January 2013

# Packet Coalescing and Server Substitution for Energy-Proportional Operation of Network Links and Data Servers

Mehrgan Mostowfi
*University of South Florida*, mostowfi@gmail.com

# Packet Coalescing and Server Substitution for Energy-Proportional Operation of Network Links and Data Servers

by

Mehrgan Mostowfi

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
Department of Computer Science and Engineering
College of Engineering
University of South Florida

Major Professor: Ken Christensen, Ph.D.
Nataša Jonoska, Ph.D.
Srinivas Katkoori, Ph.D.
Miguel Labrador, Ph.D.
Daniel Yeh, Ph.D.

Date of Approval:
May 17, 2013

Keywords: Green Networks, Energy Efficient Ethernet, Hybrid Server,
Performance Evaluation, Modeling, Simulation, Prototype

## Dedication

To my mother, Parvaneh

## Acknowledgments

I would like to sincerely appreciate the great encouragement, support and guidance I received from my advisor Dr. Christensen throughout this research. I would also like to thank Dr. Jonoska, Dr. Katkoori, Dr. Labrador, and Dr. Yeh for agreeing to serve on my dissertation committee and for their constructive comments.

Thanks are due to those who financially supported this research. Funding for this work was provided in parts by the Department of Computer Science and Engineering at USF, Korea Electronics Technology Institute (KETI), and Cisco Systems, Inc., which is greatly appreciated.

My deepest gratitude goes to my mother, Parvaneh, and my sister, Mitra, for their lifelong love and support. This dissertation would not have been completed without their constant encouragement and kindness.

Lastly, I offer my regards to all of those who supported me in any respect during the completion of this work.

## Table of Contents

## List of Tables

## List of Figures

## Abstract

Electricity generation for Information and Communications Technology (ICT) contributes over 2% of the human-generated $CO_2$ to the atmosphere. Energy costs are rapidly becoming the major operational expense for ICT and may soon dwarf capital expenses as software and hardware continue to drop in price. In this dissertation, three new approaches to achieving energy-proportional operation of network links and data servers are explored.

Ethernet is the dominant wireline communications technology for Internet connectivity. IEEE 802.3az Energy Efficient Ethernet (EEE) describes a Low Power Idle (LPI) mechanism for allowing Ethernet links to sleep. A method of coalescing packets to consolidate link idle periods is investigated. It is shown that packet coalescing can result in almost fully energy-proportional behavior of an Ethernet link. Simulation is done at both the queuing and protocol levels for a range of traffic models and system configurations. Analytical modeling is used to gain a deeper general insight into packet coalescing.

The architecture of a hybrid web server based on two platforms – a low-power (ARM based) and a high-power (Pentium based) – can be used to achieve step-wise energy-proportional operation and maintain headroom for peak loads. A new method based on Gratuitous ARP for switching between two mirrored platforms is developed, prototyped, and evaluated. Experimental results show that for up to 50 requests per minute, a hybrid server where the Master platform is a 2012 server-grade desktop PC can sleep for 50% of time with no increase in response time.

HTTP can be used for redirection in space – a new method for precise redirection in time is proposed and used to schedule requests to a high-power server in a hybrid server. The scheduling method is modeled as a single server queue with vacations where the vacation duration is fixed and the service distribution is directly a function of the request load. This approach is well suited for delay tolerant applications such as application updates and file back-up. Energy-proportional operation is shown to be achievable in a prototype system.

A first-order estimation with conservative assumptions on the adoption rate of the methods proposed and studied here shows that these methods can collectively enable energy savings in the order of hundreds of million dollars in the US annually.

# Chapter 1: Introduction

Due to the rapid conversion of information format to digital, and information and service delivery to Internet-based, there has been a phenomenal growth in the energy consumption of ICT in the last decade. The cost to power ICT systems is rapidly becoming the highest operational cost of datacenters and may soon surpass capital expense as hardware and software costs continue to drop. Burning fossil fuels to generate this electricity not only releases several pollutants to the environment, but also contributes to $CO_2$ emissions to the atmosphere. From a scientific point of view, $CO_2$ has properties that suggest it can contribute significantly to the warming of the planet [59]. This dissertation is focused on reducing the energy use of ICT systems connected to the Internet via wired networks. More specifically, this dissertation focuses on three large energy consumers of the Internet: Ethernet links, Ethernet switches, and data servers.

## 1.1 Background

To understand the scope and significance of this work, it is necessary to have a quantitative understanding of 1) how much energy ICT systems use, 2) how much of this energy is wasted, and 3) how much of the wasted energy can be saved through reducing the energy consumption of ICT systems during operation. The definitions used throughout this dissertation for a system and a component are taken from [12]. A system is a collection of components, which are typically heterogeneous in nature. A component is an atomic unit within a system. Note that these definitions are abstract and general, and their granularity can vary based on use. For instance, an Ethernet LAN switch can be considered as a system. The components within the switch will then be the CMOS switch chip, the port blocks, optional memories and CPUs within the switch, etc. A port block can also be considered as an entire system with components such as the PHY and MAC.

