low-speed and low-power subsystem exploited to maintain external availability and run the workload in periods of low utilization. Then, we address the problem of energy efficiency of packet-processing applications running on commodity multicore servers and we advocate the implementation of smart algorithms, that optimally operate the available computing resources, in order to improve the overall energy efficiency.

1.2 Problem Space

Energy is a fast growing concern both for industry and society. While some alarms have been sounded recently about the energy impact of computing [64], and of the Internet in particular [29], we believe that the current computing and networking technologies have a high potential for reducing the energy that society wastes everyday . Furthermore, the total energy consumed and embodied in the Internet, estimated by Raghavan *et al.* [112], is negligible compared to the overall energy used by society, with the total power absorbed by datacenters worldwide representing only 1.3% of the global energy consumption [95].

We focus on the energy efficiency of datacenters as a means to achieve a better scaling of computing facilities by lowering expenses for operators. The energy consumed in a datacenter is logically divided in two categories: (i) the energy absorbed by IT equipment; (ii) the energy indirectly consumed to keep the above equipment operational. The former represents the electricity spent to perform the main operations for which the datacenter has been built, *i.e.* computing, networking and storage. The latter denotes the energy overhead and includes, among other sinks, the power distribution and transformation infrastructure, the airflow and cooling system, the lighting infrastructure and personnel offices.

The overall efficiency of a single facility is expressed in terms of the *Power Usage Effectiveness (PUE)* metric. PUE is defined as the ratio between the total energy absorbed by the facility and the energy utilized by IT devices (after power conversion). The ideal PUE value is 1.0 and implies that all the energy entering a datacenter is spent for useful work.

Given this classification of datacenters energy the work towards its reduction follows two parallel paths: (i) *Reducing PUE* and (ii) *Reducing the IT costs*.

1.2.1 Reducing Power Usage Effectiveness

The U.S. Environmental Protection Agency published a report [64] in 2007 showing that computing was responsible for 1.5% of the total U.S. energy consumption, with datacenters being a big fraction of that. Also the average datacenter, at the time, was reported to operate with a 100% energy overhead, *i.e.* with a PUE of 2.0. Since then, a lot of effort has been put in reducing this overhead through the sharing of best practices [9] and measurement data [13] that were before retained by operators as corporate secrets.

Best practices include how to continuously and correctly measure the PUE, how to realize efficient cooling systems by leveraging free external resources (like cold air and water) and how to properly place the servers to maximize the air flow, how to distribute power by reducing the number of AC/DC conversions and how to harness reusable energy sources and recyclable materials to reduce the environmental impact.

Cooperation between major operators has been fostered by initiatives such as "Climate Savers Computing" [9] and "The Green Grid" [41] and eventually resulted in a community effort with Facebook releasing the whole design of their new datacenters as an opensource project [27]. The project includes the technical specifications at each design level, from custom motherboard schematics and

server chassis design [77] to electrical design of power distribution to the mechanical airflow system [28].

As a result of this engineering work the PUE in large-scale datacenters is now approaching its theoretical minimum, with the best-in-class operators advertising a one year average[1] of 1.14 [13] and a seasonal minimum of 1.07 [11]. This optimization process is slowing the growth of datacenter electricity usage with the actual values being significantly lower than the most optimistic predictions in 2007 [95].

1.2.2 Reducing IT costs

Minimizing the total cost of ownership (TCO) of IT equipment is a challenging problem since it directly involves possible optimization both in hardware and software. It is the topic of active research in Academia and industry and the proposed solutions reflect the evolution of datacenters towards today's warehouse-sized general-purpose computers [90].

Workload specific. A first approach, that keeps the base hardware architecture untouched, is to explore software solutions for optimizing power consumption in presence of specific workloads [121] . Early days datacenters hosted few tens of servers clustered together to run one specific application, such as high-performance computations or distributed databases. With the rise of cloud computing [56] however, datacenters should be able to run diverse workloads, ranging from on-line data-intensive queries [107] that require low latency, to long-running MapReduce [2, 4, 72] batch jobs that need high computing throughput and large network bandwidth. Furthermore, with Infrastructure as a Service (IaaS) [3,22,30] and Platfrom as a Service (PaaS) [12, 14, 42, 44] solutions the user has complete control over his own virtual machines and applications and no assumption can be made on the workload that the datacenter will run. Thus, the datacenter is seen as a general-purpose warehouse-scale computer [90], and workload-specific optimization are not generally applicable. With one exception.

The one specific-workload that is common to all the computing facilities and whose optimization are widely applicable is the one related to networking and packet processing applications. Switching, routing and other packet-processing

[1] Large datacenters are built in zones where it is possible to use external air or water for cooling, for this reason there is a seasonal fluctuation of the PUE due to external weather condition and temperature.

applications typically run on dedicated hardware and provide network connectivity to computing and storage hosts.

Our work on optimizing network devices for energy efficiency leverages the trend towards commoditizing the network in datacenters. New network topologies [49, 80, 82, 83] have been proposed to flatten the network. These architectures seek for a higher bisection bandwidth and lower over-subscription rates by exploiting multiple paths inside the local network. They have the potential to reduce equipment cost by installing cheaper devices. The problem with advanced topologies is that they require fundamental changes in the way packet forwarding is performed. High level forwarding-plane protocols, like OpenFlow have the role to blur the existing gap between traditional MAC layer (L2) and network layer (L3) packet forwarding and are pushing the need for more flexibility and programmability from the networking devices.

We believe that the solution to achieve the flexibility requested by this fast-changing environment is the deployment of programmable routers based on the same general-purpose server architecture used for computing and storage. A network like this can accommodate changes at the speed of software deployment. Furthermore, software developers can benefit from a well-known programming environment and Instruction Set Architecture [33, 74].

Low-power manycore. A natural approach to build an energy-efficient computing infrastructure is to use energy-efficient hardware building blocks. The fast growth of the mobile devices market has pushed the design of low-power processors and platforms in order to maximize devices' battery life. Despite being intrinsically low-performance, these platforms are more efficient than traditional server hardware in terms of *instructions per Joule*. Mobile systems design, targeted at low-power operations, has lead to well-balanced systems while in the server world we are assisting at an ever increasing performance gap between CPU, Memory and I/O. Processor cores for mobile devices are indicated as *small* cores, or more colloquially as *wimpy* cores, due to the reduced number of transistors needed to implement their functionalities. Conversely, server processors with multiple out-of-order execution engines and large multi-level caches are also indicated as *big* or *browny* cores.

Several attempts have been made to run datacenter-like applications on systems built by a high number of low-power elaboration units. These solutions differ in their integration scale and can be divided in tightly-coupled and loosely-coupled designs.

The authors of FAWN (a Fast Array of Wimpy Nodes) [53] successfully explored the ability to run a key-value storage service on an array of loosely-coupled (ethernet connected) nodes powered by a 500 MHz AMD Geode CPU and flash storage. They show that a FAWN-like architecture can be cost-effective for distributed storage seek-bound workloads.

Some industry solutions, recently appeared on the market, implement a finer scale integration of low-power cores. Seamicro product line includes platforms with up to 768 1.6 GHz Intel Atom cores in one single box [36], thanks to a high-speed integrated network fabric, and have been adopted, among others, by Mozilla to power their web-servers [35] and by eHarmony [10] for distributed data analysis with Hadoop [4]. Tilera pushes low-power cores integration even further, selling a 64-core system-on-a-chip [62] that includes 1 and 10 Gbps network, PCIe and DDR2 controllers. Each core has a shallow pipeline, for low-power operations and more predictable performance. Fast intra-core communication is achieved by means of a multi-layer mesh network and on-core L2 caches are leveraged to implement a distributed L3 cache. Experience with memcache [76] in-memory key-value storage shows that the TilePRO64 system is three times more energy-efficient than a Xeon based system [63].

However, it has been shown that solutions with a large number of low-power cores are not appropriate for all the datacenter workloads, resulting in significant performance slowdown for the most demanding applications like parallel databases [99], web-search [91] and massive-data sort [123]. A high-scale datacenter must be general purpose, it should be able to run diverse workloads, most of them unknown at design time. The main drawback of low-power manycore architectures is that the single nodes are highly constrained in their hardware capabilities. Usually they have a $2 - 4$ GB addressable memory bound and a low number of connections for disk I/O. This lack of hardware capabilities imposes strict constraints on the application programmer. Furthermore, many-core architectures force a shift in the programming model and may require a complete code refactoring in order to benefit from the high core count.

Software development is an important factor to take in consideration when computing the cost of a new hardware infrastructure in the datacenter. Hölzle [89] proposes a rule of thumb for which it is not economically advantageous to switch to a *wimpy* architecture if the single core performance is less than half the performance of current *browny* cores.

Energy proportionality. A different approach to reduce power expenses in a

large computing facility is to address the non-proportionality of the servers.

Many of these efforts focus on cluster-level solutions that dynamically consolidate work on a small number of servers. Thanks to this, underutilized servers can be turned off at will and computation moved between machines if needed [113, 120]. Load balancers and task schedulers are instrumented with a prediction component, based on previous history. Prediction results are exploited to starve some machines of requests and consequently turn them off. The overall goal is to obtain a quasi-power-proportional ensemble with non power-proportional building blocks by consolidating the load, with server or rack granularity.

To further improve the proportionality of the system as a whole, hardware heterogeneity can be exploited. In a datacenter with two different types of clusters, the classic one with blade servers and a low-power one with low-power systems, the latter can be leveraged to serve requests in a long low-load period [96, 110]. Solutions in this space affect hardware provisioning costs, charging the operator of the burden to provide its facility with both systems to handle high-load and proxy-systems for low load.

The general downside of consolidation techniques is that servers on/off cycles [2] and the workload consolidation overhead are in the order of tens of seconds. Consolidation being typically achieved through virtualization and live migration of virtual machines [59, 67, 69, 105]. Furthermore, it is challenging to predict the incoming load on a fine timescale [24, 81]. One more problem with consolidation is that network reachability is lost when machines are turned off. Most datacenters run distributed key-value storage and distributed filesytems [66, 70, 73, 76, 79] in which replicas are distributed among all the servers. Thus, a node not performing computation might be required because it holds a replica that other nodes need for their computation. For this reason, cloud providers are reluctant to turn off computers, not to affect high availability. As an example, Amazon's solution to increase the overall server utilization while amortizing the cost of keeping servers idle is to offer a service in which spare resources are offered at a variable price [1], depending on the overall datacenter utilization, and users can bid on the instantaneous price.

A more integrated approach to power proportionality is to solve the problem for a single server in order to benefit from the massive scale of the datacenter and to hide the power saving mechanisms from the programmer as much as possible. A solution in which a system can quickly transition to a deep sleep state while

[2] Current commodity servers do not expose system low-power states that are found in desktop machines such as standby and suspend (*i.e.* ACPI S3, S4 and S5 states).

still maintaining external network visibility and availability is needed to allow per-server power-proportional behavior [48, 106]. Servers running multiple services at the same time shows short idle periods in the sub-second scale, making practically unfeasible to completely shut down the machine as proposed by consolidation approaches.

We explore a similar approach to that first proposed by Meisner *et al.* [106] requiring that the system is never unavailable to serve incoming requests. Despite large periods of low utilization, servers are rarely completely idle and in the case of latency-sensitive workloads active low-power states are the only solution to reasonably scale down energy consumption with server load [107]. We leverage hardware heterogeneity at the platform level by exploring the benefits of a system that mixes high-performance energy-inefficient processors (with big cores) and low-speed high-efficiency processors (with small cores). We believe that such hybrid architectures comprising small cores and big ones can provide energy benefits at the single-server level, by maintaining availability through small cores and waking up big cores when the load increases so that service-level agreements (SLA) are not violated.

The idea of hybrid architectures for energy-efficiency is not new, systems using mixed processors are starting to appear in the mobile devices market. Major manufacturers, like NVidia, ARM and Qualcomm are pushing solutions with a big and small cores mix [45, 47] or with cores operated at different frequencies and voltages [46]. A hybrid design helps to increase energy efficiency by lowering the power envelope of the system. Furthermore, the heterogeneous hardware design can be made transparent to application programmers. In the aforementioned mobile platforms for example, small cores are used to run background tasks, such as checking e-mails, while big cores are awakened whenever high computational power is needed, typically for multimedia tasks. The switch between small and big cores is completely managed by the firmware, allowing the programmer to abstract from the underlying hardware technology.

We study the problem of heterogeneous multi-core in server hardware and explore the feasibility of mixed architectures with current high-end processors and mobile processors, specifically within the Intel Xeon and Atom processor families [18].

1.3 Thesis Overview and Contribution

The demand for highly-parallel computation on massive amounts of data is increasing at a fast pace, pushing datacenter operators to enlarge their facilities or build bigger ones. Today, datacenters are limited in scale by energy consumption with 80% of the recurrent costs scaling with electricity costs.

In this thesis we tackle the problem of energy-efficiency of the networking infrastructure and power-proportionality of the computing machinery in large-scale datacenters. Our focus is on the power-proportionality of the basic building blocks: general-purpose servers and packet-processing devices.

In Chapter 2 we present the hardware mechanisms that today's' systems implement, and that motivate our work, with a specific focus on those supported by the Operating System. Then in Chapter 3 and Chapter 4 we present the main contribution of this thesis that is twofold:

- In Chapter 3 we make the case for *hybrid datacenters*. We address the challenges in building a computing infrastructure that blends low power platforms with high performance ones. We show how these designs can handle diverse workloads with different service level agreements in an energy efficient fashion. We evaluate the feasibility of our approach through experiments and then discuss the design challenges and options of hybrid datacenters.

  Given the positive results of this analysis, we propose a hybrid multiprocessor architecture for energy-proportional web servers and datacenters, to achieve significant energy savings at the price of small increase in service delay. The server consists of a subsystem with low-power cores, always on and serving user requests when load is low, and a high-performance subsystem, by default in sleep mode, woken up only when the load is high. Through experiments on a hardware prototype and an accurate timing and power model of the server we assess the performance and the energy benefits for a Web server. Finally, we explore design guidelines for future energy-proportional systems.

- In Chapter 4 we approach energy-efficiency in datacenter networks. We aim at improving the efficiency of packet-processing applications running on commodity server hardware without compromising their performance. We investigate the design of a router that consumes power in proportion to the rate of incoming traffic. We start with an empirical study of power

consumption in current software routers, decomposing the total power consumption into its component causes. Informed by this analysis, we develop software mechanisms that exploit the underlying hardware power management features for more energy-efficient packet processing. We incorporate these mechanisms into Click [93] and demonstrate a router that matches the peak performance of the original (unmodified) router while consuming up to half the power at low loads, with negligible impact on the packet forwarding latency.

In Chapter 5 we draw our conclusion, address the applicability of our work and provide ideas and plans for future work.

Chapter 2

Background

In this chapter we present the hardware features and mechanisms that can be found in modern processors and that motivates our work.

2.1 Small and Big cores

Traditionally, power efficient designs attempt to find the right balance between two distinct, and often conflicting, requirements: (i) deliver high performance at peak power (i.e., maximize compute capacity for a given power budget) and (ii) scale power consumption with load (i.e., energy proportionality and very low power operations). A fundamental challenge in finding a good balance is that, when it comes to processor design, the mechanisms that satisfy the two requirements above are significantly different. High performance requires mechanisms to mask memory and I/O latencies using large multi-level caches (today's server processors use three cache levels with the last-level cache projected to soon reach 24MB [17]), large translation lookaside buffers, out-of-order execution, high speed buses, and support for a large number of pending memory requests. These mechanisms result in large transistor counts leading to high leakage power and overall high power consumption. In a modern high end processor, less than 20% of the transistor count is dedicated to the actual cores [117, 122]. This is reflected in the processor high power consumption. In our work we measured the power needed to keep the "uncore" components active while reducing the "core" power consumption to the minimum possible. We found that approximately 25 W are needed for a modern 6 cores Westmere CPU.

Mobile processors designs, on the other hand, focus on those processor features with low-power operations. For example, the Atom processor [78] includes an in-order pipeline that can execute two instructions per cycle, a small L2 cache and power-efficient clock distribution. The Atom simple micro-architecture implements only those features that give the best performance per watt. A strong design constraint during the processor design has been to implement a feature only if it guarantees "1% performance increase, for less than 1% power increase" [61]. *E.g.*, simultaneous multi-threading has been implemented (in higher-end models of the Atom family) since it provides a 36% performance improvement versus only a 19% increase in power.

This results in a strongly reduced transistor count with low leakage power and limited power consumption at low load. Further, Atom design is focused on allowing quick and frequent transitions to a very low power state (e.g., 80 mW with less than 100 μs exit latency [78]).

2.2 Support for power management

Modern systems provide a variety of power management mechanisms that operate at all levels in the platform. Some are only visible to the hardware while others can be controlled directly by the operating system. Here, we introduce the basic controls that are exposed to the operating system and we briefly summarize the mechanisms only available to the hardware.

Core idle states (C-states). Each core in a package can be independently put into one of several *low power* or "idle" states, which are numbered from C1 to Cn. The state C0 is present on all platforms and refers to a core that is executing instructions and consuming the highest amount of power since all the areas within the core are powered on. Higher numbered states indicate that a successively larger portion of the core is turned off resulting in a lower power draw. The particular C states that are supported are vendor and processor specific. The Westmere processors we use in Chapter 4 offers three C states (C1, C3 and C6) and allows to put each core, independently from others on the same package, in different "idleness" states.

A core enters into a C state either by executing a HALT or MONITOR/MWAIT instruction. The state entered is set using special registers of the CPU. The wakeup event can be either an interrupt or a write access to the monitored memory range specified by the MWAIT. C states are exited (to return to the C0 state) when the

desired triggering event occurs. Considering the larger amount of circuitry turned off, higher idle states take longer to revert back to the fully operational C0 state. The time to return from a higher numbered C state back to C0 is termed the *exit latency* of the state.

As an example, when in C1 the core is *clock gated* (the clock distribution into the core is blocked), but the core is still drawing power. Going from C1 to C0 only requires the clock distribution to be re-enabled; thus, the time it takes to execute the first instruction (after being in C1) is comparable to that of a cache miss.

In C3, the L1 and L2 caches are flushed and turned off. In C6, in addition to turning off the clock and the caches, the core is *power gated* (power distribution into the core is disabled) after the core's state is written to a reserved part of the L3 cache. Consequently, the time to enter/exit C3 and C6 is much higher (compared to C1) due to the need to save and restore state, and to allow the voltage levels to stabilize. Furthermore, on wakeup the core will start with a cold cache, which results in lower performance.

The time to exit from a C-state is called "exit latency", and we will measure it for our system in Section 4.6.2.

Processor performance states (P-states). While a processor core is in active C0 state, its power consumption is determined by the voltage and operating frequency, which can be changed with Dynamic Voltage and Frequency Scaling (DVFS). In DVFS, voltage and frequency are scaled in tandem. This is because a particular frequency requires a minimum voltage for the circuit to operate in a stable and reliable way. When voltage is reduced, transistors take longer to switch, and signals take longer to propagate, leading to the circuit becoming unstable. To preserve stability and correctness, the operating frequency must also be lowered.

Modern processors offer a number of frequency and voltage combinations, each of which is termed a P state. P0 is the highest performing state (pegged at the maximum voltage required by the highest operating frequency); subsequent P states operate at progressively lower frequencies and voltages. Transitions between P states require time to stabilize the frequency, but the transition can be applied while the processor is executing instructions, without halting or freezing the core nor flushing caches. While in general, cores can have independent power planes, most processors incorporate a shared voltage rail across the cores, which forces a shared P-state between cores belonging to the same package. Effectively, all the cores run at a frequency that is dictated by the most hungry core. The processor in

our system offers 14 P states with frequencies ranging from 1.6 GHz to 3.3 GHz.