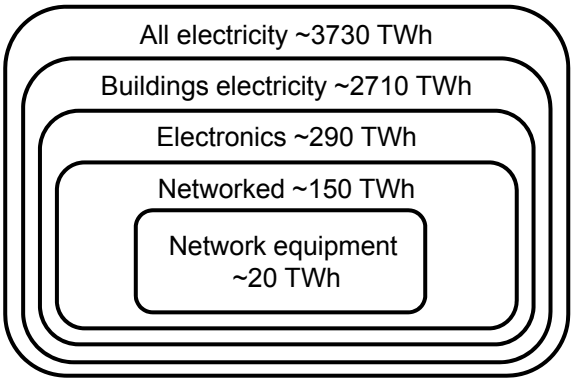



Figure 1.1: US electricity consumption in 2008 [29], [86] (not to scale)

### 1.1.1 Energy Consumption of ICT

ICT systems (including PCs, servers, cooling, fixed and mobile telephony, LANs, office telecommunications and printers, as defined in [34]) are typically connected to the Internet. The number of ICT systems has been growing rapidly, as evident by the statistics of the hosts connected to the Internet (zero to 908 million from 1980 to 2012 [54]). Although the individual consumption of these systems is decreasing, the electricity consumption of ICT as a whole is increasing due to the growing number of these systems. It has been estimated that worldwide energy use from data centers doubled between 2000 and 2005, and increased by 56% from 2005 to 2010 [63]. This corresponds to about 1.3% of all electricity use in the world and for the US about 2% of all electricity use [63]. PCs and monitors consume roughly 100 TWh/year in the US, which costs about $10 billion at the average national electricity rate of $0.10 per kWh [88]. Figure 1.1 shows the consumption of ICT in comparison to the electricity consumption of the country and in buildings in 2008. As can be seen in this figure, 75% of all electricity in the US is consumed in buildings. Almost 10% of all electricity consumed in buildings (8% of the total consumption) is due to electronics and almost half of this consumption is attributed to networked electronic systems (connected to the Internet more than any other network). ICT also contributes over 2% of the human-generated $CO_2$ to the atmosphere which is about the same as that of the aviation industry [34]. The trend is that 1) all consumer electronic systems will likely be connected to the Internet (as can currently be seen in TV sets, game consoles, etc.), and 2) the consumption of consumer electronic systems may double by 2030 [30].

### 1.1.2 Energy Proportionality as a Key Concept

It has been shown that ICT systems and components within them (such as Ethernet links), are often very lightly utilized. For instance, studies described in [90] and [92] showed that common edge links (links from edge systems like PCs to switches) operate at much lower than their full capacity most of the time. Even servers in Google datacenters are utilized between 10% to 50% most of the time [10]. ICT systems are intentionally configured for low average utilization to allow for headroom (or slack) for infrequent peak load conditions. However, these systems consume about the same power while idle as when heavily loaded. Note that power and energy are sometimes used interchangeably in the context of Green Computing, which is not correct. Energy is the work done in a system to generate the desired outcome (transmission of bits, calculation, etc.), and is measured in joules (J) in the SI system. One joule is the energy expended in one second by one ampere current against a resistance of one ohm. Power is the rate at which energy is consumed (or produced) and is measured in watts (W = J/s) in the SI system. The unit used for energy is often kilowatt hours (kWh), since it is the common billing unit for energy delivered to consumers by electric energy providers (power companies). While special care has been taken in this dissertation not to confuse the two concepts, the verb "consume" is used for both power and energy, as is common in the literature. The term "power draw" may be the correct term for power.

The Energy Proportionality Index of an ICT system at different utilization levels is defined as,

$$E_P = \frac{U}{P}, \tag{1.1}$$

where $U$, $0 \leq U \leq 1$, is the utilization as a ratio (or percentage) of the full capacity, and $P$, $0 < P \leq 1$, is the power consumption as a ratio (or percentage) of the peak power consumption. For $P = 0$ (at $U = 0$), it is defined $E_P = 1$. A system is energy-proportional when its $E_P$ for every utilization, $U \geq 0$, is equal to 1. In other words, an energy-proportional system consumes no power when not being utilized and its power consumption grows in proportion to its utilization ([10] and [109]). Here, utilization is solely determined by the traffic load on the system and not by any other factor such as computations needed to determine the route on a switch to which a packet will be forwarded or decomposition and analyzing packet headers. Therefore, if for example the aggregate packet load on a 10 Gb/s Ethernet switch is 2 Gb/s, the utilization of the switch is 20%.