Additional power states. Even though we do not use them directly in our system, we need to mention two additional classes of power states: processor throttle states (T-states) and platform sleep states (S-states). T-states force the processor frequency to be lowered when power dissipation approaches the thermal limit to prevent the processor, or other subsystems, from being damaged. T-states are similar to P-states, but they are managed by the core's firmware and cannot be controlled from software.

Turbo Boost [20] is an Intel proprietary mechanism that aims at increasing the performance of a system that is not fully using all its cores while assuring that processor power constraints are never violated. Turbo Boost is activated for cores that are in the active C0 states when only a subset of the cores in the package is active. In such particular circumstances the maximum operating frequency of the active cores can be increased while still meeting the power, current and temperature limits of the processor package. Akin to T-states, "boost-states" are only visible to hardware and can potentially introduce non-predictable performance increases since its activation depends on the workload and on the operating conditions (*i.e.* number of active cores, estimated current consumption, estimated power consumption and measured processor temperature). In the following work the Turbo Boost techniques have been disabled in order to get repeatable experiments and measures.

S-states are defined for the entire system (not just the processor). S0 is the active state, while in higher states portions of the system are turned off after saving their state to RAM or persistent storage. The S-states reflect various degrees of system-wide sleep and correspond to the familiar "stand-by" and "hibernation" modes of laptops and desktop. The transition times involved in the S-states are measured in seconds. We do not consider them in our work because the latencies involved are much higher than the latencies high-speed services can tolerate and also because server systems typically do not implement such features.

Chapter 3

Hybrid Datacenters

A more efficient use of energy is a key challenge for modern datacenter design: it would enable significant reduction of costs and increase of computing power density per unit volume. Datacenters' building blocks are commodity servers, providing ease of deployment, low costs and familiar programming environments. However, current servers - and therefore datacenters - are not energy efficient, specifically at low load. Barroso *et al.* [60] report that more than 40% of energy is consumed to keep servers online while not executing any useful work, and each server is typically used below 50% of its peak performance. Hence, the call for *energy-proportional* systems, defined as systems whose power consumption is proportional to the computing load.

In this chapter we show that a hybrid server architecture is a promising option for achieving energy proportionality while paying a small additional cost in performance. A hybrid multi-core architecture consists of a mix of low-power cores and high-performance cores that coexist on the same motherboard and are coordinated by the operating system or by the application. Low-power and high-performance cores are classified respectively as "small" and "big" cores since advanced features (such as large caches, out-of-order execution, etc.) affect core area by requiring a larger number of transistors.

The basic concept is extremely simple: small cores are always on, keeping the system responsive and serving user requests when the load is low. Big cores are by default in a deep stand-by mode, and are woken up only when the load is high. This approach is able to exploit several trends in semiconductor and processor technology. First, with the introduction of the Atom processor ("small" core in our terminology) there is now a low-power alternative to the traditional server pro-

cessor (Xeon product line) that shares the same instruction set architecture (ISA). Sharing the same ISA allows for providing a single, and familiar, programming environment to developers. A hybrid system is therefore equivalent to a traditional server and requires no changes to the datacenter architecture or the management processes of datacenter operators. Second, the performance of small cores is continuously increasing to meet users' expectation of a full Internet experience on smartphones, netbooks and tablets. Most importantly, battery life and small form factor requirements force designers to select and implement energy-efficient features and mechanisms for improving performance per Watt [61]. Finally, high performance cores are adopting power saving features (clock gating, power gating, etc.) that permit individual cores and other subsystems within the same server to be turned off to maximize energy efficiency at low load [19].

It has been recently pointed out that despite large periods of low utilization, servers are rarely completely idle and in the case of latency-sensitive workloads active low-power states are the only solution to reasonably scale down energy consumption with server load [107]. Therefore, we believe that hybrid architectures comprising small cores with big ones can provide energy benefits at the single-server level, by maintaining availability through small cores and waking up big cores when the load increases so that service-level agreements (SLA) are not violated.

We presented the key concept in a position paper [68]: here we present a complete analysis showing experimental results on a prototype hardware with realistic workloads. Using the prototype we derive and validate a simple, yet accurate, model of the service delay and power on the hybrid system. We then use that model to assess the feasibility of the hybrid approach as well as explore the design space. Specifically, we study the design space along two dimensions: the ratio of small vs. big cores in the system and the relative performance of small core vs. big cores when processing Web requests. We show that the current small core performance levels make hybrid multicore design feasible and that hybrid designs are an extremely promising solution to achieve energy proportionality.

The rest of the chapter is structured as follows. In the next section we describe the prototype system and present experimental results with different types of Web server workloads. Then, in Section 3.2, we develop a high-level model of the hybrid server architecture, capable of accurately simulating timing performance and time-dependent power consumption. The model is validated, in terms of accuracy in the estimation of distributions of service delay times, through comparison with experiments on the hardware prototype. In Section 3.3 we use the latency and power model to explore various design options and estimate the energy savings

that can be achieved. We review related work in Section 3.4.

3.1 Hybrid Server Prototype

We begin our analysis by measuring the performance of a hybrid architecture research prototype. In the following subsections we describe the hardware and software configuration and discuss experimental results.

3.1.1 Prototype architecture

The hybrid server prototype is based on a standard dual-socket motherboard suited for Intel Xeon Harpertown [18] and compatible CPUs; however, the first socket hosts an Intel Atom 330 processor, mounted by means of an adaptation board. The board maps Atom pins to the Xeon socket and adjusts voltages accordingly. Figure 3.1 shows a high-level view of the system with the two CPUs connected to the northbridge chipset (MCH) through a shared front side bus (FSB). MCH manages access to shared memory and to peripherals through the I/O controller (IOCH).

Table 3.1 summarizes the main differences between the two CPUs. Since the original purpose of the prototype was demonstrating hybrid server feasibility, some tweaks have been made to achieve interoperability at the expense of overall performance, degrading some Xeon features to those supported by Atom. For example, Xeon clock and bus frequencies have been reduced to make the FSB work at the maximum sustainable frequency of the Atom, *i.e.*, 533 MHz. For this reason the big CPU (a Xeon X5450) is constrained to a 1.2 GHz operating frequency, much lower than the 3 GHz maximum clock rate. Furthermore, cache coherency is achieved by making the smaller cache ignore snoops for entries that exceed its size.

Atom and Xeon processors have different microarchitectures but implement partially-overlapping Instruction Set Architectures (ISAs). They both have full x86 compatibility exhibiting some differences only in instruction set extensions (*e.g.* SSE instructions [39]). However, operating system support is needed to let the applications abstract from these asymmetries [114].

The server runs the HeterOS operating system [100], developed from Intel, that is a 64-bit compliant OS providing the following improvements on top of Linux:

Migration on fault: an application can execute instructions known only to a sub-

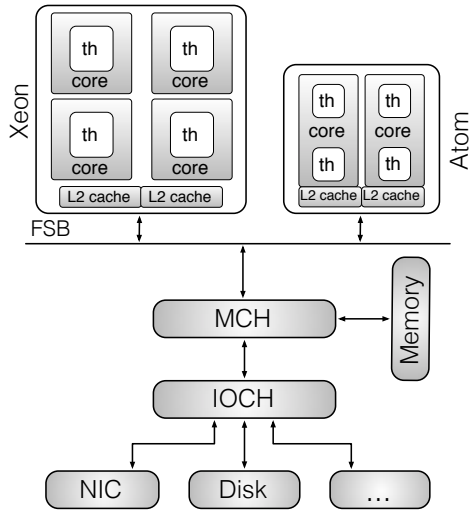


Figure 3.1: Prototype hardware architecture. FSB is the Front Side Bus, MCH is the Northbridge chipset, IOCH is the I/O controller.

set of cores. The operating system handles unknown instruction exceptions and migrates the process that caused the fault to a core that understands the instruction;

Dynamic scheduling: HeterOS provides both *slow cores first* and *fast cores first* scheduling policies. They dynamically reschedule tasks, based on performance metrics, to take advantage of CPU bound and I/O bound phases within the same exectution flow.

As already mentioned, the hybrid server prototype is not optimized for power consumption, and therefore exhibits high power consumption when only the Atom is active. The main reason for that is the legacy shared-bus architecture, where the bus is always on and MCH and IOCH do not have sleep states. Measured wall power is 228 W at idle and 268 W at maximum load. For this reason, we will use experiments to verify the feasibility of our proposal and to validate our model for a hybrid server. Then, we will use the model to explore the performance achievable if power saving features were fully implemented.

Name	Xeon X5450	Atom 330
Clock rate	1.2GHz	1.6GHz
Threads	4	4
Cores	4	2
Hyperthreading	No	Yes
L2 Cache	2x6MB	2x512KB

Table 3.1: Main differences between CPUs in the hybrid server prototype. They have the same number of threads thanks to Atom implementing hyperthreading.

3.1.2 Software configuration

Here we describe the software setup used to assess the performance of our hybrid architecture prototype serving HTTP requests. The workbench consists of a request generator connected to the server through a 1 Gbps Ethernet link.

Request generator. The client machine runs multiple instances of the *http_load* [15] request generator in order to increase the request rate up to the maximum load sustainable by the server. By using this tool, requests can be generated at a constant rate or with a specified degree of parallelism. The client machine also measures per-request latency by computing it as the interval between the connection instantiation at the server and the time the response is received by the client.

Web server. The software stack on the server machine is based on the *Lighttpd* [21] Web server and its *FastCGI* module. Lighttpd is a lightweight and modular Web server while FastCGI is a typical solution for boosting the execution of dynamic requests. It defines a communication protocol between the Web server worker threads and external processes, using sockets as the communication channel. FastCGI processes are created at Web server startup and can run on the server machine as well as remotely. In the FastCGI working model processes are persistent and serve more than one request, saving the time needed in legacy CGI systems to fork one process per request.

Processes are distributed in a way that enables us to exploit multicore parallelism. Lighttpd runs one worker thread on the first Atom core while FastCGI processes run on both the Xeon and Atom processors. The worker thread is responsible for accepting all the incoming requests and scheduling them on the most

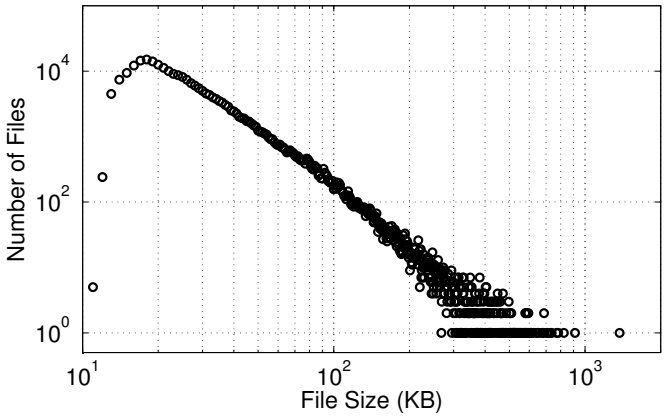


Figure 3.2: Dataset file sizes follow a power-law distribution

suitable FastCGI process. We want requests to be allocated to a core and to run on it until completion, without incurring in rescheduling overhead.

Scheduler. In our solution, processes are statically assigned to cores with a one to one mapping. We modified the FastCGI module to make it aware of the underlying asymmetric processor set and changed the default load-balancing algorithm into a threshold-based scheduler. The scheduler allocates incoming requests to small cores when the load (as measured by the number of concurrent requests) is below a defined threshold (S_t) and to big cores when the threshold is reached. The threshold can be larger than the number of small cores, in that case pending requests are queued and their execution is deferred. The threshold effectively is a knob to trade-off power consumption for latency since increasing S_t reduces the duty cycle of large cores. With this simple mechanism we create opportunities for big cores to go in their deepest sleep state whenever the server is underutilized. In Section 3.3 we analyze the effectiveness of this mechanism in terms of energy efficiency.

Dataset. We collected a 10 GB dataset from the English version of Wikipedia. Our dataset is composed of $250,000$ files with sizes ranging from 6 KB to 1.4 MB. This is a representative dataset for Web applications that work with plain HTML files. Figure 3.2 shows that file size in the dataset follows a Zipf distribution, as in Ref. [58].

Workload. The definition of a workload that is both simple and representative of the applications of today's datacenters is not straightforward. Servers in virtualized environments and MapReduce nodes must be as general purpose as possible. Instead, in the underlying infrastructure that powers large websites, most servers act as specialized back-end nodes running only one kind of requests on their fragment of the whole dataset (e.g. memcached nodes, message search). We consider the following workloads to be executed on files in our dataset:

readfile: statically serve the file as is;

ksort: sort (key, value) pairs based on key values;

compress: deflate the input file using gzip;

count: computes a histogram of bytes.

The workloads above allow us to explore different deployment scenarios. The readfile workload represents the traditional Web server; the other three instead require to process and perform computation on the input data before sending the response to the client. The computations vary in complexity and the subsystems (CPU, memory, I/O) they employ. The workloads are conceptually simple and do not represent applications in which slow blocking database requests are made. However, they are representative for servers managing a high number of concurrent connections with low latency. Exploring more complex workloads is a major part of our future work.

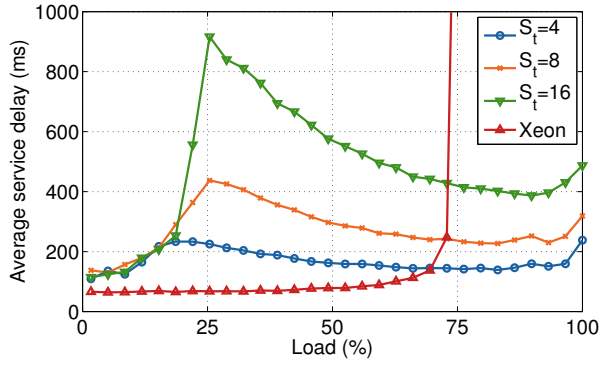
3.1.3 Experimental results

In order to assess the service delay implications in systems with an hybrid set of processors, we stress the server by gradually increasing the input query rate up to the maximum sustainable load. We only report results for the *count* workload since results for other workloads exhibits a very similar behavior.

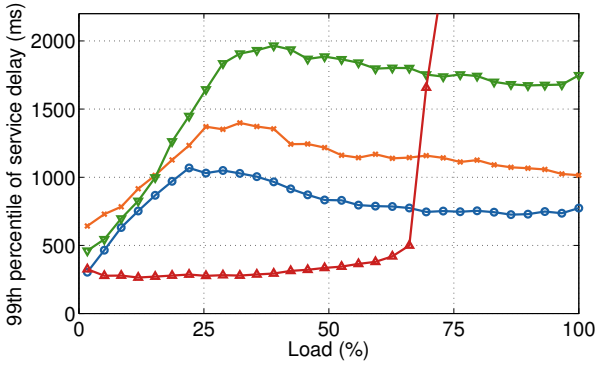
Figure 3.3 reports the system performance in terms of service delay. It shows values for the average service delay (top) and the 99th percentile of the service delay (bottom). Each curve represents a different value for the scheduling threshold S_t, *i.e.*, the maximum number of simultaneous requests served by small cores.

By setting $S_t = 0$ we force the system to use big cores only. As can be seen they sustain up to 75% of the maximum load of the hybrid solution. All the other curves have a peak around 25% of the maximum load, where requests are being processed almost exclusively by low-performance cores. When the load

increases further, requests are forwarded to big cores and performance improves, until saturation of the whole system is reached. From Figure 3.3 we evince a $3X$ speedup for our Xeon processor compared to Atom. Even if both processors operate at the same clock frequency, the Atom cores are much slower as they do not implement all performance boosting techniques of a Xeon core (such as out-of-order processing, aggressive speculation, and instruction transformation).



(a)



(b)

Figure 3.3: Experiment on hybrid prototype server: Average (a) and 99th percentile (b) service delay for the *count* workload for scheduling threshold $S_t = 0, 4, 8, 16$. Note that for $S_t = 0$ only Xeon is active.

3.2 Latency and power model

In this section we present a high-level model for timing and power consumption of a hybrid server. The model is validated with experiments on the prototype and then used to assess achievable energy savings and performance.

3.2.1 Response Latency

The hybrid architecture we consider is a system with two distinct types of elaboration units — a small CPU with low-power cores and a big CPU with high-performance cores — which share a single infrastructure. A scheduler is responsible for the allocation of incoming requests between processors, *i.e.*, it selects a CPU c for each incoming request.

A CPU can run T^c concurrent execution threads (see Table 3.1). When it is fully loaded (*i.e.*, if $load = T^c$) incoming requests are queued and will start being served as soon as one of the execution units becomes available. The total service delay a request experiences is then the sum of two contributions: *queuing delay* and *execution delay*. For each CPU we only model execution delay since queuing delay directly depends on the requests execution speed and on the arrival process.

For a given workload type, service time is a function of the size of the requested file, of the processor selected by the scheduler and of the number of currently active requests ($load$). The file size determines the number of instructions to run while the instantaneous processor load is an index for the inter-process interference due to competition for shared resources (*i.e.* L2 and L3 caches, shared bus, $\cdots$). Let t be the time a request starts being served and c the CPU that serves the request. The service delay D is computed as

$$D(\text{file}, load, c) = \alpha(c) + \text{size}(\text{file})^{\gamma} \cdot \beta(c, load) \tag{3.1}$$

α models the time needed to accept a request before starting the execution, and depends only on the type of processor. β accounts for the performance of processor c at a given load and is a workload-dependent coefficient. Given the restricted range of file sizes in the dataset we express the dependency on file size as a power function with exponent γ.

3.2.2 Power Usage

It has been shown that the CPU is the most energy-proportional component in a server system [60], thanks to idle sleep states (C-States) with short transition times and active sleep states (P-States) that trade execution speed for energy. Other components, such as memory, disks and high speed buses either do not implement sleep states or are rarely completely idle, even if underutilized [107]. The trend in server architectures however, is promising due to the progressive integration of memory and I/O controllers on the processor die. This can affect power consumption in two ways. First, the shared bus is replaced by point to point links (*e.g.* QuickPath Interconnect, QPI) that allow for more fine grained power control. Second, the hardware logic dedicated to interfacing with memory and peripherals directly depends on the CPU power state and can quickly transition to and from deep sleep states.

Without loss of generality, for the purpose of evaluating power consumption we can logically divide the whole system into subsystems each associated to a given CPU. The total power consumption $P(t)$ can be written as

$$P(t) = \sum_c P^c(t),$$

where $P^c(t)$ is the power consumption of the subsystem associated to CPU c. Each subsystem can be in three different energy states: *active*, *sleep* and *transition*, depending on the state of the corresponding CPU.

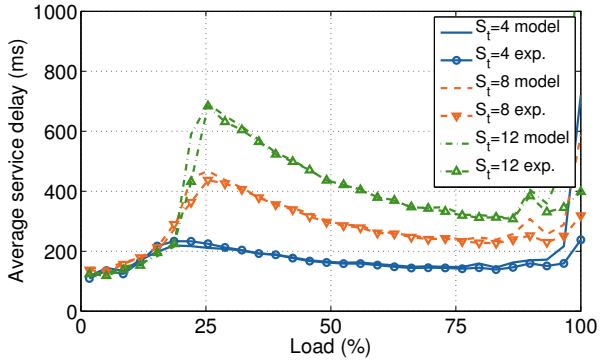
Let us consider the subsystem associated to CPU c: its instantaneous power consumption, $P^c(t)$, is a function of the state: P^c_{sleep} is the power consumption of the CPU in sleep state. During transition between active and sleep state, power consumption is approximated by P^c_{idle}, power dissipation when the CPU is idle. Finally, when CPU is active, power is proportional to the instantaneous processor load L^c:

$$P^c(t) = P^c_{idle} + \frac{P^c_{max} - P^c_{idle}}{T^c} \cdot L^c(t), \tag{3.2}$$

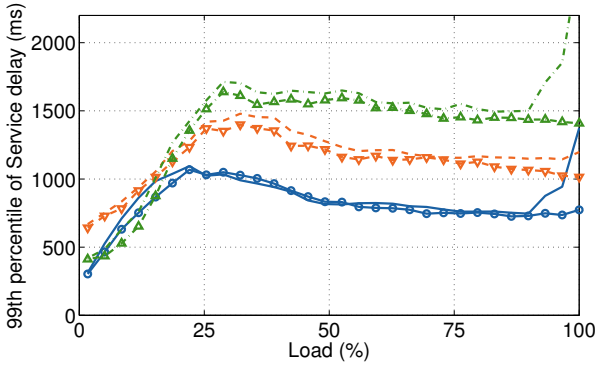
where P^c_{max} is power consumption at 100% utilization.