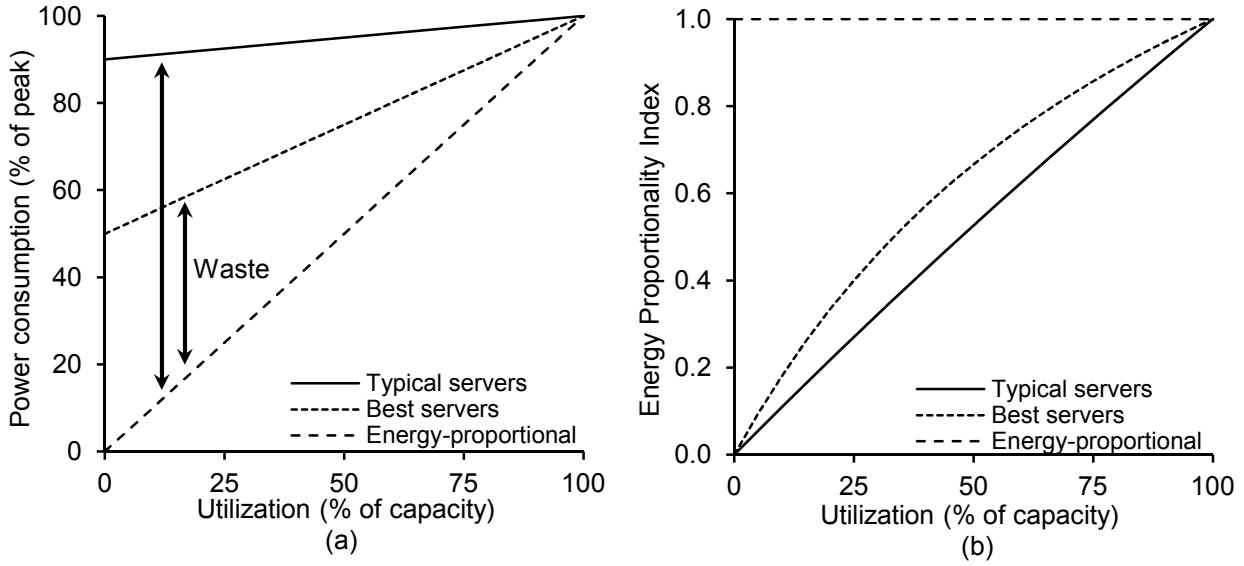



Figure 1.2: Servers (a) power consumption, and (b) energy proportionality index

Figure 1.2a shows the power consumption of typical servers. The peak and base power in most servers are almost the same – power consumption in 0% utilization (idle) is 90% of the peak power. For instance, a Dell OptiPlex 790 server-grade PC that is used as a part of the testbeds in this dissertation has shown to have about 50 W power consumption when idle, and 55 W when ten simultaneous file download requests for a 100 MB static file are constantly made to the server (heavy utilization). In Figure 1.2b, the $E_P$ of conventional, best, and energy-proportional servers is depicted. As shown in this figure, the $E_P$ of an energy-proportional system is always 1.

It is important to note that $E_P$ is different than efficiency, as efficiency is the useful output of a server divided by the total input, and is always less than 1. Energy Proportionality Index, $E_P$, however, can be greater than 1. For instance, consider server 1 and server 2 coupled to collectively make a hybrid server (hybrid servers will be defined and studied in detail in Chapter 4). Server 1 is a high-power high-performance server capable of serving loads up to 100% of its capacity with acceptable quality of service. Server 2 is a low-power low-performance server with the same serving capability of server 1. However, server 2 can only serve loads up to 40% of server 1's full capacity with acceptable quality of service. Beyond 40% load, the quality of service of server 2 would not be acceptable. The server switches the serving server to server 1 when the overall load on the hybrid server exceeds 40%, and to server 2 when the load on the server falls below 40%. The full power consumption of the server as a whole is server 1's peak power and the capacity of the server is 100% of server 1's capacity. Figure 1.3 depicts this hybrid server's power consumption as a

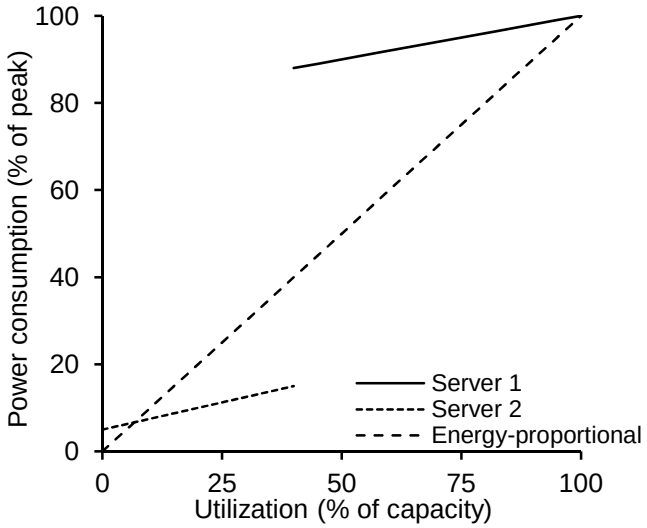



Figure 1.3: Power consumption of a hybrid server

function of load. As can be seen in this figure, if the overall load on the server stays below 40%, the server can deliver an $E_P$ of more than 1. A hybrid server with an operation similar to this example will be studied in Chapter 4.

There are major efforts underway to reduce the energy consumption of ICT systems. The methods that are closely related to this dissertation are reviewed in detail in the next chapter. Reducing the energy consumption of ICT systems requires a focus on the following areas:

1. Reducing peak power use such that systems do not hit a "power wall" beyond which unconventional and high-cost cooling methods such as water cooling is required for the systems to operate.

2. Increasing system utilization to operate in a more favorable region where $E_P$ is greater.

3. Making systems more energy-proportional by decreasing the idle power consumption and making the consumption more proportional to utilization.