3.2.3 Model validation

The model has been implemented in an event-driven simulator using the *simpy* [37] framework and has been tested against experimental data. Traces from experiments in Section 3.1.3 are used as input to the simulator in order to replicate the request arrival pattern.



(a)



(b)

Figure 3.4: Comparison between simulation and experiments for average (a) and 99th percentile (b) of service delay.

Values of the model coefficients have been experimentally measured using the test bench presented in Section 3.1.2 and instructing the request generator to issue a fixed number N of simultaneous requests. For each $N \leq T^c$ requests are immediately executed and no queuing occurs. We generated random requests across the whole dataset spectrum and measured the execution delay of each request. For each CPU and load level we compute model parameters α and β, introduced in Section 3.2.1, using Equation 3.1 to fit the experimental data. In our case we obtain a good fit with a linear function, *i.e.*, for $\gamma = 1$.

Figure 3.4 shows the request service delay, average (left) and 99th percentile (right) for the *count* workload. Lines with symbols represent experimental data,

simple lines simulation results. As can be seen, despite its simplicity, our model correctly replicates the behavior of the real system with a delay peak in correspondence of slow cores saturation and a performance speedup as soon as the scheduler starts allocating requests to fast cores. The error introduced by the model is within few percentage points even for the 99th percentile, which is harder to replicate than the average.

We validate our assumption by showing that the model correctly predicts the system behavior for different values of the scheduler threshold $S_t(4, 8, 12)$. The threshold indeed only affects queuing delay on small cores. For example when the threshold is set to the number of small cores (*i.e.*, $S_t = T$) there are no pending requests and queuing delay is null on small cores.

3.3 Evaluating hybrid server design

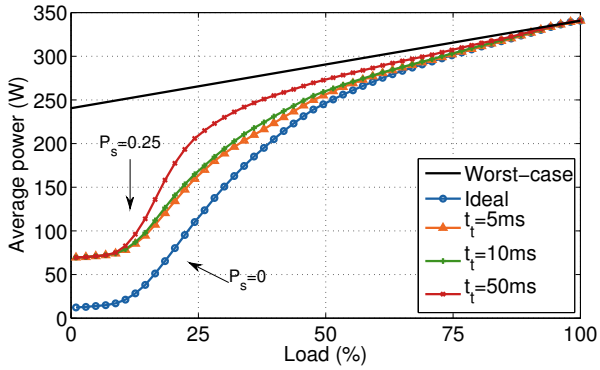
Based on the model described in the previous section, we assess power consumption of hybrid multicore architectures as a function of load and evaluate achievable energy savings and performance.

As mentioned, our prototype would not be adequate to achieve significant energy saving, because idle power is too high and Xeon performance is intentionally suppressed for the sake of interoperability.

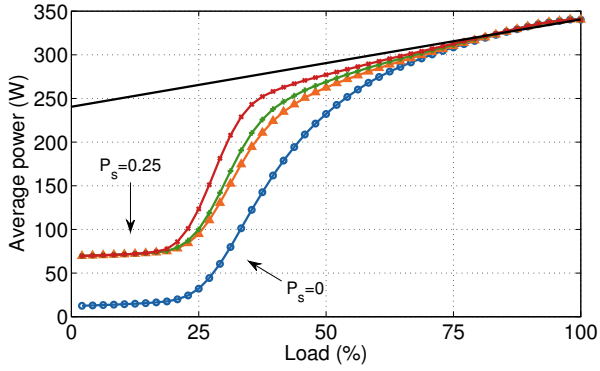
To evaluate achievable power consumption for an optimized board, we measured power consumption of an AtomN450 netbook, which exhibited the same 10 W dynamic range as the Atom 330 but lower idle power. It consumes 12.1 W with no load and 20.4 W at maximum load. Then, we consider a fully performing Xeon as large CPU, with power at peak of 320 W.

We also assume that the subsystem associated to the large CPU can be turned to a low-power standby mode in low-load time intervals, and is awakened by the scheduler only when needed.

Reasonably, the power consumed in a sleep state can be expressed as a fraction of P_{idle}. In the following we consider two cases, $P_s = 0$ and $P_s = 0.25 \cdot P_{idle}$. The first is an ideal case and is used, coupled with a zero transition time, to consider an ideal optimistic situation, whereas the second case is very conservative, and should represent a pessimistic case. Coordinated sleep states among components cannot be as fast as CPU sleep states (C-states) which have transition latencies in the order of microseconds. For this reason we consider transition times t_t up to tens of milliseconds.



(a) count



(b) compress

Figure 3.5: Average power consumption vs. system load for the *Count* (a), and *Compress* (b) workloads.

Figure 3.5 shows the average power consumption over the entire load range for the *count* 3.5(a) and the *compress* 3.5(b) workload.

The thick straight line represents the worst-case scenario, *i.e.*, a system that consumes the sum of idle power for the two subsystems at idle and the sum of peak power at maximum. Different transition times t_t are considered, corresponding to $t_t = 5, 10, 50$ ms.

Note how all the solutions presented exhibit good energy proportionality, compared to the worst case, thanks to the low energy of the small CPU subsystem and to efficient sleep states of the large CPU subsystem. When the load increases above 70% all the power curves converge towards the worst case scenario. This is due to

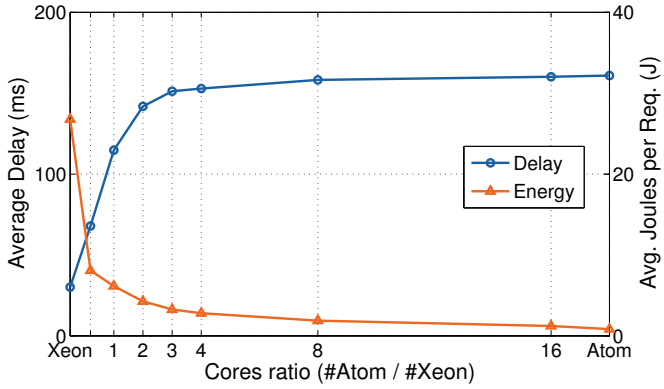


Figure 3.6: Average delay and average energy cost per request for different small to big cores ratio.

the lack of idle periods that the big CPU can exploit to undergo the transition to a sleep state.

The real advantage, in energy terms, is due to the fact that a typical server only spend a small fraction of time in high load region and has a low load for most of the time. In our solution power dissipation is small at low load, when only the small CPU subsystem is active. This come at the price of increased latency in servicing requests as we have shown in the previous section.

To better understand the latency vs. energy savings trade-off we can use our service delay model of Section 3.2 to evaluate several what-if scenarios. In particular the model allows us to explore the design space across two dimensions: i) the ratio of small to big cores, and ii) the relative performance of small to big cores.

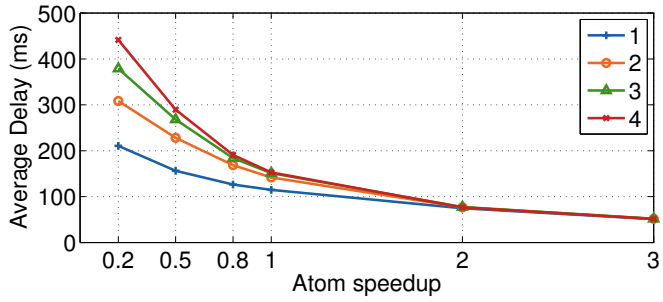
Along the first dimension, as the number of small cores increases, more requests will be served by small cores and the overall average service delay will be larger. On the other hand, energy usage will be lower because high power big cores can be idle longer.

Figure 3.6 shows the average service delay and average energy spent per request as a function of the ratio of number of Atom cores to Xeon cores. To derive this figure we used as input a request load that mirrors the utilization distribution reported in [60] for Google's datacenters. From the figure a designer can pick a desired average service latency (or energy per request) and obtain the best number of Atom cores to reach that performance. Furthermore, the model shows at what point adding more Atom cores has little impact on latency or energy usage. This

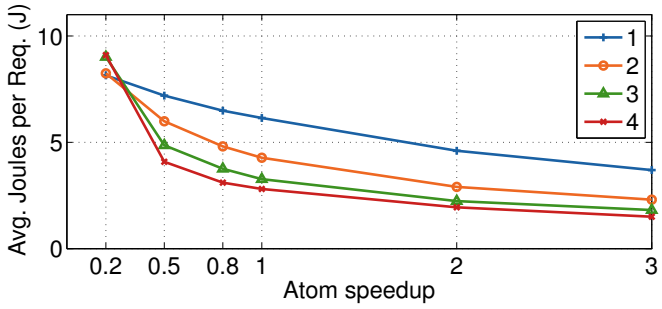
is expected as for those ratios, all requests are always served by the Atom cores and the Xeon cores stay always idle. The actual value is workload-dependent (a ratio of 8 in our case) and datacenter operators can estimate the load of requests and choose the best operating point accordingly.

The second dimension, the relative performance of cores, allows us to understand the impact of technology advances. Given the size and growth prospects of today's mobile and tablet markets, a significant development effort is dedicated to improve the performance of small low-power cores. 3.7(a) and 3.7(b) plot the average response delay and energy per request as a function of the "Atom speedup", *i.e.*, as we vary the performance of the Atom cores while keeping the Xeon cores the same. We also scale power usage linearly with performance in keeping with Atom's design constraint for which a feature is implemented only if it provides a "1% performance increase for less than 1% power increase" [61]. The workload is the same as before and follows the utilization distribution similar to [60].

Each line in the plots corresponds to a ratio of Atom vs. Xeon cores from one to four. The figures show two interesting trends. As the Atom performance drops below 0.5 of the current performance, there is very little energy advantage in using more Atom cores. This is because the processing time of the Atom cores is so large that more incoming requests need to be handled by the Xeon. As a consequence, Xeon cores have very few opportunities to transition to an idle state. At the other side of the spectrum, as the performance of the Atom cores increases adding more small cores has little impact on the response delay but helps in reducing energy per request significantly. If Atom cores were twice as fast as the cores in our hybrid prototype, the energy per request could be almost halved incurring a negligible service delay penalty.



(a)



(b)

Figure 3.7: Average delay (a) and average energy cost per request (b) for different slow-core performance (speedup is normalized to the performance of current Atom available on the market)

3.4 Related Work

The idea of hybrid or heterogeneous architectures is not new. At the datacenter level, several proposals have been put forward in the literature to include arrays of low-power systems next to high performance ones [36,53,81,96] that could handle low-load scenarios or specific I/O-intensive workloads. These proposals however require changing the way datacenters are designed and maintained and have seen little adoption by datacenters operators. Further, developers need to refactor their code to accomodate for the two different architectures that present very different memory and I/O bandwidth constraints. Hybrid systems that share the same memory banks and I/O subsystems enable the same codebase to achieve high single-thread performance when required and otherwise keep power consumption low.

At the system or processor level, asymmetric multicore processors (AMPs)

have been proposed as natural successors of chip multi-processors (CMPs) since they can provide energy efficiency in the execution of parallel threads, due to a large number of small cores, while improving performance of sequential phases through big cores. A large body of work in this field addresses the problem at the single-chip level [54, 85, 97, 98, 125] through theoretical analysis and simulations using synthetic workloads. We base our analysis on a real prototype akin to the one used by Reddy *et al.* [114]. Access to a working system allows for actual experimentation with real workloads as opposed to previous works that use FPGA-synthesized CPUs [124] or symmetric multiprocessors (SMP) architectures with cores tuned to work at different frequencies [57, 100].

Chapter 4

Energy-proportional Router

The network infrastructure is often viewed as an attractive target for energy-efficient design since routers are provisioned for peak loads but operate at low average utilization levels – *e.g.*, studies report network utilization levels of <5% in enterprises [109], 30-40% in large ISPs [75], and a 5x variability in ADSL networks [104]. At the same time, network equipment is notoriously inefficient at low load – *e.g.*, a survey of network devices [65, 103] reveals that the power consumed when a router is not forwarding *any* packets is between 80-90% of its peak power consumed when processing packets at full line rate. Thus although in theory networks offer significant opportunity for energy efficiencies, these savings are rarely realized in practice.

This inefficiency is increasingly problematic as traffic volumes continue their rapid growth and has led to growing attention in both research and industry [7, 16, 65, 75, 84, 103, 104, 109]. In anecdotal evidence: discussions with a major ISP revealed that infrastructure power consumption represents their second largest monthly recurring cost (second only to labor)[1] and we see calls for research on the topic from router vendors [8].

To date however, there have been no published reports on attempts to actually build an energy-efficient router. Motivated by this deficiency, we thus tackle the question of how one might build such an energy-efficient router.

The traditional challenge in building an energy efficient system lies in the inherent tradeoff between energy consumption and various performance metrics—in our case, forwarding rate and latency. We cannot compromise on peak forwarding

[1] Private commn. Stuart Elby, VP of Network Architecture, Verizon.

rates since the incoming traffic rate is dictated by factors external to a given router (and since we do not want to modify routing protocols). We can however explore options that tradeoff latency for improved efficiency. The ability to do so requires: (1) that the underlying hardware expose primitives for low-power operation and (2) higher-level algorithms that invoke these primitives to best effect. Our work focuses on developing these higher-layer algorithms, and we do so in the context of general-purpose server hardware.

General-purpose hardware typically offers system designers three 'knobs' for low-power operation: (i) regulate the frequency (and hence power consumption) at which individual CPU cores process work, (ii) put an idle core into a 'sleep' state (powering down sub-components of the idle core) (iii) consolidate packet processing onto fewer cores (adjusting the number of active cores).

As we shall see, not only do each of the above offer very different performance-vs-power tradeoffs, they also lend themselves to very different strategies in terms of the power management algorithms we must develop. For example, sleep-mode savings are best exploited by maximizing idle times which implies processing work as quickly as possible, while frequency-scaling is best exploited by processing work as slowly as possible which reduces idle times.

The question of how to best combine the above hardware options is, to our knowledge, still an area of active research for most application contexts and is entirely uncharted territory for networking applications. We thus start by studying the energy savings enabled by different hardware options, for different traffic workloads. Building on this understanding, we develop a unified power management algorithm that invokes different options as appropriate, dynamically adapting to load for optimal savings. We implement this algorithm in a Click software router [94] and demonstrate that its power consumption scales in proportion to the input traffic rate while introducing little latency overhead. For real-world traffic traces, our prototype records a 50% reduction in power consumption, with no additional packet drops, and only a small (less than 10 μs) cost in packet forwarding latency. To our knowledge, this is the first demonstration of an energy-efficient router prototype.

Before proceeding, we elaborate on our choice of general-purpose hardware as prototyping platform and how this impacts the applicability of our work. To some extent, our choice is borne of necessity: to our knowledge, current network hardware does not offer low-power modes of operation (or, at least, none exposed to third-party developers); in contrast, server hardware does incorporate such support, with standard software interfaces and support in all major operating systems.

That said, current network equipment may employ very different hardware options and hence we expand more concretely on the applicability of our work to each:

- Our work applies directly to software routers built on commodity x86 hardware, an area of growing interest in recent research [55, 74, 86, 102] with commercial adoption in the lower-end router market [43].
- Network appliances – load-balancers, WAN optimizers, firewalls, IDS, *etc.* – commonly rely on x86 hardware [5, 32] and these form an increasingly important component of the network infrastructure – *e.g.*, a recent paper [119] revealed that an enterprise network of $\sim$ 900 routers deployed over 600 appliances! Our work is likewise directly applicable to such appliances.
- Commercial routers typically rely on network processors (NPs) rather than general-purpose ones. While the specifics of our results may not apply directly to NPs, we expect that the overall methodology by which we compare and combine different power options will be of relevance. This is because the power modes that x86 hardware offers reflect fundamental technology options – turning cores on/off, frequency and voltage scaling, clock gating, *etc.* – and hence it is likely that NP designers will pursue similar approaches.[2]
- Many routers rely on specialized ASICs, particularly for simple low-level tasks such as checksum calculations; it is unclear what form of power modes ASICs will evolve to offer and hence we conservatively assume our work does not apply to these.
- Finally, we note a growing trend towards equipment that *augments* the traditional ASIC and NP-based platform with general-purpose CPUs in the form of 'beefed up' control planes (*e.g.*, Arista Networks), as one of multiple heterogeneous data-plane engines (*e.g.*, Cavium) or as 'service blades' on the data-plane (*e.g.*, Cisco's Application eXtension Platform [6]).

More generally, we believe our results can positively influence the hardware support for power management that emerges in specialized NPs and ASICs by demonstrating: what magnitude of energy savings are possible using basic hardware primitives; what performance tradeoff comes with these savings; and which of the feasible kinds of hardware primitives maximize benefits.

[2] NPs, like modern multi-core systems, use a large number of computation cores/engines executing network software in parallel [40].

The remainder of this chapter is organized as follows. We start with a review of the power consuption of current software routers (Section 4.1). We continue with techniques to reduce power by eliminating software inefficiencies (Section 4.2) and then study the tradeoffs between different power management options (Section 4.3). We present the design and implementation of our unified power management algorithm in Section 4.4 and the evaluation of our prototype in Section 4.5. We finish with an overview of related work.

4.1 Deconstructing Power Usage

A necessary step before attempting the design of a power proportional router is to understand the contribution of each server component to the overall power usage.

4.1.1 Server architecture

For our study, we chose an off-the-shelf server based on the Intel Xeon processor that is commonly used in datacenter and enterprise environments. The overall architecture is similar to that in servers from other manufacturers. Figure 4.1 shows a simplified diagram of the components that make up the server: our server has two CPU processors each of which consists of six cores packaged onto a single die along with several "uncore" components (L3 caches, memory controllers, power control unit, *etc.*).

The two processors are Xeon 5680 "Westmere" with a maximum clock frequency of 3.3 GHz. They are connected to each other and to the I/O hub via dedicated point-to-point links (called QuickPath Interconnect in our case). The memory (6 chips of 1 GB) is directly connected to each of the processors. We equipped the server with two dual-port Intel 10Gbps Ethernet Network Interface Cards (NICs). The I/O hub interfaces to the NICs (via PCIe) and to additional chipsets on the motherboard that control other peripherals. Other discrete components include the power supply and the fans.

From a power perspective, we consider only the components that consume a non-negligible amount of power: CPUs, memory, NICs, fans, and the motherboard. We use the term "motherboard" as a generic term to include components like the I/O Hub and PCIe bridge. It also includes other system peripherals not directly used in our tests: USB controller, SATA controller, video card, and so forth. The power supply unit (PSU) delivers power to all components using a single 12V

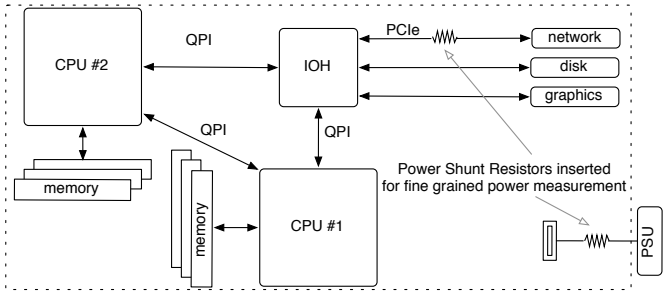


Figure 4.1: Server Architecture

DC line to the motherboard, which is in turn responsible for distributing power to all other subsystems (CPUs, fans, *etc.*).