The first area is largely addressed by a focus on low-power VLSI design. The second area is addressed more by approaches in the areas of load balancing, server virtualization and energy-aware routing. The third area is the focus of this dissertation.

In this dissertation, a power consumption proportional to utilization (or offered load, if there is no loss) is considered as the ideal power consumption for the systems under study. The methods studied in this work try to reduce the difference between the actual and this ideal power consumption (eliminate the energy

waste), thus bringing the system closer to energy-proportional (the lines labeled as "energy-proportional" in Figure 1.2).

### 1.1.3 Opportunity for Energy Savings in Idle Periods

ICT systems, and components within them, often have the capability to be placed into a low-power sleep state to conserve energy. In the sleep state, these systems consume a fraction of the power they would normally use when on. Two examples are Energy Efficient Ethernet [51] and Hybrid Servers [25], which will be the focus of the majority of this dissertation. Common between the two is that both utilize the sleep capability in idle periods to gain energy savings and approach energy-proportionality. A brief explanation of each follows. A thorough explanation and details of each will be presented later in Chapters 3 and 4. IEEE 802.3az Energy Efficient Ethernet (EEE) [51] is an standard for twisted pair Ethernet to enable the Ethernet link to enter a Low-Power Idle (LPI) state when no traffic is on the link to transmit. By entering the LPI state, the physical layer is put to sleep which reduces the overall power consumed by the link. A hybrid server consists of two co-located heterogeneous server platforms where one platform is low-performance and low-power and the other is high-performance and high-power. The low-power platform can perform some (or all) of the tasks of the high-power platform, although with lower performance. When the high performance of the high-power platform is not needed (in idle periods, for instance) it is put to sleep and the service is switched to the low-power platform.

Given the low utilization, there are often periods of idleness where the ICT system is not needed. If the system is put to sleep during these periods, the power consumption will drop to that of the sleep state and energy will be saved (Figure 1.4). However, there is always a transition time associated with sleeping and waking up. The transition times are $T_s$ (wake-to-sleep) and $T_w$ (sleep-to-wake). An idle period is shown by $t_{gap}$, the time the system spends in the sleep state is $t_{sleep}$, and,

$$t_{sleep} = t_{gap} - (T_s + T_w). \tag{1.2}$$

However, not all idle periods are suitable for sleeping because of the transition overhead. Of interest is the conditions that should be met for sleeping in idle period to result in energy savings. This analysis, presented below, is similar to what is presented in [50].

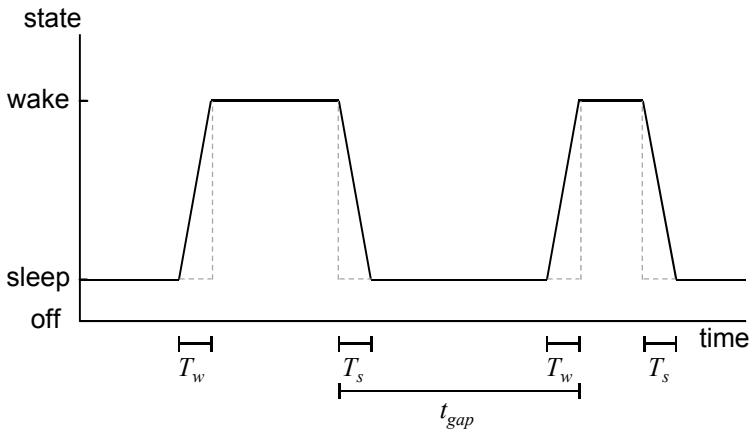



Figure 1.4: States of a system with the capability of sleeping in idle periods

Let the power consumption of the system in the wake state and the sleep state (as a percentage or ratio of the system's peak power) be $P_a$ and $P_s$, respectively. Similarly, let the power consumption of the system while in wake-to-sleep transition and while in sleep-to-wake transition be $P_{ws}$ and $P_{sw}$, respectively. Also, let $G$ denote the energy savings gained by sleeping in an idle period. Clearly,

$$G = t_{gap}P_a - \left(T_sP_{ws} + T_wP_{sw} + P_st_{sleep}\right), \tag{1.3}$$

where $t_{sleep}$ can be obtained using Equation (1.2). For instance, in a system where $P_a = 1$, $P_s = 0.1$, $P_{ws} = 0.7$, $P_{sw} = 1.4$, $T_s = 2$ s, and $T_w = 4$ s, $G = 0.9t_{gap} - 7.0$. So, in order for $G > 0$, $t_{gap}$ should be greater than 6.3 s. Sleeping in a shorter idle period will result in a zero, or negative energy gain. Formally,

$$G > 0 \tag{1.4}$$

$$t_{gap}P_a - \left(T_sP_{ws} + T_wP_{sw} + P_st_{sleep}\right) > 0 \tag{1.5}$$

$$t_{gap}P_a - T_sP_{ws} - T_wP_{sw} - P_s\left(t_{gap} - T_s - T_w\right) > 0 \tag{1.6}$$

$$t_{gap}\left(P_a - P_s\right) > T_s(P_{ws} - P_s) + T_w(P_{sw} - P_s). \tag{1.7}$$

So, the necessary condition that must be met in order for sleeping in an idle period of length $t_{gap}$ to achieve energy savings is,

$$t_{gap} > \frac{T_s(P_{ws} - P_s) + T_w(P_{sw} - P_s)}{(P_a - P_s)}. \tag{1.8}$$

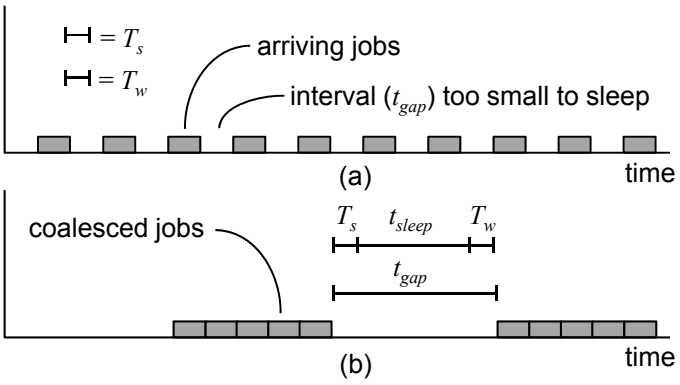



Figure 1.5: Arriving jobs (a) without, and (b) with coalescing

Typically, $P_a \geq P_{sw} \geq P_{ws} \geq P_s$, although in systems where there are power spikes when the system wakes up from sleep, $P_{sw} > P_a$. In the systems studied in this dissertation, it is assumed that $P_a = P_{sw} = P_{ws}$. This assumption is either based on direct measurements of the systems under study, or on the values advised by the manufacturers of the systems under study, which will be explained in their corresponding chapters. With this assumption, Equation (1.8) can be simplified as

$$t_{gap} > T_s + T_w. \tag{1.9}$$

Energy saving by transitioning to sleep cannot be achieved in idle periods that are not suitable for sleeping. On the other hand, if a few idle periods are consolidated into a longer, continuous idle period, the system can use it for sleep and saving energy. Figure 1.5a shows the notion of arriving jobs with interarrival times (that is, idle periods between individual jobs) too short for sleeping, but when the jobs are coalesced (Figure 1.5b) the now fewer idle periods are of extended and sufficient duration for sleeping. In Figure 1.5a, the time between job arrivals is less than $T_s + T_w$ – Equation (1.9) – and thus the idle period is not suitable for sleeping.

## 1.2 Motivation

This work is motivated by the following factor: In two equal systems with equal sum of idle periods, the system that has more suitable-for-sleeping time in idle periods has a greater opportunity to save energy. Coalescing of workload (or jobs) before being served can be used to consolidate idle periods, thus creating

extended idle periods to exploit for sleep. The challenge is to understand, predict, and being able to control the energy-delay trade-off caused by coalescing and the effect of the delay in network traffic including the downstream effects.

## 1.3 Thesis Statement and Contributions

The thesis is that by scheduled coalescing of jobs, the energy consumption of network links and data servers can be reduced with a controlled trade-off of energy use and response time. The key contributions of this work are the following:

1. A new taxonomy is developed that concisely categorizes system-level power management methods that focus on reducing the energy consumption of wired networks. This taxonomy enables a deeper insight into power management methods, both past and future. This work is not yet published.

2. A method that uses packet coalescing to consolidate idle periods in Ethernet links that support the IEEE 802.3az EEE standard is studied and evaluated. This method achieves near energy-proportional operation in an Ethernet link. This work is published in [24], [75], and [76].

3. A new design for a hybrid data server is presented and evaluated. The new design allows seamless switching of serving platforms based on predicted future load and achieves near energy-proportional operation for the server. This work is published in [78].

4. An extension of the notion of HTTP redirection is proposed that allows redirection in time in addition of space. The extension enables distributed coalescing of web requests to achieve near energy-proportionality of hybrid servers for delay-tolerant applications. Two publications resulted from this work, [77] and [79]. The former is under review for publication and the latter is published.

Also note that some sections in Chapter 3 are copied verbatim from [74]. Repeating these material here is necessary for clarity. These sections are 3.2.1, 3.2.2, and 3.2.3, which are taken from Chapter 3 of [74].

## 1.4 Outline

The chapters are organized as follows:

- Chapter 2 presents a literature review of the past work done on system-level power management methods. A new taxonomy is developed in this chapter to categorize the past work.

- Chapter 3 presents an analytical model and a simulation model for packet coalescing for EEE, and also the performance evaluation of packet coalescing for EEE in Ethernet links. In this chapter, performance evaluation of a new EEE policy of synchronous coalescing of packets in network hosts and edge routers is also presented. This method is evaluated using an ns-2 simulation model of a LAN switch. It is shown that the method can significantly reduce the overall energy use of a LAN switch while introducing limited and controlled effects on typical Internet traffic.

- Chapter 4 presents the design, prototype implementation, and performance evaluation of a hybrid data server. In this chapter, it is shown how a new hybrid data server architecture based on two co-located, mirrored platforms (one high-performance and high-power, and the other low-performance and low-power) can be architected to appear as a single system image to clients, significantly reduce energy consumption, and maintain an acceptable response time.