We measure the current absorbed by the system by reading the voltage across 0.05Ω shunt resistors placed in series to the 12V line. This gives us the ability to measure power with high accuracy and high sampling frequency and excludes from the measurement the efficiency of the PSU, which is non-linear and can be quite low in certain cases.[3] Shunt resistors (hence individual power readings) are also used for the two 10Gbps NICs: the cards are connected using PCIe riser boards, with shunt resistors on the power supply lines.

4.1.2 Workload

Our server runs Linux 2.6.24 and Click [94] with a 10G Ethernet device driver with support for multiple receive and transmit queues. Using multiple queues, a feature available in all modern high speed NICs, we can make sure each packet is handled from reception to transmission by only one core without contention. We use Receive Side Scaling (RSS) [31] on the NIC to define the number of queues and, hence, the number of cores that will process traffic. RSS selects the receive queue of a packet using the result of a hash computed on the 5-tuple of the packet header. This way traffic is evenly spread across queues (and cores) with packets belonging to the same flow always processed by the same core. In the rest of the paper, when we refer to a number of cores n we also imply that there are n independent receive queues.

In the following sections, we first focus on the performance and power con-

[3]The PSU in our system is rated for 1400 W to have room for a large disk array, graphic cards, etc. Our setup needs instead 300W or less where the PSU efficiency is in the 60-70% range.

sumption of a single server operating as an IPv4 router with four 10G ports. Later, in Section 4.5, we consider more advanced packet processing applications such as NetFlow, AES-128 encryption and redundancy elimination [51]. For traffic generation we use additional servers that are set up to generate either synthetic workloads with fixed size packets or trace-driven workloads. We used two machines identical to our router server, each generator sends packets on two 10Gbps interfaces using multiple cores associated to multiple RSS queues in order to maximize the packet rate.

4.1.3 Power characterization

Isolating the power consumption of each component is useful to answer two different questions: (1) where does the power go in a modern server? (2) how does the power consumption vary with load?

The answer to the first question will let us identify which components are the largest consumers while the answer to the second question tells us how much we can potentially save by making that component more energy proportional. An ideal component to focus on is one that consumes a large fraction of the total power and whose consumption varies significantly with the input load.

We measure power in three sets of experiments: i) an idle system without Click running; ii) a system running Click with no input traffic; iii) a system running Click and forwarding 40 Gbps over four interfaces.

Given that we have only one aggregate power measurement for the entire server (plus one dedicated to the NICs), we need to perform several measurements to isolate each component. The system has eight fans, two CPUs and six memory chips. For each scenario we measure the power P with all components and then we physically remove one component (a memory chip, a CPU or a fan) and then measure the power P'. The difference $P - P'$ is then the power consumed by that component in that scenario.

Figure 4.2 shows the results of our experiments with IPv4 routing. At peak the system reaches 269 W – 2.3x the idle power consumption of 115 W. With Click running and no traffic the power consumption is 262 W, which is less than 3% lower than the peak power. Several design considerations stem from the results in Figure 4.2:

The CPUs are clearly the dominant component when it comes to power usage. Even when all cores are idle, they consume almost half of the total idle power,

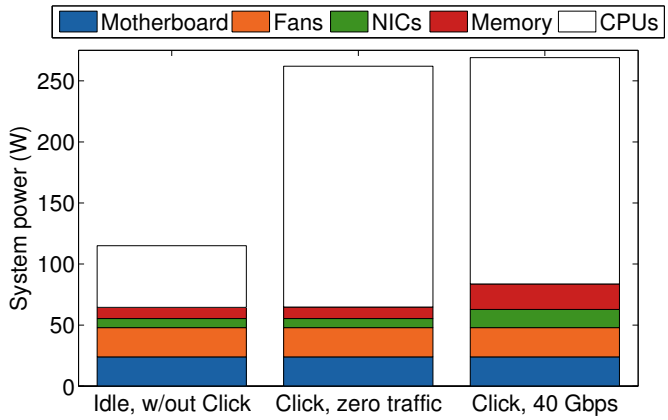


Figure 4.2: Breakdown of power consumption across various components when the system is idle and Click is not running (left), with Click running and no traffic (middle), and with Click forwarding 40 Gbps of traffic (right).

or $\approx$ 25 W each. That energy is mostly used to keep the "uncore" online. At peak, the CPUs reach $\approx$ 92 W each, which is about four times their idle power; this contributes significantly to the more than doubling of the total system power compared to idle.

The other system components contribute little to the overall power consumption. The motherboard, the component with the second largest power draw, reaches 24 W and it does not vary from idle to peak load. The memory and NICs exhibit a significant dynamic range – peak power is about twice the idle power – but the overall contribution is quite limited (20 W for the six memory chips and 15 W for the four 10 Gbps interfaces).

The system idle power is relatively high at 115 *W.* Even though this is an ongoing concern for system architects, the general trend is towards a reduction in idle power. Figure 4.3 shows the idle power of a number of systems since 2007 as reported by SpecPower [38]. The plot shows a clear downward trend in idle system power – halving in only 3 years! In the near future, this trend will continue by integrating more components into the CPUs and designing more energy efficient NICs [16] and memory chips [34].

Finally, *the software stack exhibits very poor power scaling*: the total power consumption is almost constant at zero and full load. Click continuously polls the

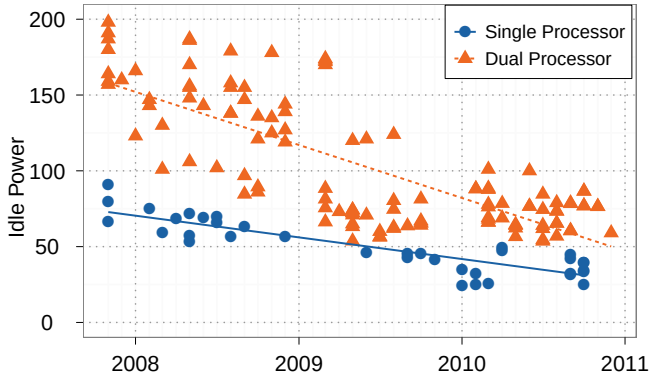


Figure 4.3: Server idle power over time (data from [38])

network interface to look for incoming packets. While this mechanism allows to easily achieve high throughput it also prevents the CPU from being idle even in absence of traffic.

In summary, our analysis indicates that to achieve energy efficiency we should focus on controlling the CPUs as they draw the most power and exhibit the largest dynamic range from idle to peak. In the following section, we explore the mechanisms available to software developers for controlling CPU power usage.

4.2 Addressing Software Inefficiencies

A pre-requisitive to exploiting low power states is to ensure that the software stack itself is efficient – *i.e.*, doesn't engage in needless work that prevents the creation of idle periods. We saw from Figure 4.2 that Click's polling architecture causes our server's power consumption to be the same at zero load (no packets) and at full bit rate. This section briefly describes how we fix this source of inefficiency.

Traditionally, operating systems and device drivers implemented packet processing via interrupt handlers: incoming packets generate an interrupt that is processed by the CPU. This approach impacts performance (each packet causes a context switch and pays the interrupt handling penalty), and is also prone to livelock at higher loads [108]. To get around this, Click disables interrupts on the network card and resort to *polling* the interface when the CPU is not busy. While this does provide higher performance, this also means the CPU does not have idle

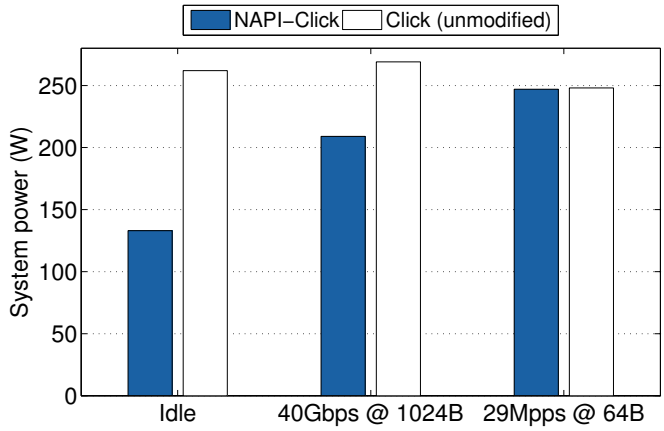


Figure 4.4: Power consumption of NAPI-Click vs Click in polling mode.

periods where it can be put in low power states.

The Linux NAPI packet processing framework addresses this problem with a hybrid approach [118]. Interrupts still wake up the CPU and schedule the packet processing thread. However, the packet processing routine disables interrupts and switches to polling mode while there are still packets in the input queue. After the queue is drained, or a maximum batch of packets has been processed, the routine ends and interrupts are re-enabled.

We modified the 10G driver and Click to operate in a similar fashion – we refer to the resultant stack as "NAPI-Click". Each receive queue of an interface raises a different interrupt that is directed to the core that is handling that queue. Figure 4.4 shows the power savings with NAPI-Click *vs.* unmodified polling-mode Click for different packet rates. We see that using NAPI reduces power consumption by 50% in idle mode and 22% when forwarding 40 Gbps of 1024B packets[4].

On the performance side, enabling interrupts has no impact on the maximum packet rate the router can forward – 29 Mpps with 64B packets. Another performance metric of interest is packet processing latency. Additional latency may be caused by the hardware rate-limiting interrupts and by the interrupt processing required before running the polling routine. To measure the latency, we ran

[4]A careful analysis of Figure 4.4 reveals that Click in pure polling draws more power with no traffic than at peak packet rate. In absence of traffic, the polling loop is quite short and the CPU keeps reading memory locations present in cache (no new packets have arrived). As soon as packets arrive, caches are invalidated triggering a sequence of cache misses. We conjecture that cores stall waiting for the memory reads to complete and, thus, do less work and draw less power.

experiments using a server with two network interfaces, forwarding packets from one to the other interface in a tight loop. A different machine acts as sender and receiver and timestamps outgoing and incoming packets using the hardware timestamp counter. In an unloaded system and without NIC-level batching, the latency is about 7 μs with polling and 12 μs with NAPI-Click. At high load, the latencies of the two methods tend to converge as NAPI-Click degenerates to polling at high load. We refer the reader to Section 4.6.1 for a more exhaustive description of our setup and analysis of the latency results.

To conclude, the hybrid interrupt/polling solution allows us to scale the power consumption between high load (when polling is used and power consumption is high), and low load (when switching to interrupts drives power consumption down for a negligible impact on latency).

4.3 Studying the Design Space

We now turn to exploring the design space of power-saving algorithms. A system developer has three knobs by which to control CPU power usage: i) the number of cores allocated to process traffic; ii) the frequency at which the cores run; and iii) the sleep state that cores use when there is no traffic to process.

The challenge is how and when to use each of these knobs to maximize energy savings. We first consider the single core case and then extend our analysis to multiple cores. Our exploration is based on a combination of empirical analysis and simple theoretical models; the latter serving to build intuition for why our empirical results point us in a particular direction.

4.3.1 Single core case

With just one core to consider, we are left with two design knobs and our design options boil down to one question: is it more efficient to run the core at a lower frequency for a longer period of time *or* run the core at a (relatively) higher frequency for a shorter period of time and then spend the remaining time in a sleep state?

To understand which option is superior, we compare two extreme options by which one might process W packets in time T:

- **the "hare"**: the core runs at its maximum frequency, f_{max} and then enters a low-power sleep state (C1 or below). This strategy is sometimes referred to as

"race-to-idle" as cores try to *maximize* their idle time.

- **the "tortoise"**: the core runs at the minimum frequency, f_x, required to process the input rate of W/T. I.e., we pick a core frequency such that there is *no* idle time. This is a form of "just-in-time" strategy.

To compare the two strategies we can write the total energy consumption of one core over the period T as:

$$E = P_a(f)T_a(W, f) + P_sT_s + P_iT_i, \tag{4.1}$$

where $T = T_a(W, f) + T_s + T_i$. The first term $P_a(f)T_a(W, f)$ accounts for the energy used when actively processing packets. $P_a(f)$ is the active power at frequency f and $T_a(W, f)$ is the time required to process W packets at frequency f. The second term considers the energy required to transition in and out of a sleep state (to store/retrieve core state, stabilize the clock, *etc.*). The third term is the energy consumption when idle.

With the *tortoise* strategy, the core runs at a frequency $f = f_x$ such that $T = T_a(W, f_x)$ and no idle time; hence:

$$E_{tortoise} = P_a(f_x)T_a(W, f_x) \tag{4.2}$$

With the *hare* strategy, $f = f_{max}$ and hence:

$$E_{hare} = P_a(f_{max})T_a(W, f_{max}) + P_sT_s + P_iT_i, \tag{4.3}$$

To facilitate comparison, let's assume (for now) that $T_s = 0$ (instantaneous transitions to sleep states) and $P_i = 0$ (an ideal system with zero idle power). Note that these assumptions greatly favor any *hare*-like strategy.

With these assumption the comparison between the *tortoise* and *hare* boils down to a comparison of their $P_a()T_a()$ terms in Equations 4.2 and 4.3. The literature on component-level chip power models tell us that the active power $P_a(f)$ for a component using frequency/voltage scaling grows faster than $O(f)$ but slower than $O(f^3)$.[5] The term $T_a(W, f)$ instead scales as $1/f$ in the best case.

[5]This is because power usage in a chip can be modeled as cV^2f where c is a constant that reflects transistor capacitance, V is the voltage and f is the switching frequency. As frequency increases voltage must also increase – larger currents are needed to switch transistors faster. Hence power scales at best linearly, at worst cubically with frequency. Note that this power model assumes the contribution from static power (such as leakage currents) is negligible.

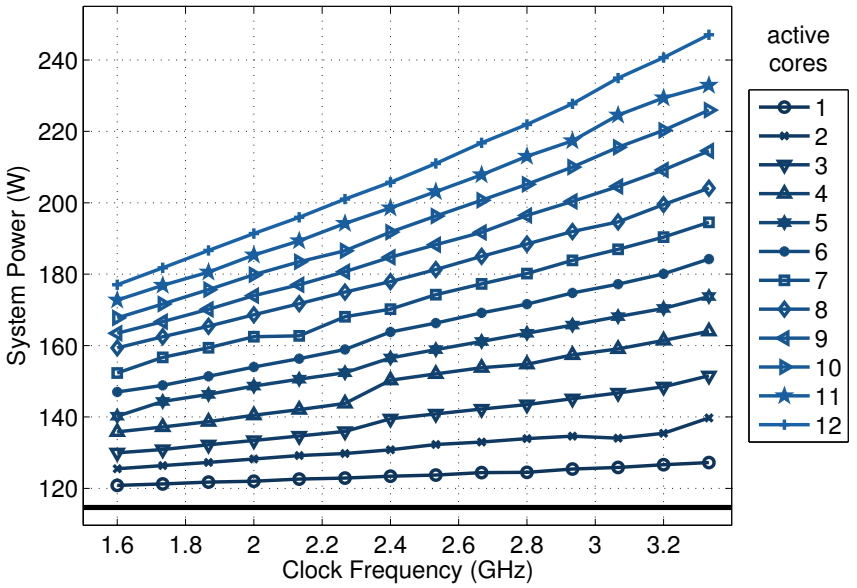


Figure 4.5: Power vs. Frequency at maximum sustained rate. The solid black line marks the system idle power.

Hence, putting these together, we would expect that $P_a(f_{max})T(W, f_{max})$ is always greater than $P_a(f_x)T(W, f_x)$, since $f_{max} \geq f_x$. Hence, despite our very '*hare* friendly' assumptions ($T_s = P_i = 0$), basic chip power models would tell us that it is better to behave like a *tortoise* than a *hare*.

Do experimental results agree with the above reasoning? To answer this, we use the same experimental setup as in Section 4.1 with a workload of 64B packets and plot results for our server using between 1 to 12 cores. We show results with more than just one core to ensure we aren't inadvertently missing a trend that arises with multiple cores; in all cases however we only draw comparisons across tests that use the same number of cores (*i.e.*, unlike the comparisons we draw in the following section).

We first look at how $P_a(f)$ scales with f in practice. Figure 4.5 plots the active power consumption, $P_a()$, as a function of frequency. For each data point, we measure power at the maximum input packet rate that cores can sustain at that frequency; this ensures zero idle time and hence that we are indeed measuring only $P_a()$. We see that $P_a()$ does not grow as fast as our model predicted – *e.g.*, halving the frequency leads to a drop of only about 5% in power usage with one

core, up to a maximum of 25% with twelve cores. The reason for this is that, in practice, many of the server's components, both within the CPU (*e.g.*, caches and memory controllers) and external to the CPU (*e.g.*, memory and I/O subsystems) are not subject to frequency and voltage scaling and this leads to lower savings than predicted.

Thus, active power consumption grows much more slowly with frequency than expected. What about $T_a(W, f)$? Since the processing time dictates the forwarding rate a core can sustain, we look at the forwarded packet rate corresponding to each of the data points from Figure 4.5 and show the results in Figure 4.6. Once again, we see that our model's predictions fall short: doubling the frequency does not double the sustainable packet rate. For example, with one core, doubling the frequency from 1.6 GHz to 3.2 GHz leads to an increase of approx. 70% in the forwarded packet rate (down to 45% with twelve cores). Why is this? Our conjecture is that the perfect $1/f$ scaling of processing time applies only to a CPU-intensive workload. Packet processing however is very memory and I/O intensive and access times for memory and I/O do not scale with frequency. Therefore, as we increase the frequency we arrive at a point where the CPU does its work faster and faster but it is then stalled waiting for memory accesses to complete, leading to a point where increasing the frequency no longer improves productivity.

In summary, we find that, $P_a()$ grows more slowly than expected with frequency but at the same time $T_a()$ decreases more slowly than expected. Where does this leave the product $P_a()T_a()$? We cannot directly measure this product (energy) and hence we look at efficiency in terms of packets-forwarded per Joule, obtained by dividing the maximum sustained forwarding rate (from Figure 4.6) by the power consumption (Figure 4.5). The result is shown in Figure 4.7 for increasing frequency. As before, this captures system efficiency while actively processing work (*i.e.*, there's no idle time). We see that, empirically, running at the maximum frequency is marginally more energy efficient in all configurations. That is, if our assumptions of $T_s = P_i = 0$ were to hold, then the *hare* would actually be marginally *more* power-efficient than the *tortoise* – counter to theoretical guidelines.

Of course, our assumptions are not realistic. In particular, we saw earlier (Figure 4.4) that P_i is quite high. Hence, the consumption due to the P_i and P_s terms in Equation 4.3) tilts the scales back in favor of the *tortoise*.