- Chapter 5 presents a method of time shifting web requests using HTTP redirection to coalesce them before being served by a data server. The performance of the method is studied using a prototype hybrid server with time shifting capability.

- Chapter 6 concludes this work and suggests possible related future work in the field and introduces open problems.

## Chapter 2: Background and Literature Review

In the life cycle of an ICT system, there are three parts that contribute to its overall energy consumption. The first part is the energy consumed to manufacture the system (embodied energy or Emergy [95]). The second part is the energy consumed to operate the system, which is the subject of conventional networks energy-efficiency research (as well as the subject of this dissertation). For instance, [70] estimates the energy to produce and operate a rack switch for 10 years to be 780 kWh and 13000 kWh (assuming 150 W average power consumption), respectively. Other sources have also estimated the energy consumption and the environmental impact of the life cycle of PCs ([115] and [48], for instance), computer systems in enterprise environments ([45], and [101], for instance) and network equipment ([87]). There is also a third part in the energy consumption of ICT systems which is the energy consumed to end the life-cycle of the system. The energy consumption and the environmental impacts of the first part can be reduced by more energy-efficient and cleaner mining and manufacturing techniques, and using less materials with lower toxic substances. Similarly, the third part can be made greener by more efficient recycling practices or refurbishing and reusing the system. An essential method of reducing the operational energy consumption (the second part) which is the focus of this dissertation is to power manage a system. That is, for instance, to put the whole system or some components within it to sleep in response to lower utilization.

In this chapter, a history and background of power management methods at the system level is first presented. Then, power management methods in the context of networks are reviewed in a new taxonomy consisting of three broad areas of scheduling resources, substituting technologies, and consolidating resources.

## 2.1 Background and History of Dynamic Power Management

Prior to 1998, several methods of reducing the power consumption of individual ICT systems, or components within them, were studied and evaluated. Two sources, [68] and [12], were the first to formally define the term Dynamic Power Management (DPM) and categorize the approaches that had been studied until then. DPM is defined as "a design methodology that dynamically reconfigures an electronic system to provide the requested services and performance levels with a minimum number of active components or a minimum load on such components" [12]. DPM addresses how and when to turn off or reduce the rate at which a system works (and thus the power consumption) during periods where the system (or some components within it) is idle or partially needed. Determining when to reconfigure the system is the power management policy. For instance, the simplest policy in a system-level DPM method in a PC is to turn the display off after a certain period of inactivity, which can be defined as no user input.

Invariably, power managing a system results in some performance degradation. The concept of performance in a system is defined based on many factors such as delay in performing tasks (such as serving web requests), loss (such as dropping network packets), variation in response time, availability at times when the system is needed, and so on. Also, these factors can change based on the nature of the system, its applications, location, types of its users, etc. The trade-off between energy savings and the decrease in performance is one of the main issues that needs to be addressed when studying any power management method. Without this issue in consideration, one can (correctly) claim that the best power management method is to turn the entire system off all the time, thus consuming no power and achieving 100% energy savings.

Today's DPM methods seek to reduce the power consumption of a system while limiting the performance decrease to an acceptable level. There are two essential approaches to achieve this:

- Sleep and wakeup, where all or some components of a system are put to a low(er)-power state (usually called sleep state or mode).
- Rate adaptation, where the rate at which the system or component works is lowered when possible to reduce the power consumption.

Several DPM methods can be used simultaneously in a system at various levels and at different time scales. The time scale at which the above approaches can work depends heavily on the transition time to,

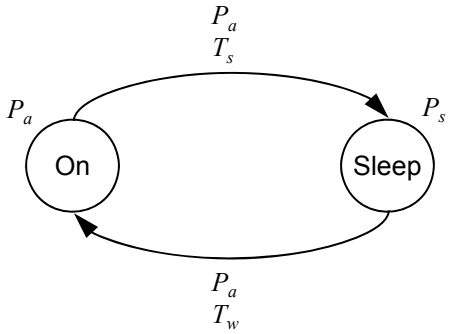



Figure 2.1: State transitions in sleep and wakeup approach

and from, sleep or the reduced-rate state. For instance, while an individual CPU can sleep in time scales of milliseconds, an entire PC needs an idle period of magnitudes longer (minutes) to be able to sleep due to its higher transition times (typically a few seconds). Figure 2.1 shows the states, their corresponding power consumptions ($P_a$ and $P_s$), state transitions, their transition times ($T_w$ for waking up and $T_s$ for going to sleep) for a system that has two states, on and sleep. As is typical in many systems, the power consumed during transitions can conservatively be assumed equal to the power consumption when on.

### 2.1.1 Sleep and Wakeup

As shown in Figure 2.1, a system can transition between active and sleep states, each of which taking a non-zero time. There are normally periods of inactivity (idle periods) for any system. Maximum sleep time with minimum performance impact will be achieved if the system is put to sleep at the beginning of an idle period and is woken up just as long before the end of the idle period to cover the transition time to active state. So, the system would be ready exactly when it needs to serve its workload and be sleeping otherwise. Sleeping in the optimal time periods requires information about the future and is not possible considering the random nature of ICT systems workload. This is the cause of performance trade-off. Since the end of an idle period (arrival of the next job) cannot be predicted precisely, two scenarios can happen; either the system is still sleeping and the job is lost due to the system's unavailability (unless another system covers or proxies during this period), or the job is buffered and served when the system becomes active again. This latter causes added delay on jobs. On the other hand, when the beginning of an idle period is not determined precisely, some power saving opportunities will be lost.

Predicting the beginning and duration of idle periods to exploit for energy savings was first addressed in 1996 [105], and was shortly followed by [50] in 1997. In [105], the authors use a regression model and a simple heuristic based on past history to predict the next idle period. The authors of [50] use an exponentially weighted moving average method to predict the next idle period. These works will be reviewed in more detail later in this chapter. Predicting idle periods has also been addressed using stochastic models such as Markov models [11], semi-Markov models [102], or adaptive learning trees [26]. Note that although prediction of idle intervals precisely is of great value, the computation overhead of sophisticated methods can sometimes increase the system's power consumption, or even take magnitudes longer than the timescales where idle periods occur. Therefore, simpler methods of prediction may be more practical and beneficial in certain cases.

### 2.1.2 Rate Adaptation

Rate Adaptation methods are the approaches that reduce the system's rate and bring it closer to its utilization in order to reduce the power consumption. This is done by reducing the clock frequency of integrated circuits in the system (CPUs, for instance). The power consumed by a CMOS integrated circuit ($P$) is

$$P = CV^2 f, \tag{2.1}$$

where $C$ is the capacitance, $V$ is the voltage, and $f$ is the frequency. Since $V$ is directly proportional to frequency, the relationship between power and frequency is

$$P \propto f^3. \tag{2.2}$$

Theoretically, if the frequency of a circuit is halved, the power consumption will decrease to $1/8$ of the original. However, the same task would then take twice the original time to complete. Therefore, the energy consumption for completing the same task would drop to $1/4$ of the original consumption. Changing frequency for energy efficiency in CMOS design was first proposed in 1992 [17] and was broadly utilized in different DPM approaches ever since.

## 2.2 Methods of Reducing the Energy Consumption of Networks

Energy consumption of the Internet was not a primary concern prior to the 21st century since the global deployment of the Internet was at its early stages and the total energy consumed by the Internet was negligible compared to the national energy consumption. Prior to 2003, the research on energy consumption of network systems was confined to architecture and VLSI levels for the components of the system in which approaches to lower the energy consumption of hardware components of interconnection networks such as the microprocessor and switching fabric were explored ([44] and [104], for example). However, none of them addressed the energy consumption of networks as a whole. In 1999, the opinion of the energy consumption of transferring data over the Internet being too high was expressed in an unrefereed publication by Huber and Mills [49]. They estimated the energy required to transfer 2 MB of data over the Internet to be equivalent to burning 1 pound of coal. They also claimed that the Internet and systems connected to it account for 13% of the energy consumption in the US. However, it was later revealed that their estimate was "at least eight times too high and their estimate of total power used by office equipment is overstated by at least a factor of four" [64]. The concern of the energy consumption of the Internet did not attract much attention until 2003 where Gupta and Singh used the absolute values of the energy consumed by various network systems to show that although the relative percentage of this consumption seemed low, the absolute energy consumption is too high to be neglected anymore [41]. They used the energy consumption of hubs, LAN and WAN switches and routers stated in [99] and calculated a total of 6.05 TWh for the total energy consumption of network systems in 2000. They also noted that this total consumption is almost 0.07% of total US energy expenditure of 2000 which seems to be negligible. They, however, brought forward three main reasons to argue that there still were good reasons to consider reducing this consumption. Their reasons were the energy inefficiency, the opportunity for vaster deployment of the Internet, and benefits in case of a natural disaster. To show the significance of the first reason, they compared the energy consumption per byte between the current wired Internet and the ideal consumption in wireless communication and concluded that wireless is between 1.25 and 10 times more efficient. The second reason is that lower energy consumption would allow the Internet to be deployed in energy-scarce parts of the world and the third reason was that in case of a power shortage, networking equipment could operate longer on their UPS batteries if they were to be made more energy-efficient.

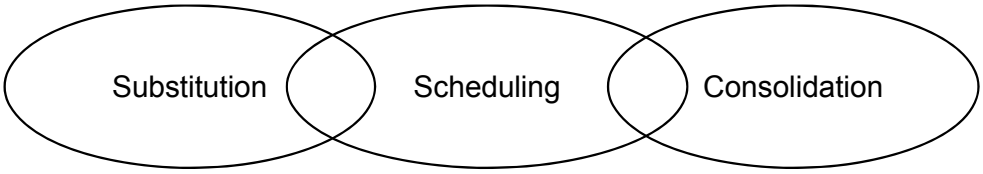



Figure 2.2: Broad categories for methods of reducing the energy consumption of networks

Since 2003, many methods of making networks more energy-efficient were proposed and studied. The most significant of these methods that are related to this dissertation will be reviewed in the following sections. In a broad categorization, methods of reducing the energy consumption of networks can be classified into the following categories (Figure 2.2):

1. Scheduling, where jobs are shifted in time, or their service time may change, to make (or extend) idle periods where the system can sleep, or work at a lower rate.