The final validation can be seen in Figure 4.8 where we plot the total power consumption for a fixed input packet rate at different frequencies. These are the first set of results (in this section) where we include idle and transition times. Based

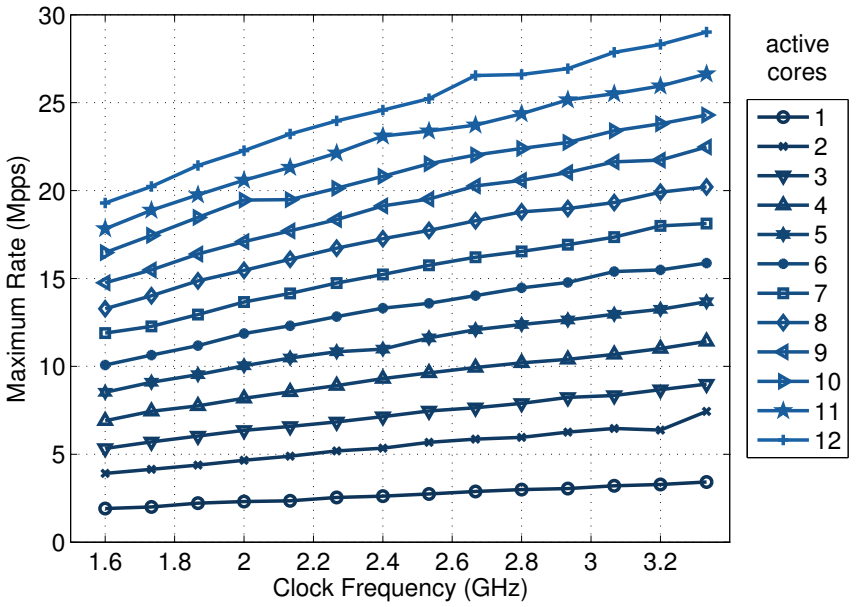


Figure 4.6: Packet rate vs. frequency at maximum sustained rate for different number of cores.

on our analysis from the following section, we make an idle core enter the C1 sleep state. We consider 1, 6 and 12 cores and, for each experiment, we fix the input packet rate to be the maximum rate the system can sustain at a frequency of 1.6 GHz; *i.e.*, at 1.6 GHz, there is no idle time and this corresponds to the *tortoise* strategy; the tests at 2.4 GHz and 3.3 GHz represent the (fast and fastest) *hare* strategy. It is clear from the figure that the configuration with the lowest frequency is always the most energy efficient.

Hence, in summary, the *tortoise* wins in keeping with what the theoretical power model would suggest. However this is *not* because the work is more efficiently processed at low frequency (as the power models would indicate) but because being idle is not sufficiently efficient (*i.e.*, the terms $P_i T_i$ and $P_s T_s$ are quite significant).

4.3.2 Multiple cores

We now consider the case where we have N cores to (as before) process W packets in time T. The results in the previous section show that it is more energy efficient

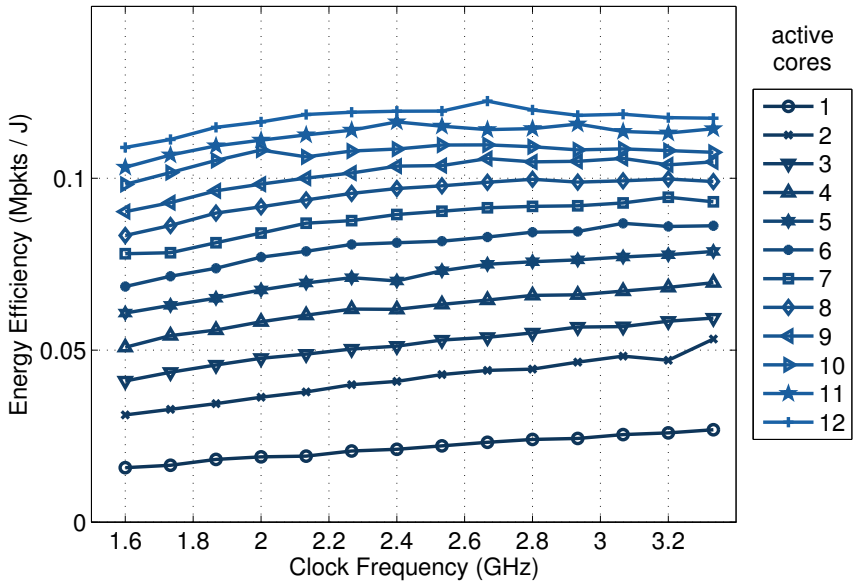


Figure 4.7: Processed packets per Joule varying the frequency and the number of cores.

to run an individual core at a low frequency and minimize idle time. Applying this to the multiple cores case would mean dividing the work – the W incoming packets – across multiple cores such that each core runs at a frequency at which it is fully utilized (*i.e.*, no idle time). In doing so, however, we must decide whether to divide the work across fewer cores running at a (relatively) higher frequency *or* more cores at a low frequency. Again, we first look to theoretical models to understand potential trade-offs.

Let us assume that a single core at a frequency f can process exactly W/k packets in time T. Then, our design choice is between:

- **"strength in speed":** use k cores at a frequency f. Each core processes W/k packets, sees no idle time, and hence consumes energy $kP_a(f)T(W/k, f)$.
- **"strength in numbers":** use nk cores at a frequency f/n. Each core now processes W/nk packets, sees no idle time, and hence consumes energy $nkP_a(f/n)T(W/kn, f/n)$.

As before, an idealized model gives $T(W/k, f) = T(W/kn, f/n)$ (since

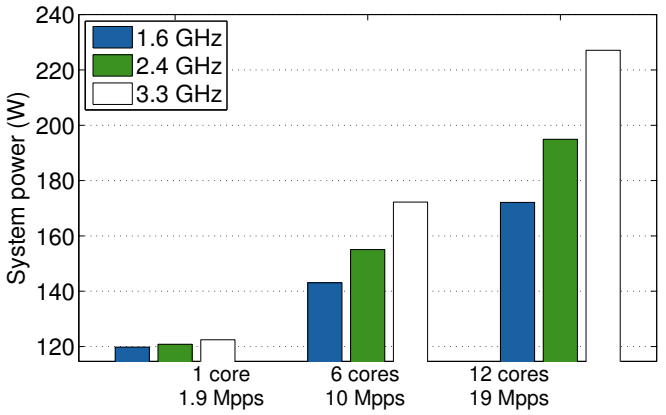


Figure 4.8: Power consumption of three frequencies for a constant input rate. At the lowest frequency (blue bar) the system has no idle time.

cores at f/n process $1/n$-th the number of packets at $1/n$-th the speed) and hence our comparison is between $kP_a(f)$ and $nkP_a(f/n)$. If the active power $P_a(f)$ grows faster than $O(f)$ (as the theory suggests) then the strength-in-numbers approach is clearly more energy efficient. This points us towards running with the maximum number of cores, each running at the minimum frequency required to sustain the incoming traffic.

Looking for empirical validation, we consider again the packets-forwarded per Joule but now comparing two sets of configurations: one with k cores running at frequency f and one with nk cores at frequency f/n. Unfortunately, since our hardware offers a frequency range between $[1.6, 3.3]$, we can only derive data points for $n = 2$ and $k = [1, 6]$ – the corresponding results are shown in Figure 4.9. From the figure we see that the empirical results do indeed match the theoretical guidelines.

However, the limited range of frequencies available in practice might raise the question of whether we can expect this guideline to hold in general. That is, what can we expect from m_1 cores at frequency f_1 *vs.* m_2 cores at frequency f_2 where $m_1 < m_2$ and $f_1 > f_2$? To answer this, we run an exhaustive experiment measuring power consumption for all possible combinations of number of cores (m) and core frequency (f). The results are shown in Figure 4.10 – for each $\langle m, f \rangle$ combination we measure the power consumption (shown on the Y-axis) and maximum forwarding rate (X-axis) that the m cores can sustain at a frequency f. Thus in all test scenarios, the cores in question are fully utilized (*i.e.*, with no

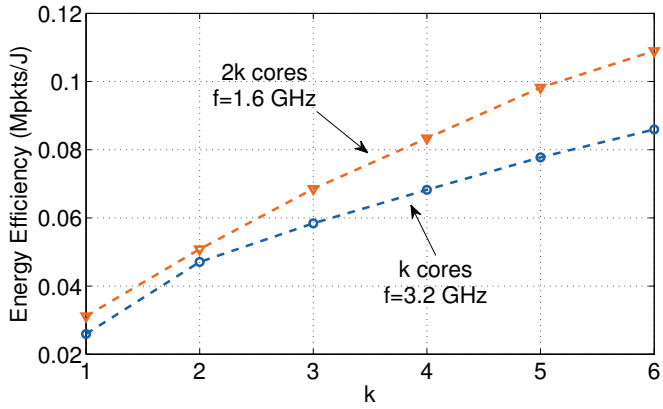


Figure 4.9: Comparing processed packets per Joule for k cores at frequency f and nk cores at frequency $\frac{f}{n}$.

idle times). We see that, for any given packet rate, it is always better to run more cores at a lower frequency, validating that "strength in numbers" is the preferred approach to exploiting multiple cores.

Applying this strategy under the practical constraints of a real server system however raises one additional problem. A naive interpretation of the strength-in-numbers strategy would be to simply turn on all N available cores and then crank up/down the frequency based on the input rate — *i.e.*, without ever worrying about how many cores to turn on. This would be reasonable if we could tune frequencies at will, starting from close to 0 GHz. However, as mentioned, the frequency range available to us starts at 1.6 GHz and, so far, we've only considered the efficiency of m cores that run *fully utilized* at any frequency. For example, if we consider the 12 core case in Figure 4.10 we only see the power consumption for data rates higher than approx. 19 Mpps (the max forwarding rate at 1.6 GHz). How would 12 cores fare at lower packet rates?

Before answering this question, we must first decide how to operate cores that are under-utilized *even at their lowest operating frequency*. The only option for such cores is to use sleep states and our question comes down to which of the three sleep states – C1, C3 or C6 – is best. The answer to this depends on the power-savings *vs.* transition-latency associated with each C state. We empirically measured the idle power consumption and average transition (a.k.a 'exit') latency for each C state – the results are shown in Table 4.1. We see that C3 and C6 offer only modest savings, compared to C1, but incur quite large transition times. This

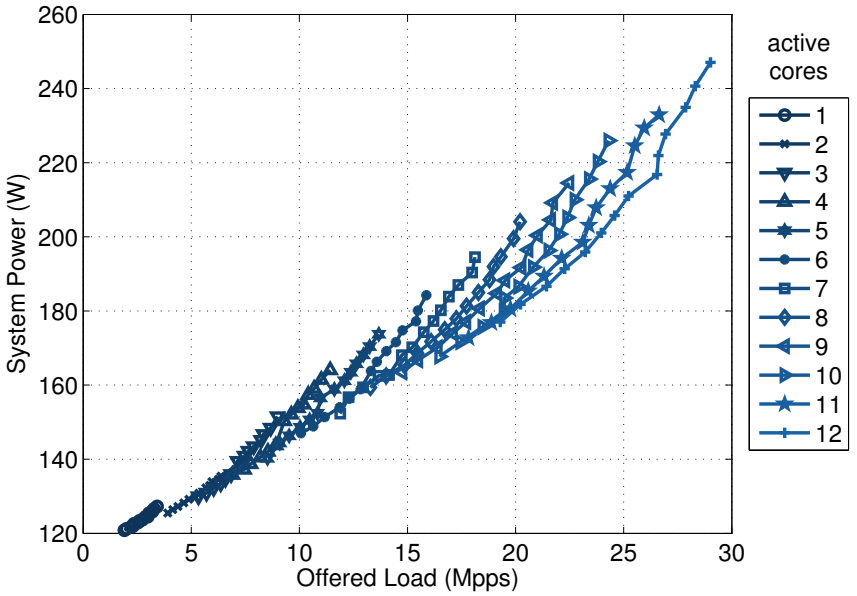


Figure 4.10: System power varying number of cores and operating frequency. Each point corresponds to the maximum 64B packet rate that configuration can sustain.

suggests that C1 offers the 'sweet spot' in the tradeoff.

To verify this, we measured the power consumption under increasing input packet rates, using a fixed number of cores running at a fixed frequency. We do three set of experiments corresponding to whether an under-utilized core enters C1, C3 or C6. Figure 4.11 shows the results for 2, 4, and 12 cores and a frequency of 1.6 GHz. We see that which C-state we use has very little impact on the total power consumption of the system. This is because, even at low packet rates, the packet-interarrival times are low enough (*e.g.*, 1 Mpps implies 1 packet every 1 μs) that cores have few opportunities to transition to C3 or C6. In summary, given its low transition time, C1 is the best choice for an under-utilized core.

Now that we know what an under-utilized core should do, we extend the results from Figure 4.10 by lowering the input data rates to consider the case of under-utilized cores. For clarity, in Figure 4.12 we plot only a subset of the data points. We see that, for any packet rate, the appropriate strategy is *not* to run with the maximum cores at the lowest frequency but rather to run with the maximum cores that can be kept *fully utilized*. For example, at 5 Mpps, the configuration with 3

sleep state	system power	avg. exit latency
C1	133 W	< 1 μs
C3	120 W	60 μs
C6	115 W	87 μs

Table 4.1: Power consumption and exit latency for different sleep states. See Section 4.6.2 for details on the experimental setup used to measure the above.

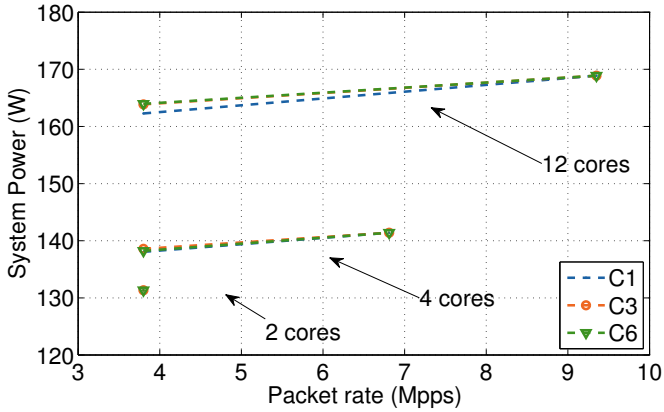


Figure 4.11: Comparing the impact of different C states on power consumption. All cores are running at 1.6 GHz.

active cores saves in excess of 30 W compared to the 12 cores configuration.

In summary, our study reveals three key guidelines that maximize power savings:

1. **strength in numbers:** the aggregate input workload should be equally divided between the maximum number of cores that can be kept fully utilized when processing their share of the workload.

2. **act like a tortoise:** each core should run at the lowest possible frequency required to keep up with its input workload (as opposed to running at higher frequencies and then going to sleep).

3. **take quick-n-light naps:** if a single core running at its lowest possible frequency is under-utilized, it should enter the lightest C1 sleep state (as opposed to deeper C3, C6 sleep states).

Following the above guidelines leads to the lower envelope of the curves in

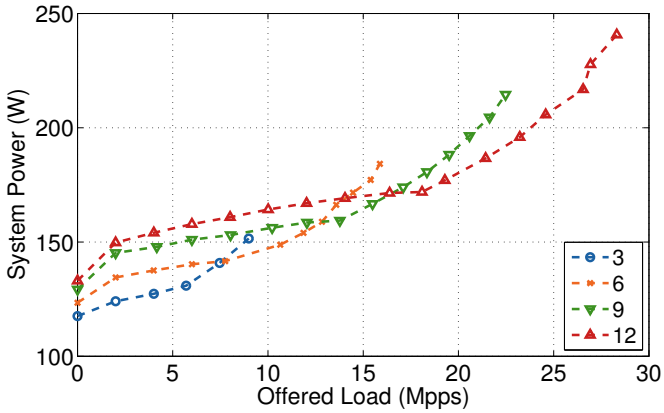


Figure 4.12: System power consumption varying the input data rate. Cores transition to C1 when underutilized.

Figure 4.10 which represents the optimal power-consumption at any given packet rate. In the following section, we describe how we combine these guidelines into a practical algorithm.

4.4 Implementation

Our empirical results tell us that a strategy that yields the maximum power saving is one that tracks the lower envelope of the curves in Figure 4.10. A simple implementation of that strategy could be to use a lookup table that, for any input data rate, returns the optimal configuration in terms of number of cores and frequencies. However, such an approach has two limitations. First, to work as promised, it requires perfect knowledge of the instantaneous data rate. Second, any change in the hardware — a new processor with more cores or a wider range of operating frequencies — would require comprehensive benchmarking to recompute the entire table for all possible configurations.

An alternative is to devise an algorithm that, while adhering to the guidelines of the previous section, tries to approximate the lower envelope of the curves with no explicit knowledge of those curves. An approach that maps quite well to the guidelines is the following iterative algorithm (that we will call “PowerSave”):

- Start with only one core active at the minimum frequency. All other cores are in C6 deep power down state (“inactive”). The NIC is instructed to send packets

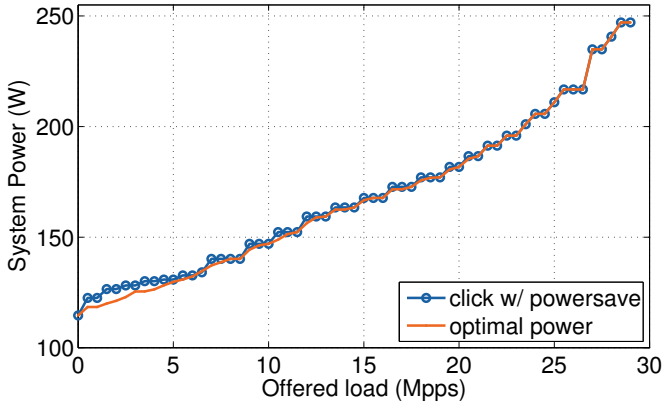


Figure 4.13: Power usage as a function of packet rate for our power saving algorithm and an optimal strategy.

only to the one active core.

- As the load increases – beyond what the active core can sustain – wake up one more core and split the traffic evenly among all active cores. If all cores are active, increase the frequency of all cores to keep up with the input data rate.

- If traffic load decreases, first lower the frequency of all cores and then start turning off one core at a time consolidating traffic onto the remaining active cores.

To understand how well this algorithm could approximate the optimal power curve we emulate its behavior by using the power measurement used to plot Figure 4.10. In Figure 4.13, we show the power usage for an optimal algorithm (*i.e.*, one that follows the lower envelope of Figure 4.10) and for our approximated power saving algorithm. The approximation closely tracks the optimal algorithm. At very low rate our algorithm chooses to turn two cores on much earlier than the optimal solution would. Even so, the additional power consumption is very small (below 5 W) given that the two cores always run at the lowest frequency.

Turning this algorithm into an actual implementation requires two mechanisms: i) a way of determining whether the current number of cores (and frequency) is the minimum required to sustain the input rate, and ii), a way of diverting traffic across more cores or consolidating traffic onto fewer cores. The remainder of this section describes our implementation of the two mechanisms.

4.4.1 Online adaptation algorithm

The goal of the adaptation algorithm is to determine if the current configuration is the minimum configuration that can sustain the input traffic rate. The algorithm needs to be able to quickly adapt to changes in the traffic load and do so with as little overhead as possible.

A good indicator of whether the traffic load is too high or low for the current configuration is the number of packets in the NIC's queues. Obtaining the queue length from the NICs incurs very little overhead as that information is kept in one of the NIC's memory-mapped registers. If the queues are empty – and stay empty for a while – we can assume that too many cores are assigned to process the incoming packets. If the queues are filling up instead, we can assume that more cores (or a higher frequency) are needed to process the incoming packets.

This load estimation is run as part of the interrupt handler to allow for a quick response to rate changes. Only one core per interface needs to run this estimator to decide when to wake up or put to sleep the other cores. We set two queue thresholds (for hysteresis) and when the number of packets is below (above) a threshold for a given number of samples we turn off (turn on) one core – or if all cores are already on increase/decrease the frequency. Listing 4.1 shows the pseudocode of the algorithm.

The parameters q_h and q_l are used to set the target queue occupancy for each receive queue. Setting low queue thresholds leads to smaller queueing delays at a cost of higher power usage. The choice of the most appropriate thresholds is best left to network operators: they can trade average packet latency for power usage. In our experiments we set $q_h = 32$ and $q_l = 4$ for a target queueing delay of ≈ 10 μs – assuming a core can forward at 3 Mpps.

The variables c_h and c_l keep track of how many times the queue size is continuously above or below the two thresholds. If the queue is above the q_h threshold for a sufficient time, the system is under heavy load and adds one more core or increases the frequency (if all cores are already active). Note that every time we change the operating point (number of cores or frequency), the counter is reset to give the system sufficient time to react to the change. Conversely, if we are under the q_l threshold for a sufficient time, we decrease the frequency or turn off one core if the frequency is already at the minimum. Once again, the counter is reset after a shift to give the system time to react. After adjusting the operating level, we process a batch of packets, and either repeat the loop, or halt (thus moving to state C1) if there are no more packets to be processed. In case of a halt, the next

```
for (;;) {
   // get a batch of packets and queue length
   qlen, batch = input(batch_len);

   // now check current queue length
   if (qlen > q_h) {
      // the queue is above the high threshold.
      c_l = 0;
      c_h++;
      if (c_h > c_up) {
         c_h = 0;
         <add one core, if all are on, raise frequency>
      }
   } else if (qlen < q_l) {
      // the queue is below the low threshold
      c_l++;
      c_h = 0;
      if (c_l > c_down) {
         c_l = 0;
         <decrease frequency, if at min remove one core>
      }
   } else {
      c_h = 0;
      c_l = 0;
   }
   // process a full batch of packets
   process(batch);

   if (qlen == 0) halt();
}
```

Listing 4.1: Pseudocode for the adaptation algorithm.

interrupt will resume processing with no further delays other than those related to interrupt generation and dispatching.

4.4.2 Assigning queues to cores dynamically

Our algorithm requires that the NIC be able to split the incoming traffic evenly between a variable set of cores. For this we use the support for multiple input and output queues that is present in all modern NICs. The mechanism we use is called Receive Side Scaling (RSS) [31] and operates as follows.

For each incoming packet, the NIC computes a hash function on the five-tuple

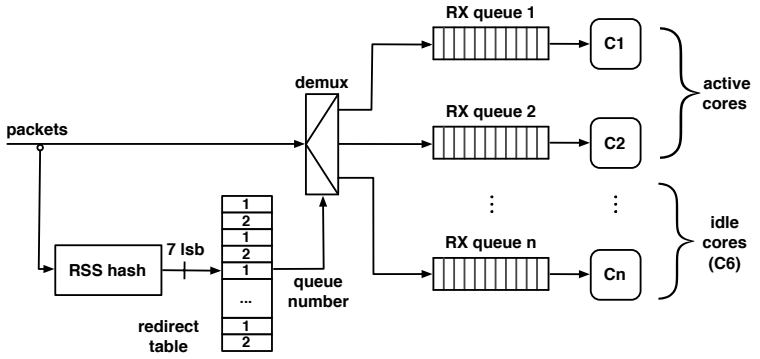


Figure 4.14: NIC queues and redirect table. Hashes are computed on the 5-tuple of each packet, and used to address one of the 128 entries of the redirect table. Each entry in the table contains a queue number.

(source and destination IP, source and destination port, and protocol number) of the packets. The 7 least significant bits of the hash are used as an index into a *redirect table*. This table holds 128 entries that contain the number of the queue to be used for packets with that hash, as shown in Figure 4.14. When a packet is placed in its corresponding queue an interrupt is raised. Each queue has a dedicated interrupt that we can program to be directed to any core.

The number of queues on a NIC can only be set at startup. Any change in the number requires a reset of the card. For this reason we statically assign queues to cores as we start Click. To make sure inactive cores do not receive traffic, we modify the redirect table by mapping entries to queues that have been assigned to active cores. We always keep the entries in the table well balanced so that traffic is spread evenly across cores that are active and processing packets. The table is only 128 bytes long, so modifications are relatively fast and cheap (considering that we do not modify it too often).

If a core does not receive traffic on its queue it remains in deep power down. When additional cores are needed to process traffic, we reconfigure the redirect table to include the additional queue. The update to the redirect table has almost immediate effect; this is important because it immediately reduces the load on the active cores. As soon as the first packets arrives to the new queue, the interrupt wakes up the corresponding core, which eventually (after the exit latency) starts processing the traffic.

Taking out a core is equally simple: we reprogram the redirect table so that no

new packets will reach the extra core. Also, we instruct the core to enter a C6 state on `halt()` instead of C1. Eventually, the queue will become empty, and the core will enter C6 from which it does not exit until the queue is enabled again.

4.5 Evaluation

To evaluate the performance of our prototype router in a realistic setting we generate traffic using two packet traces: the "Abilene-I" trace collected on the Abilene network [26] and a one day long trace from the "2009-M57 Patents" dataset [23] collected in a small-enterprise 1Gbps network. The former contains only packet headers while the latter includes packet payloads as well.

Our generator software is composed of many traffic sources, each reading from a copy of the original traces. By mixing multiple traffic sources we can generate high bit rates as well as introduce burstiness and sudden spikes in the traffic load. This helps in testing the responsiveness and stability of our power saving algorithm.

We are interested in evaluating the performance of our system with a broad range of applications. To this end, we run experiments with the following packet processing applications: IPv4 routing, a Netflow-like monitoring application that maintains per-flow state, an IPSEC implementation that encrypts every packet using AES-128 and the Redundancy Elimination implementation from [51] which removes duplicate content from the traffic. Note that we use all applications *as-is*, without modifications from their original Click-based implementations.

This set of applications is quite representative of typical networking applications. Routing is a state-less application where each packet can be processed independently of the other. Netflow is stateful and requires to create and maintain a large amount of per-flow state. IPSEC is very CPU intensive but stateless. Finally, Redundancy Elimination is both stateful (keeps information about previously observed packets) and CPU intensive (to find duplicate content in packet payloads).

We generate similar input traffic profiles for all applications. Figure 4.15 shows the traffic load over time for the different applications. We tune the traffic profiles so that the average utilization of the router over the duration of the experiment is around 35%. This results in different bit rates for different applications as they differ in per-packet processing cost (with Redundancy Elimination being the most demanding). The load is evenly distributed across the four interfaces and the routing table is uniform so that no output interface has to forward more than the 10 Gbps

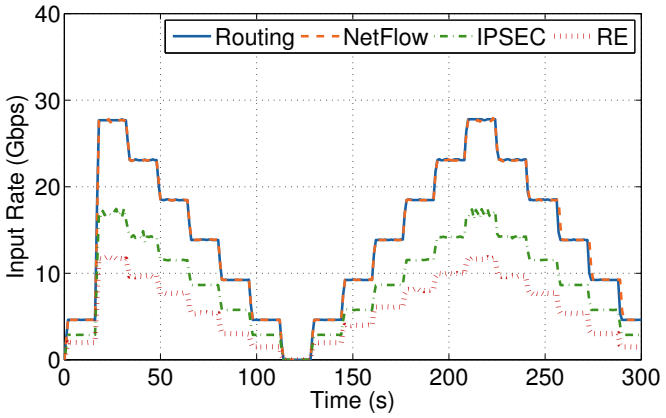


Figure 4.15: Input traffic profile for different applications.

it can handle.

We first focus on the power consumption of the various applications and then measure the impact of *PowerSave* on more traditional performance metrics such as latency, drops and packet reordering.

Power consumption.

Figures 4.16[a-d] show the power consumption over time for the four applications under study. In the figures we compare the power consumption of the default Click implementation, the NAPI-Click and Click with PowerSave. As expected, unmodified Click fares the worst with a power consumption hovering around 270 W for all applications at all traffic rates. The power usage with NAPI-Click and PowerSave instead tracks, to varying degree, the input traffic rate.

PowerSave is consistently the most energy efficient at all rates for all applications. The advantage of PowerSave is more prominent when the load is fairly low ($10-20\%$) as that is when consolidating the work across cores yields the maximum benefits. Over the duration of the experiments, PowerSave yields an overall energy saving between 12% and 25% compared to NAPI-Click and 27-50% compared to unmodified Click.

Traditional performance metrics. We turn our attention to other metrics beyond power consumption, namely: packet loss, latency and reordering. Our algorithm

may introduce loss or high latency by turning on cores too slowly in the face of varying traffic demands. Reordering may be introduced when packets belonging to the same flow are redirected to a different core.

Regarding packet losses, the PowerSave algorithm reacts quickly enough to avoid packet drops. Indeed, no packet losses were observed in any of our experiments.

We also measured the latency of packets traversing the router. Figure 4.17 plots the average, minimum and 99th percentile over each 1s interval – we collect a latency sample every 1ms. We only plot results for one PowerSave experiment with IPv4 Routing. Other experiments show similar behavior and are reported in Section 4.6.1.

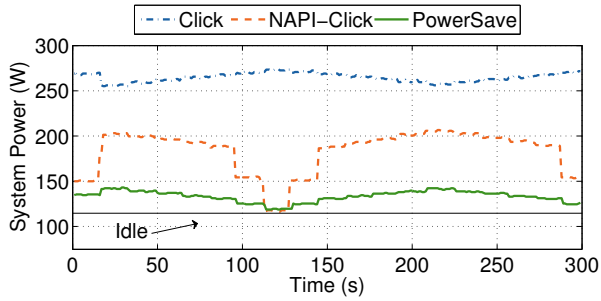
The average latency hovers in the $15 - 20$ μs range. The same experiments with Click unmodified yield an average latency of 11 μs. The 99th percentile of the latency is up to 4 times the average latency – it peaks at 55 μs. Some of the peaks are unavoidable since waking up a core on from C6 may introduce a latency of up to 85 μs. However, that is limited to the first packet that reaches the queue handled by that core.

As a last performance metric we also measured packet reordering. In our system, reordering can only occur when traffic is diverted to a new queue. Hence, two back to back packets, A and B, belonging to the same flow may be split across two queues and incur very different latency. Given that new queues are activated only when the current ones start to fill up, it is possible to have packet A at the back of one queue while packet B is first in line to be served on the new queue. On the other hand, packets in the newly activated queue will not be processed until the core assigned to it exits a C6 state. Given that in our setting the adaptation algorithm aims for ≈ 10 μs of queueing delay, it is quite likely that packet A will be served while the new core is still exiting C6. We confirmed this conjecture in our experiments. We have measured zero packets being reordered. To consider an extreme case of very high bandwidth flows we even measured reordering of all packets, independently of the flow they belong to, and still found none.

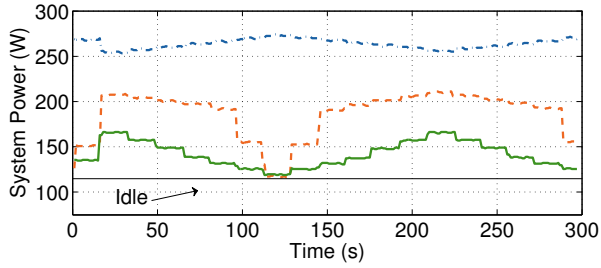
Impact of traffic burstiness. Our results so far were based on input traffic whose fine-grained "burstiness" (*i.e.*, over short timescales) derived from the natural variability associated with a software-based traffic source and multiplexing traffic from multiple such sources. To study the impact of fine-grained burstiness in a more controlled manner, we modified the traffic sources to generate traffic following an

on/off pattern. During an "on" period the traffic source generates a burst of back-to-back packets at the maximum rate. We thus control the burstiness of the traffic by picking the length of the "on" period (*i.e.*, the number of back-to-back packets) while keeping the average packet rate constant.

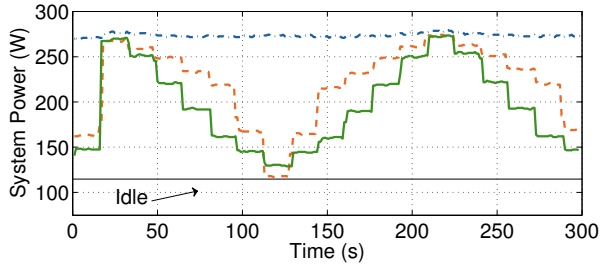
Figure 4.18 shows the power usage of our system as we vary the burstiness of the sources (a value of 1 corresponds to uniform traffic). The average traffic rate is around 2.5 Gbps per interface. In the interest of space we only plot the results for IPv4 Routing. Other applications exhibit a similar behavior (see Section 4.6.3). As we can see the power usage varies very little as we increase the burstiness. This shows that the controller is stable enough to make sure large short-term burstiness does not impact its performance. As expected, the power usage with unmodified Click is flat while NAPI-Click benefits from traffic burstiness as interrupts are spaced out more with burstier traffic. However, for all levels of burstiness, PowerSave is clearly the most energy efficient. We measured also latency and packet loss and saw similar results where burstiness has little or no impact. We refer the reader to Section 4.6.3 for a detailed analysis of the latency results with bursty traffic.



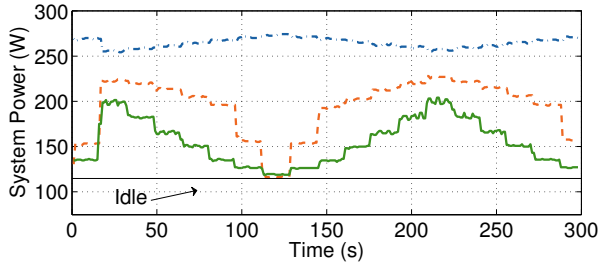
(a) Routing



(b) NetFlow



(c) IPSEC



(d) Redundancy Elimination

Figure 4.16: Power usage with (a) IPv4 routing, (b) NetFlow, (c) IPSEC, (d) Redundancy Elimination.

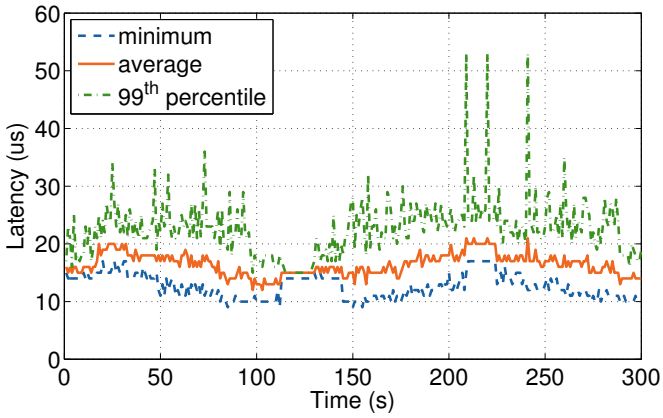


Figure 4.17: Average, minimum and 99th percentile of the packet forwarding latency when using Click with the power save algorithm.

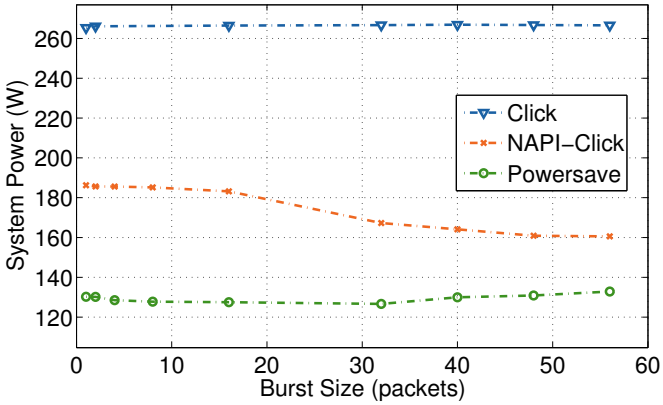


Figure 4.18: System power varying the burstiness of the input traffic. The average traffic rate is constant at 2.5 Gbps per interface

4.6 Testbed setup and Additional measures

4.6.1 Measuring Latency

To measure the packet latency introduced by our router we run experiments using a server with two interfaces. The server is configured to forward packets between the two interfaces in a tight loop. A different machine acts as a sender and receiver and timestamps incoming and outgoing packets. Packets are timestamped with the current value of the 64-bit TimeStamp Counter register (TSC) – counting the number of elapsed clock cycles. In order to measure the latency introduced by the server we first measured the baseline round trip time on the generator machine by replacing our server with a cable. This gives us an estimate of the delay due to packet generation and timestamping. We then run the same experiments replacing the cable with our software router and we then remove the baseline from all the measures. We measure the latency in processing a packet as a function of the packet rate for 64B packets.

Figure 4.19 shows the results for the legacy Click stack and for the NAPI-Click one while the input rate is increased up to the maximum the router can sustain. When the system is underloaded the latency is about 7 μs with polling and 12 μs with NAPI-Click. When the traffic increases to 1 Mpps and beyond the two methods tend to converge. Recall that with NAPI the interrupt handling routine disables the interrupts and polls the interface until there are packets left in the queue. The number of interrupts is function of the incoming packet rate, where more packets means that less interrupts are received improving the measured latency. At very high load there is no opportunity to re-enable interrupts and NAPI degenerates into polling, as can be seen from the latency measured at the maximum sustainable rate.

4.6.2 Sleep states

In Section 4.3 we summarized the power and performance implications of using sleep states (C-States) for inactive cores, here we provide detailed measures of the power consumption and the exit latency for each sleep state supported by our platform. Figure 4.20 shows the system power consumption measured while keeping all the cores in the same state. Our machine supports C1, C3 and C6 states in addition to the active C0 state. In order to keep cores in C0, each of them executes an empty infinite loop. We also run cores at the minimum available frequency (1.6 GHz), in this way we measure the lowest possible power consumption while all the cores are utilized at 100%, i.e. 171 W. In C1 instead the power consumption

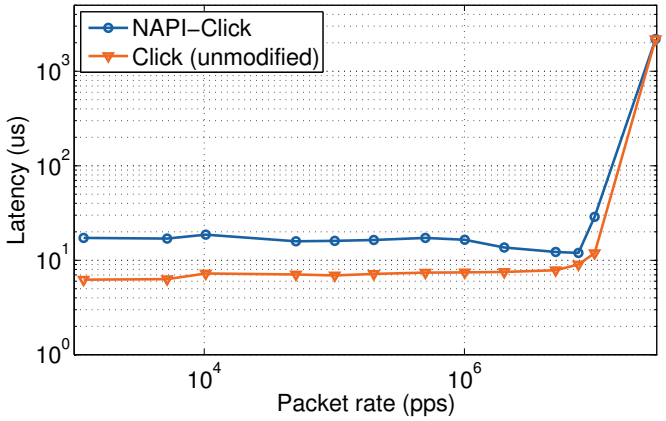


Figure 4.19: Packet processing latency with polling and interrupts varying the input traffic rate.

is 133 W, that is 78% of the power in C0.

With all cores in C3 the system can shed an additional 14 W compared to keeping all cores in C1. When all cores are in C6 the system power usage is down to 115 W, that is 86% of the power consumed when all the cores are in C1. These power savings are evenly distributed across cores.

To estimate the exit latencies we measure the delay of packets through a software router using the same setup described in Section 4.6.1. The router is driven by traffic at a very low packet rate (1 packet/sec) so that when an interrupt arrives, the core must first exit from its C-state (C1, C3, C6 depending on the experiment) and then packet processing can begin. Due to the low packet rate, cores have plenty of time to return to a deep power down state before the next interrupt.

Figure 4.21 plots the packet delay (minimum, maximum, average) depending on the C-state. The forwarding latencies in C0 and C1 are practically the same. This validates that the exit latency from C1 is comparable to a cache miss. Waking up from a deep power down state instead requires a longer time: 60 μs for C3 and about 100 μs for C6.

The variability in packet latency is due to the fact that a core can wake from a sleep state in order to process interrupts issued by peripherals other than NICs, or to run operating system timer handlers. At the time a new packet arrive it is possible that the core has already started the wakeup transition, so the packet experiences a low latency. In the worst case instead a packet arrives immediately

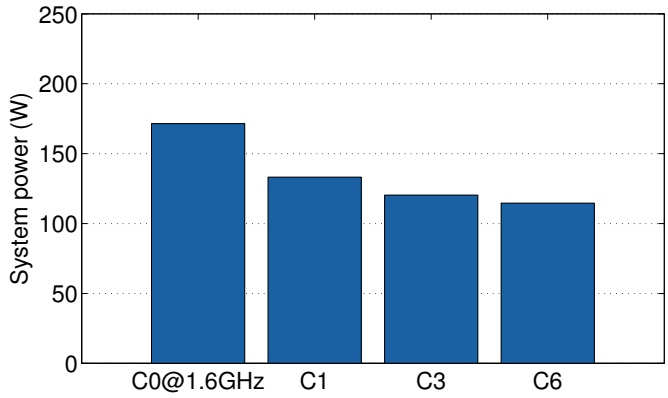


Figure 4.20: Power consumption with no load and varying the C-states.

after the core has started a transition to enter the sleep state, thus paying both the time to make the core sleep and the time to wake it up.

4.6.3 Traffic Burstiness

In Section 4.5 we have briefly analyzed how power consumption is affected by the input traffic burstiness and we reported results only for a simple IPv4 routing workload. Here we report additional results for all the applications analyzed in the evaluation section.

In order to assess performance while varying traffic burstiness at a fine grained level we have modified our traffic generators to aggregate packets in bursts while keeping the overall packet rate constant. The traffic generator is an ON/OFF generator that batches packets during the OFF period and sends them all at once over the wire in the ON period.

In all the experiments the rate is fixed to 2.5 Gbps per interface, a rate that all the applications we consider can sustain. The packet sizes and payloads are generated as explained in Section 4.5. We show results for NetFlow, IPSEC and Redundancy Elimination applications.

Figures 4.22[a-c] show the system power consumption for Click, NAPI-Click and our PowerSave solution while varying traffic burstiness. There is no surprise in Click behavior, showing an almost constant power in all the configurations. The common trend we notice is that NAPI benefits from a high burstiness in traffic since this makes subsequent interrupts more spaced in time. Our power saving

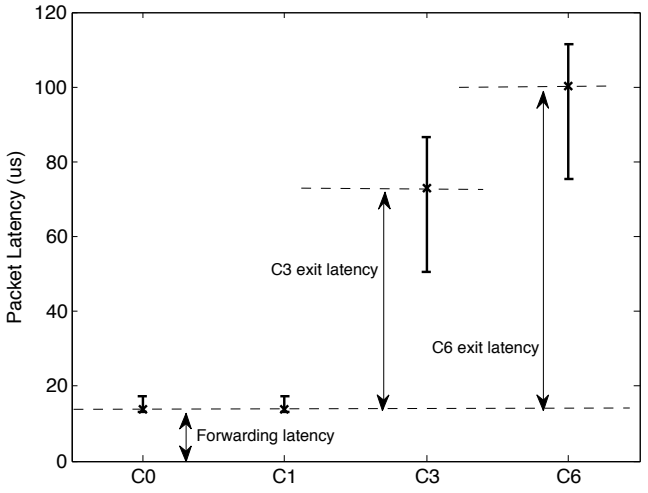
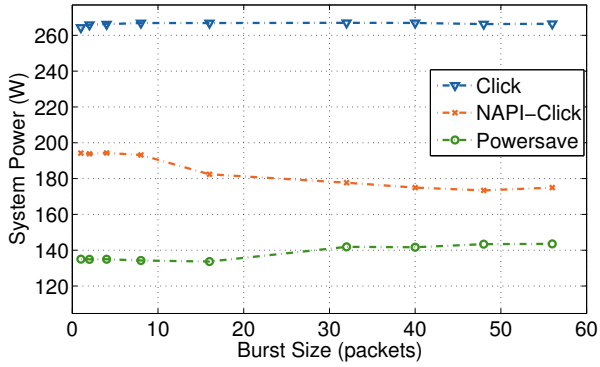


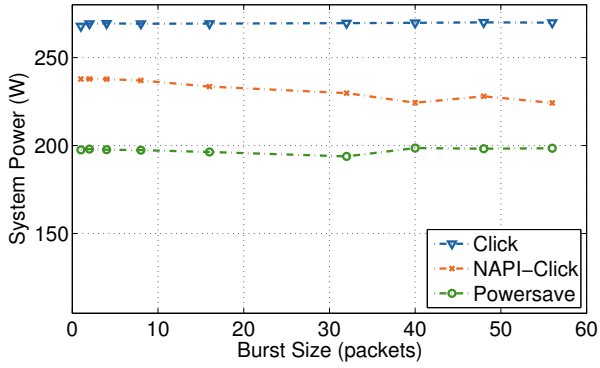
Figure 4.21: Packet processing latency when the core is in C1, C3, and C6 idle state. Error bars show the minimum and maximum latency while the cross shows the average latency.

algorithm instead benefits from bursty traffic only for values lower than 32. For higher levels of burstiness the average power slightly increases, due to the controller switching to higher performance configurations as soon as a bust arrives. Nonetheless the powersaving controller is able to respond quickly to incoming bursts while keeping the system power consumption at minimum in all configurations.

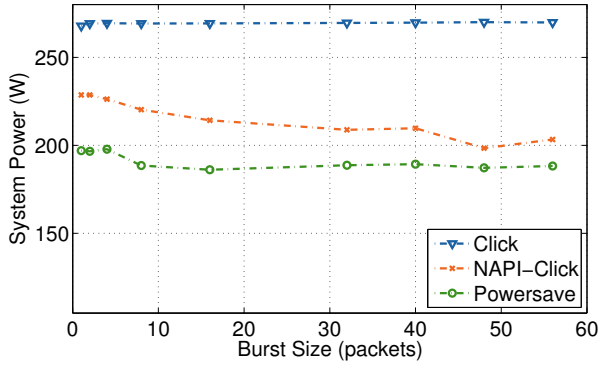
The side effect of bursty traffic is that it increases the overall packet latency by quickly filling up the input queues. Figure 4.23 shows the average latency for IPv4 routing and for NetFlow while varying the generator burstiness level. The powersave controller is running and we can see how it make the latency increase slower than linearly since it tries to keep the input queue occupancy between the defined thresholds.



(a) NetFlow



(b) IPSEC



(c) Redundancy Elimination

Figure 4.22: System power varying the burstiness of the input traffic while running different workloads.

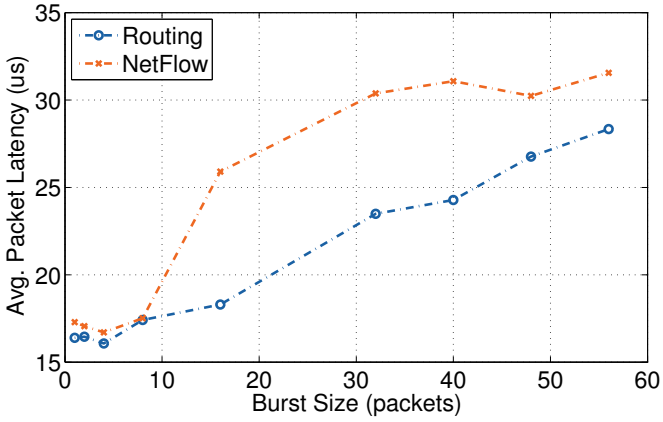


Figure 4.23: Average packet latency for IPv4 Routing and NetFlow while varying the input traffic burstiness.

4.7 Related Work

We discuss relevant work in the context of both networking and computer systems.

Energy efficiency in networks. Prior work on improving network energy efficiency has followed one of two broad trajectories. The first takes a network-wide view of the problem, proposing to modify routing protocols for energy savings. The general theme is to re-route packets so as to consolidate traffic onto fewer paths. If such re-routing can offload traffic from a router entirely, then that router may be powered down entirely [65, 88].

The second line of work explores a different strategy: to avoid changing routing protocols (an area fraught with concerns over stability and robustness), these proposals instead advocate changes to the *internals* of a router or switch [52, 84, 109, 115]. The authors assume hardware support for low-power modes in routers and develop algorithms that invoke these low-power modes. They evaluate their algorithms using simulation and abstract models of router power consumption; to date, however, there has been no empirical validation of these solutions.

Our focus on building a power-proportional prototype router complements the above efforts. For the first line of work, a power-proportional router would enable power savings even when traffic cannot be *entirely* offloaded from a router. To the second, we offer a platform for empirical validation and our empirical results in Section 4.3 highlight the pitfalls of theoretical models.

Energy-efficient systems. With the rise of data centers and their emphasis on energy-efficiency, support for power management in servers has matured over the last decade. Today's servers offer a variety of hardware options for power-management along with the corresponding software hooks and APIs. This has led to recent *systems* work exploring how to best leverage this hardware support with the aim of achieving *power proportionality* while actively processing a particular workload.

Many of these efforts focus on cluster-level solutions that dynamically consolidate work on a small number of servers and power-down the remaining [67, 87, 120]. Such work is orthogonal to ours as their techniques are not applicable to our context.

Closer to our focus is recent work looking at single-server energy proportion-

ality [91, 101, 106, 107, 121]. Some focus on improved hardware capabilities for power management [91, 101]. The remaining [106, 107, 121] focus (like us) on leveraging existing hardware support, but do so in the context of very different application workloads.

Tsirogiannis *et al.* focus on database workloads and evaluate the energy efficiency of current query optimizers [121]. The authors in [106] focus on non-interactive jobs in data centers and advocate the use of system-wide sleep ("S states" described in Section 2.2) during idle times. They use analytical models and simulations to demonstrate the potential power savings. This approach, however, assumes relatively long periods (on the order of tens of milliseconds and greater) of idleness across memory, I/O, and CPUs. As such, their results (and S-states more generally) are not applicable to typical network equipment that, even if lightly utilized, is never completely idle. More recently, Meisner *et al.* [107] explore power management trade-offs for online data-intensive services such as web search, online ads, *etc*. This class of workloads (like ours) face stringent latency requirements and large, quick variations in load. Using benchmarks, the authors in [107] derive analytical models to study the latency tradeoffs of power management options, based on which they offer general recommendations for hardware designers of future energy-efficient architectures.

In contrast to the above, we focus on networking workloads and exploiting low-power options in current hardware. The result of our exploration is a lightweight, online, power saving algorithm that we validate in a real system. We leave it to future work to generalize our findings to application workloads beyond networking.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

In this thesis we proposed solutions for improving the energy efficiency of datacenter facilities. The problem has been tackled on two sides, respectively the computing devices and the networking devices.

First we proposed a hybrid multiprocessor server architecture that is able to exploit the advantages of diverse elaboration units; the system we envision blends mobile processors with high-end ones and redirects the incoming requests to the right processor in order to preserve as much energy as possible. This is an explorative work since architectures like this are not commercially available yet. After our initial analysis through analytical models and simulation we had the luck to work with an early prototype, from Intel, that mounts a low-end Atom processor with a high-performance Xeon. The system, even if built with a legacy shared front-side bus design, could efficiently run a Web workload and gave us the opportunity to get some preliminary measures on which we founded our successive studies. We developed a simple but effective analytical model to describe the timing and energy characteristics of hybrid architecture designs. The model was used to evaluate performance and power savings achievable with future hybrid servers, showing the implications of the ratio of big and small cores in building more power-proportional servers. We believe our work can be useful for hardware designers to better understand the tradeoffs involved with hybrid architectures and to finely tune the performance and energy operating point of devices.

We then approached the problem of improving the power efficiency of network

devices without compromising their performance. For this, we studied the power-*vs*-performance tradeoffs offered by different approaches to low-power operation, in the context of devices based on commodity server hardware. We studied the saving potentials offered by different mechanisms such as C-states, P-states and number of cores, and studied the impact of these mechanisms on performance and latency. Our study revealed three guiding principles to optimally combine different hardware power management options: (i) run the system at the lowest possible frequency (P-state) that can keep up with the load, (ii) use as few cores as possible, and (iii) keep the other cores in the lowest possible sleep state (C-state). Based on these guidelines we build a prototype Click-based software router whose power consumption grows in proportion to the offered load, using between 50% and 100% of the original power, without compromising on peak performance. The general-purpose hardware we used in our analysis is very different from dedicated hardware and accelerators in today's network devices. We have extensively discussed the applicability of our work in Chapter 4 and we believe that the need for energy saving algorithms in networks will stem from the trend toward using commodity devices in large datacenter networks. Also, we expect that the overall methodology by which we compare and combine different power options will be of relevance, given that the sleep and throttle mechanisms we rely on reflect fundamental techniques in CMOS transistor technology.

5.2 Future Work

Future Operating Systems and programming languages runtime will need to support compute cores heterogeneity. With big (fast) and slow (small) cores available, the programmer will need to be aware of performance and energy asymmetry and to be able to map diverse parts of the application on different sets of cores. An interesting and challenging research topic will be to evaluate the implications for applications developers, and to identify the software primitives needed to carefully tune the operating point of the underlying hardware, in order to trade performance for energy savings when possible and to request high computing capabilities when required.

The work we presented about energy efficiency in datacenter networks has been carried on with a particular software router implementation based on X86 architecture and the Click [93] packet processing framework. The next step would naturally be to extend our results to different hardware architectures and software stacks [74, 86, 116]. A careful performance evaluation of our power saving algo-

rithm with applications beyond those explored here, especially applications with strict real-time requirements, is needed in order to extend the applicability of our work to devices acting as complex network appliances, also known as middleboxes [25]. We are also interested in applying the same design principles to more general streaming workloads like video, music, voice and gaming.

Bibliography

[1] Amazon EC2 spot instances.
http://aws.amazon.com/ec2/spot-instances
(Cited on page 7.)

[2] Amazon Elastic MapReduce.
http://aws.amazon.com/elasticmapreduce
(Cited on page 4.)

[3] Amazon Web Services.
http://aws.amazon.com
(Cited on page 4.)

[4] Apache Hadoop.
http://hadoop.apache.org
(Cited on pages 4 and 6.)

[5] BlueCoat Proxy.
http://www.bluecoat.com
(Cited on page 35.)

[6] Cisco Application Extension Platform.
http://www.cisco.com/en/US/products/ps9701/index.html
(Cited on page 35.)

[7] Cisco Green Research Symposium, March 2008.
http://goo.gl/MOUNl
(Cited on page 33.)

[8] Cisco Research Call on Power.
http://goo.gl/np8wr
(Cited on page 33.)

[9] Climate Savers Computing.
http://www.climatesaverscomputing.org
(Cited on page 3.)

[10] eHarmony Finds a Compatible Match with SeaMicro.
http://www.seamicro.com/sites/default/files/SM_CP01_v1.3.pdf
(Cited on page 6.)

[11] Facebook Prineville Datacenter.
http://www.facebook.com/prinevilleDataCenter
(Cited on page 4.)

[12] Google App Engine.
http://code.google.com/appengine
(Cited on page 4.)

[13] Google energy-saving data centers.
http://www.google.com/about/datacenters
(Cited on pages 3 and 4.)

[14] Heroku: Cloud Application Platform.
http://www.heroku.com
(Cited on page 4.)

[15] http_load - multiprocessing http test client.
http://www.acme.com/software/http_load
(Cited on page 19.)

[16] IEEE 802.3az Standard - Media Access Control Parameters, Physical Layers and Management Parameters for Energy-Efficient Ethernet.
http://goo.gl/fHtBq
(Cited on pages 33 and 39.)

[17] Intel Previews Intel Xeon 'Nehalem-EX' Processor.
http://goo.gl/460la
(Cited on page 11.)

[18] Intel processors specs.
http://ark.intel.com
(Cited on pages 8 and 17.)

[19] Intel Sandy Bridge.
http://www.intel.com/SandyBridge
(Cited on page 16.)

[20] Intel Turbo Boost Technology - On Demand Processor Performance.
http://goo.gl/sjTGX
(Cited on page 14.)

[21] Lighttpd homepage.
http://www.lighttpd.net
(Cited on page 19.)

[22] Linode Cloud.
http://www.linode.com
(Cited on page 4.)

[23] "M57 Patents" Dataset.
https://domex.nps.edu/corp/scenarios/2009-m57/net
(Cited on page 57.)

[24] Marlowe: Intelligent Control For the Cloud.
http://research.microsoft.com/en-us/projects/marlowe
(Cited on page 7.)

[25] Middleboxes: Taxonomy and Issues.
http://tools.ietf.org/html/rfc3234
(Cited on page 73.)

[26] NLANR: Internet Measurement and Analysis.
http://moat.nlanr.net
(Cited on page 57.)

[27] Open Compute Project.
http://opencompute.org
(Cited on page 3.)

[28] Open Compute Server Technology.
http://opencompute.org/project_category/server-technology
(Cited on page 4.)

[29] Powering a Google Search.
http://goo.gl/PKbF
(Cited on page 2.)

[30] Rackspace Hosting.
http://www.rackspace.com
(Cited on page 4.)

[31] Receive-Side Scaling Enhancements in Windows Server 2008.
http://goo.gl/hsiU9
(Cited on pages 37 and 55.)

[32] Riverbed WAN optimization.
http://www.riverbed.com/us/solutions/wan_optimization
(Cited on page 35.)

[33] Routebricks: Enabling General Purpose Network Infrastructure.
http://routebricks.org
(Cited on pages 2 and 5.)

[34] SAMSUNG Develops Industry's First DDR4DRAM, Using 30nm Class Technology.
http://www.samsung.com/greenmemory
(Cited on page 39.)

[35] Seamicro, Mozilla case study.
http://goo.gl/YHSkL
(Cited on page 6.)

[36] Seamicro SM10000 Family of High Density, Low Power Servers.
http://www.seamicro.com/node/164
(Cited on pages 6 and 31.)

[37] SimPy Simulation Package.
http://simpy.sourceforge.net
(Cited on page 25.)

[38] SPECpower ssj2008 results.
http://www.spec.org/power_ssj2008/results
(Cited on pages 39 and 40.)

[39] Streaming SIMD Extensions.
http://en.wikipedia.org/wiki/Streaming_SIMD_Extensions
(Cited on page 17.)

[40] Tensilica Multicore Design.
http://www.tensilica.com/methodology/multicore-design.htm
(Cited on page 35.)

[41] The Green Grid.
http://www.thegreengrid.org
(Cited on page 3.)

[42] VMware Cloud Foundry.
http://cloudfoundry.com
(Cited on page 4.)

[43] Vyatta Router Firewall and VPN Products.
http://www.vyatta.com/products
(Cited on page 35.)

[44] Windows Azure.
http://www.windowsazure.com
(Cited on page 4.)

[45] Big.LITTLE Processing with ARM CortexTM-A15 and Cortex-A7. Improving Energy Efficiency in High-Performance Mobile Platforms. *ARM whitepaper*, 2011.
(Cited on page 8.)

[46] Snapdragon S4 Processors: System on Chip Solutions for a New Mobile Age. *Qualcomm whitepaper*, 2011.
(Cited on page 8.)

[47] Variable SMP A Multi-Core CPU Architecture for Low Power and High Performance. *NVidia whitepaper*, 2011.
(Cited on page 8.)

[48] Y. Agarwal, S. Hodges, R. Chandra, J. Scott, P. Bahl, and R. Gupta. Somniloquy: augmenting network interfaces to reduce pc energy usage. In *Proceedings of the 6th USENIX symposium on Networked systems design and implementation*, NSDI'09, pages 365–380, Berkeley, CA, USA, 2009. USENIX Association.
(Cited on page 8.)

[49] M. Al-Fares, A. Loukissas, and A. Vahdat. A scalable, commodity data center network architecture. *SIGCOMM Comput. Commun. Rev.*, 38:63–74, August 2008.
(Cited on page 5.)

[50] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan. Data center tcp (dctcp). In *Proceedings of the ACM SIGCOMM 2010 conference on SIGCOMM*, SIGCOMM '10, pages 63–74, New York, NY, USA, 2010. ACM.
(Cited on page 2.)

[51] A. Anand, A. Gupta, A. Akella, S. Seshan, and S. Shenker. Packet caches on routers: the implications of universal redundant traffic elimination. In *Proceedings of the ACM SIGCOMM 2008 conference on Data communication*, SIGCOMM '08, pages 219–230, New York, NY, USA, 2008. ACM.
(Cited on pages 38 and 57.)

[52] G. Ananthanarayanan and R. H. Katz. Greening the switch. In *Proceedings of the 2008 conference on Power aware computing and systems*, HotPower'08, pages 7–7, Berkeley, CA, USA, 2008. USENIX Association.
(Cited on page 69.)

[53] D. G. Andersen, J. Franklin, M. Kaminsky, A. Phanishayee, L. Tan, and V. Vasudevan. Fawn: a fast array of wimpy nodes. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, SOSP '09, pages 1–14, New York, NY, USA, 2009. ACM.
(Cited on pages 6 and 31.)

[54] M. Annavaram, E. Grochowski, and J. Shen. Mitigating amdahl's law through epi throttling. In *Proceedings of the 32nd annual international symposium on Computer Architecture*, ISCA '05, pages 298–309, Washington, DC, USA, 2005. IEEE Computer Society.
(Cited on page 32.)

[55] M. B. Anwer, M. Motiwala, M. b. Tariq, and N. Feamster. Switchblade: a platform for rapid deployment of network protocols on programmable hardware. In *Proceedings of the ACM SIGCOMM 2010 conference on SIGCOMM*, SIGCOMM '10, pages 183–194, New York, NY, USA, 2010. ACM.
(Cited on page 35.)

[56] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. Above the clouds: A Berkeley view of cloud computing. Technical report, Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley, 2009.
(Cited on page 4.)

[57] S. Balakrishnan, R. Rajwar, M. Upton, and K. Lai. The impact of performance asymmetry in emerging multicore architectures. In *Proceedings of the 32nd annual international symposium on Computer Architecture*, ISCA '05, pages 506–517, Washington, DC, USA, 2005. IEEE Computer Society.
(Cited on page 32.)

[58] P. Barford and M. Crovella. Generating representative web workloads for network and server performance evaluation. *SIGMETRICS Performance Evaluation Review*, 26:151–160, June 1998.
(Cited on page 20.)

[59] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*, SOSP '03, pages 164–177, New York, NY, USA, 2003. ACM.
(Cited on page 7.)

[60] L. A. Barroso and U. Hölzle. The case for energy-proportional computing. *Computer*, 40(12):33–37, 2007.
(Cited on pages 1, 15, 25, 29 and 30.)

[61] B. Beavers. The story behind the intel atom processor success. *Design Test of Computers, IEEE*, 26(2):8 –13, march-april 2009.
(Cited on pages 12, 16 and 30.)

[62] S. Bell, B. Edwards, J. Amann, R. Conlin, K. Joyce, V. Leung, J. MacKay, M. Reif, L. Bao, J. Brown, et al. Tile64-processor: A 64-core soc with mesh interconnect. In *Solid-State Circuits Conference, 2008. ISSCC 2008. Digest of Technical Papers. IEEE International*, pages 88–598. IEEE, 2008.
(Cited on page 6.)

[63] M. Berezecki, E. Frachtenberg, M. Paleczny, and K. Steele. Many-core key-value store. In *Green Computing Conference and Workshops (IGCC), 2011 International*, pages 1–8. IEEE, 2011.
(Cited on page 6.)

[64] R. Brown, E. Masanet, B. Nordman, B. Tschudi, A. Shehabi, J. Stanley, J. Koomey, D. Sartor, P. Chan, J. Loper, et al. Report to congress on server and data center energy efficiency. *Public law*, 109:431, 2007.
(Cited on pages 2 and 3.)

[65] J. Chabarek, J. Sommers, P. Barford, C. Estan, D. Tsiang, and S. Wright. Power awareness in network design and routing. In *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE*, pages 457 –465, april 2008.
(Cited on pages 33 and 69.)

[66] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Bigtable: A distributed storage system for structured data. *ACM Trans. Comput. Syst.*, 26:4:1–4:26, June 2008.
(Cited on page 7.)

[67] J. S. Chase, D. C. Anderson, P. N. Thakar, A. M. Vahdat, and R. P. Doyle. Managing energy and server resources in hosting centers. In *Proceedings of the eighteenth ACM symposium on Operating systems principles*, SOSP '01, pages 103–116, New York, NY, USA, 2001. ACM.
(Cited on pages 7 and 69.)

[68] B.-G. Chun, G. Iannaccone, G. Iannaccone, R. Katz, G. Lee, and L. Niccolini. An energy case for hybrid datacenters. *SIGOPS Operating Systems Review*, 44:76–80, March 2010.
(Cited on page 16.)

[69] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live migration of virtual machines. In *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation - Volume 2*, NSDI'05, pages 273–286, Berkeley, CA, USA, 2005. USENIX Association.
(Cited on page 7.)

[70] B. F. Cooper, R. Ramakrishnan, U. Srivastava, A. Silberstein, P. Bohannon, H.-A. Jacobsen, N. Puz, D. Weaver, and R. Yerneni. Pnuts: Yahoo!'s hosted data serving platform. *Proc. VLDB Endow.*, 1:1277–1288, August 2008.
(Cited on page 7.)

[71] P. Costa, T. Zahn, A. Rowstron, G. O'Shea, and S. Schubert. Why should we integrate services, servers, and networking in a data center? In *Proceedings of the 1st ACM workshop on Research on enterprise networking*, WREN '09, pages 111–118, New York, NY, USA, 2009. ACM.
(Cited on page 2.)

[72] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *ACM Communication*, 51:107–113, Jan. 2008.
(Cited on page 4.)

[73] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: amazon's highly available key-value store. In *Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*, SOSP '07, pages 205–220, New York, NY, USA, 2007. ACM.
(Cited on page 7.)

[74] M. Dobrescu, N. Egi, K. Argyraki, B.-G. Chun, K. Fall, G. Iannaccone, A. Knies, M. Manesh, and S. Ratnasamy. Routebricks: exploiting parallelism to scale software routers. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, SOSP '09, pages 15–28, New York, NY, USA, 2009. ACM.
(Cited on pages 2, 5, 35 and 72.)

[75] W. Fisher, M. Suchara, and J. Rexford. Greening backbone networks: reducing energy consumption by shutting off cables in bundled links. In *Proceedings of the first ACM SIGCOMM workshop on Green networking*, Green Networking '10, pages 29–34, New York, NY, USA, 2010. ACM.
(Cited on page 33.)

[76] B. Fitzpatrick. Distributed caching with memcached. *Linux J.*, 2004:5–, Aug. 2004.
(Cited on pages 6 and 7.)

[77] E. Frachtenberg, A. Heydari, H. Li, A. Michael, J. Na, A. Nisbet, and P. Sarti. High-efficiency server design. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '11, pages 27:1–27:27, New York, NY, USA, 2011. ACM.
(Cited on page 4.)

[78] G. Gerosa et al. A Sub-2W Low Power IA Processor for Mobile Internet Devices in 45nm High-k Metal Gate CMOS. *IEEE Journal of Solid-State Circuits*, Jan. 2009.
(Cited on page 12.)

[79] S. Ghemawat, H. Gobioff, and S.-T. Leung. The google file system. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*, SOSP '03, pages 29–43, New York, NY, USA, 2003. ACM.
(Cited on page 7.)

[80] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta. Vl2: a scalable and flexible data center network. *Commun. ACM*, 54:95–104, Mar. 2011.
(Cited on pages 2 and 5.)

[81] B. Guenter, N. Jain, and C. Williams. Managing cost, performance, and reliability tradeoffs for energy-aware server provisioning. In *INFOCOM, 2011 Proceedings IEEE*, pages 1332–1340. IEEE, 2011.
(Cited on pages 7 and 31.)

[82] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu. Bcube: a high performance, server-centric network architecture for modular data centers. In *Proceedings of the ACM SIGCOMM 2009 conference on Data communication*, SIGCOMM '09, pages 63–74, New York, NY, USA, 2009. ACM.
(Cited on pages 2 and 5.)

[83] C. Guo, H. Wu, K. Tan, L. Shi, Y. Zhang, and S. Lu. Dcell: a scalable and fault-tolerant network structure for data centers. In *Proceedings of the ACM SIGCOMM 2008 conference on Data communication*, SIGCOMM '08, pages 75–86, New York, NY, USA, 2008. ACM.
(Cited on pages 2 and 5.)

[84] M. Gupta and S. Singh. Greening of the internet. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '03, pages 19–26, New York, NY, USA, 2003. ACM.
(Cited on pages 33 and 69.)

[85] V. Gupta and R. Nathuji. Analyzing performance asymmetric multicore processors for latency sensitive datacenter applications. In *Proceedings of the 2010 international conference on Power aware computing and systems*, HotPower'10, pages 1–8, Berkeley, CA, USA, 2010. USENIX Association.
(Cited on page 32.)

[86] S. Han, K. Jang, K. Park, and S. Moon. Packetshader: a gpu-accelerated software router. In *Proceedings of the ACM SIGCOMM 2010 conference on SIGCOMM*, SIGCOMM '10, pages 195–206, New York, NY, USA, 2010. ACM.
(Cited on pages 35 and 72.)

[87] T. Heath, B. Diniz, E. V. Carrera, W. Meira, Jr., and R. Bianchini. Energy conservation in heterogeneous server clusters. In *Proceedings of the tenth ACM SIGPLAN symposium on Principles and practice of parallel programming*, PPoPP '05, pages 186–195, New York, NY, USA, 2005. ACM.
(Cited on page 69.)

[88] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, and N. McKeown. Elastictree: saving energy in data center networks. In *Proceedings of the 7th USENIX conference on Networked systems design and implementation*, NSDI'10, pages 17–17, Berkeley, CA, USA, 2010. USENIX Association.
(Cited on page 69.)

[89] U. Hölzle. Brawny cores still beat wimpy cores, most of the time. *IEEE Micro*, 30(4), 2010.
(Cited on page 6.)

[90] U. Hölzle and L. A. Barroso. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. Morgan and Claypool Publishers, 1st edition, 2009.
(Cited on pages 1 and 4.)

[91] V. Janapa Reddi, B. C. Lee, T. Chilimbi, and K. Vaid. Web search using mobile cores: quantifying and mitigating the price of efficiency. In *Proceedings of the 37th annual international symposium on Computer architecture*, ISCA '10, pages 314–325, New York, NY, USA, 2010. ACM.
(Cited on pages 6 and 70.)

[92] R. Katz. Tech titans building boom. *Spectrum, IEEE*, 46(2):40–54, 2009.
(Cited on page 1.)

[93] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. The click modular router. *ACM Trans. Comput. Syst.*, 18:263–297, August 2000.
(Cited on pages 10 and 72.)

[94] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. The click modular router. *ACM Trans. Comput. Syst.*, 18:263–297, August 2000.
(Cited on pages 34 and 37.)

[95] J. Koomey. Growth in Data center electricity use 2005 to 2010. *Oakland, CA: Analytics Press*, 2011.
(Cited on pages 2 and 4.)

[96] A. Krioukov, P. Mohan, S. Alspaugh, L. Keys, D. Culler, and R. Katz. Napsac: design and implementation of a power-proportional web cluster. *SIGCOMM Comput. Commun. Rev.*, 41:102–108.
(Cited on pages 7 and 31.)

[97] R. Kumar, K. Farkas, N. Jouppi, P. Ranganathan, and D. Tullsen. Processor power reduction via single-isa heterogeneous multi-core architectures. *Computer Architecture Letters*, 2(1):2, january-december 2003.
(Cited on page 32.)

[98] R. Kumar, K. Farkas, N. Jouppi, P. Ranganathan, and D. Tullsen. Single-isa heterogeneous multi-core architectures: the potential for processor power reduction. In *Microarchitecture, 2003. MICRO-36. Proceedings. 36th Annual IEEE/ACM International Symposium on*, pages 81 – 92, dec. 2003.
(Cited on page 32.)

[99] W. Lang, J. M. Patel, and S. Shankar. Wimpy node clusters: what about non-wimpy workloads? In *Proceedings of the Sixth International Workshop on Data Management on New Hardware*, DaMoN '10, pages 47–55, New York, NY, USA, 2010. ACM.
(Cited on page 6.)

[100] T. Li, P. Brett, R. Knauerhase, D. Koufaty, D. Reddy, and S. Hahn. Operating system support for overlapping-isa heterogeneous multi-core architectures. In *High Performance Computer Architecture (HPCA), 2010 IEEE 16th International Symposium on*, pages 1 –12, jan. 2010.
(Cited on pages 17 and 32.)

[101] K. Lim, P. Ranganathan, J. Chang, C. Patel, T. Mudge, and S. Reinhardt. Understanding and designing new server architectures for emerging warehouse-computing environments. In *Proceedings of the 35th Annual International Symposium on Computer Architecture*, ISCA '08, pages 315–326, Washington, DC, USA, 2008. IEEE Computer Society.
(Cited on page 70.)

[102] G. Lu, C. Guo, Y. Li, Z. Zhou, T. Yuan, H. Wu, Y. Xiong, R. Gao, and Y. Zhang. Serverswitch: a programmable and high performance platform for data center networks. In *Proceedings of the 8th USENIX conference on Networked systems design and implementation*, NSDI'11, pages 2–2, Berkeley, CA, USA, 2011. USENIX Association.
(Cited on page 35.)

[103] P. Mahadevan, P. Sharma, S. Banerjee, and P. Ranganathan. A power benchmarking framework for network devices. In *Proceedings of the 8th International IFIP-TC 6 Networking Conference*, NETWORKING '09, pages 795–808, Berlin, Heidelberg, 2009. Springer-Verlag.
(Cited on page 33.)

[104] G. Maier, A. Feldmann, V. Paxson, and M. Allman. On dominant characteristics of residential broadband internet traffic. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, IMC '09, pages 90–102, New York, NY, USA, 2009. ACM.
(Cited on page 33.)

[105] M. R. Marty and M. D. Hill. Virtual hierarchies to support server consolidation. In *Proceedings of the 34th annual international symposium on Computer architecture*, ISCA '07, pages 46–56, New York, NY, USA, 2007. ACM.
(Cited on page 7.)

[106] D. Meisner, B. T. Gold, and T. F. Wenisch. Powernap: eliminating server idle power. *SIGPLAN Not.*, 44:205–216, March 2009.
(Cited on pages 8 and 70.)

[107] D. Meisner, C. M. Sadler, L. A. Barroso, W.-D. Weber, and T. F. Wenisch. Power management of online data-intensive services. *SIGARCH Comput. Archit. News*, 39:319–330, June 2011.
(Cited on pages 4, 8, 16, 25 and 70.)

[108] J. C. Mogul and K. K. Ramakrishnan. Eliminating receive livelock in an interrupt-driven kernel. *ACM Trans. Comput. Syst.*, 15:217–252, August 1997.
(Cited on page 40.)

[109] S. Nedevschi, L. Popa, G. Iannaccone, S. Ratnasamy, and D. Wetherall. Reducing network energy consumption via sleeping and rate-adaptation. In *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, NSDI'08, pages 323–336, Berkeley, CA, USA, 2008. USENIX Association.
(Cited on pages 33 and 69.)

[110] M. Olson, K. Christensen, S. Lee, and J. Yun. Hybrid Web Server: Traffic Analysis and Prototype. In *Conference on Local Computer Networks*. IEEE, 2011.
(Cited on page 7.)

[111] L. Popa, S. Ratnasamy, G. Iannaccone, A. Krishnamurthy, and I. Stoica. A cost comparison of datacenter network architectures. In *Proceedings of the 6th International COnference*, Co-NEXT '10, pages 16:1–16:12, New York, NY, USA, 2010. ACM.
(Cited on page 2.)

[112] B. Raghavan and J. Ma. The Energy and Emergy of the Internet. ACM, 2011.
(Cited on page 2.)

[113] P. Ranganathan, P. Leech, D. Irwin, and J. Chase. Ensemble-level power management for dense blade servers. In *Proceedings of the 33rd annual international symposium on Computer Architecture*, ISCA '06, pages 66–77, Washington, DC, USA, 2006. IEEE Computer Society.
(Cited on page 7.)

[114] D. Reddy, D. Koufaty, P. Brett, and S. Hahn. Bridging functional heterogeneity in multicore architectures. *SIGOPS Operating System Review*, 45:21–33, February 2011.
(Cited on pages 17 and 32.)

[115] P. Reviriego, K. Christensen, A. Sánchez-Macián, and J. A. Maestro. Using coordinated transmission with energy efficient ethernet. In *Proceedings of the 10th international IFIP TC 6 conference on Networking - Volume Part I*, NETWORKING'11, pages 160–171, Berlin, Heidelberg, 2011. Springer-Verlag.
(Cited on page 69.)

[116] L. Rizzo and M. Landi. netmap: memory mapped access to network devices. In *Proceedings of the ACM SIGCOMM 2011 conference on SIGCOMM*, SIGCOMM '11, pages 422–423, New York, NY, USA, 2011. ACM.
(Cited on page 72.)

[117] S. Rusu, S. Tam, H. Muljono, D. Ayers, J. Chang, B. Cherkauer, J. Stinson, J. Benoit, R. Varada, J. Leung, R. Limaye, and S. Vora. A 65-nm dual-core multithreaded xeon reg; processor with 16-mb l3 cache. *Solid-State Circuits, IEEE Journal of*, 42(1):17 –25, jan. 2007.
(Cited on page 11.)

[118] J. H. Salim, R. Olsson, and A. Kuznetsov. Beyond softnet. In *Proceedings of the 5th annual Linux Showcase & Conference - Volume 5*, pages 18–18, Berkeley, CA, USA, 2001. USENIX Association.
(Cited on page 41.)

[119] V. Sekar, S. Ratnasamy, M. K. Reiter, N. Egi, and G. Shi. The middlebox manifesto: enabling innovation in middlebox deployment. In *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*, HotNets '11, pages 21:1–21:6, New York, NY, USA, 2011. ACM.
(Cited on page 35.)

[120] N. Tolia, Z. Wang, M. Marwah, C. Bash, P. Ranganathan, and X. Zhu. Delivering energy proportionality with non energy-proportional systems: optimizing the ensemble. In *Proceedings of the 2008 conference on Power aware computing and systems*, HotPower'08, pages 2–2, Berkeley, CA, USA, 2008. USENIX Association.
(Cited on pages 7 and 69.)

[121] D. Tsirogiannis, S. Harizopoulos, and M. A. Shah. Analyzing the energy efficiency of a database server. In *Proceedings of the 2010 international conference on Management of data*, pages 231–242. ACM, 2010.
(Cited on pages 4 and 70.)

[122] G. Varghese, J. Sanjeev, T. Chao, K. Smits, D. Satish, S. Siers, V. Naydenov, T. Khondker, S. Sarkar, and P. Singh. Penryn: 45-nm next generation Intel core 2 processor. In *Solid-State Circuits Conference, 2007. ASSCC '07. IEEE Asian*, pages 14 –17, nov. 2007.
(Cited on page 11.)

[123] V. Vasudevan, L. Tan, D. Andersen, M. Kaminsky, M. A. Kozuch, and P. Pillai. FAWNSort: Energy-efficient Sorting of 10GB.
(Cited on page 6.)

[124] Q. Wang, R. Kassa, W. Shen, N. Ijih, B. Chitlur, M. Konow, D. Liu, A. Sheiman, and P. Gupta. An fpga based hybrid processor emulation platform. In *Proceedings of the 2010 International Conference on Field Programmable Logic and Applications*, FPL '10, pages 25–30, Washington, DC, USA, 2010. IEEE Computer Society.
(Cited on page 32.)

[125] D. H. Woo and H.-H. S. Lee. Extending amdahl's law for energy-efficient computing in the many-core era. *Computer*, 41:24–31, December 2008.
(Cited on page 32.)