2. Substitution, where some (or the entire) services of a high-power system is transferred to a low-power system, and the high-power system is put to sleep.

3. Consolidation, where load from multiple systems are aggregated on fewer systems, enabling the excess systems to be put to sleep.

In fact, each of the above class of methods use idle periods in a different way to reduce the energy consumption. Scheduling utilizes idle periods in time since it manipulates the time of occurrence and the duration of idle periods, substitution and consolidation exploit idle periods in space since they both move the idle periods from one (or many) systems to another (or others). Scheduling can be a part of consolidation and substitution methods. For instance, a substitution method may manipulate idle periods by scheduling jobs to make (or extend the existing) idle periods that are suitable for switching the service to the lower-power system with minimal performance degradation. These types of methods will be studied in more detail later in this chapter.

### 2.2.1 Scheduling

As mentioned before, Gupta and Singh [41] were the first to address the energy consumption of the Internet as a whole. They also proposed the first general scheduling method for energy efficiency of wired

networks. They proposed a generic approach (called uncoordinated sleeping) "where an interface sleeps based on local decisions alone. This link layer approach could work as follows – when an interface is about to sleep, it informs its neighbor of the fact (on a point to point link this neighbor is well-defined, in a broadcast medium, all neighbors are informed via a broadcast). When the neighbor needs to send packets to the sleeping interface, it first sends a packet that forces the interface to wake up. Then, after waiting the necessary wakeup time, the actual packets are sent." [41]. As they mentioned, their scheduling method (as well as many other scheduling approaches studied later) was likely to be inspired by the earlier work that had been done in wireless and sensor networks. Not surprisingly, energy efficiency in wireless networks had been subject to extensive research for several reasons. Wireless interfaces are used in mobile systems and sensor networks which are both battery powered. Network interfaces consume a significant portion of the battery energy in wireless systems, so improving their energy consumption has a great impact on battery life [106]. Many solutions studied in the context of wireless networks can be applied to the Internet as well. Two comprehensive surveys of the research done in this field is presented in [5] and [56].

DPM methods in wireless networks have been studied in different network layers which include transport layer, network layer (energy efficient routing algorithms) and in MAC layer. Scheduling for energy efficiency in MAC layer explores methods that enable the wireless network interface to sleep while there is no data to transmit or receive while maintaining the system's presence in the network. Methods such as Sparse Topology and Energy Management (STEM) [100] and Power Aware Multi-Access protocol (PAMAS) [103] are among the most significant approaches to increase the time a wireless network interface spends in sleep mode in order to conserve energy. Shruger's et al. proposed STEM [100] which uses one radio for transmitting and receiving wakeup signals and another radio for packet transmission. The data transmission radio is in stand-by mode and only turns on when the wakeup radio receives a wakeup beacon from a neighbor node. The wakeup radio is also in sleep mode but is periodically turned on to listen to beacons from other nodes. Therefore, the neighbor that needs to initiate a connection sends a stream of beacons to ensure that at least one is received by the wakeup radio. Singh and Raghavendra studied PAMAS [103] which schedules the times of packet transmission for each node and powers down the node when it is not scheduled to transmit. Using PAMAS, nodes are scheduled to transmit data if they do not overhear transmission. Otherwise, they are powered down since existing transmission means that other nodes in the neighborhood cannot transmit data without causing collision. A widely-used method in wireless networks which is standardized as a part of the IEEE 802.11 standard [52] is a beaconing method. Figure 2.3 shows

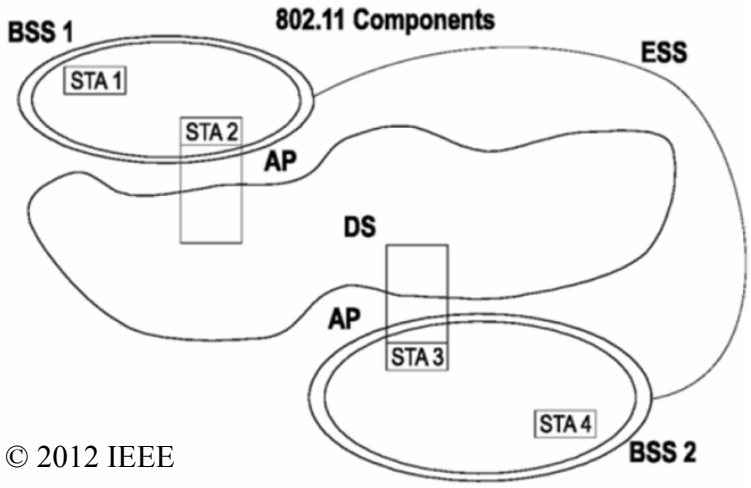



Figure 2.3: Components of an IEEE 802.11 wireless network (taken from [52])

the components of a IEEE 802.11 wireless network. In this figure, STAs are the wireless stations, which can be mobile (wireless devices such as cell phones) or fixed (Access Points, or APs). A collection of STAs and an AP forms a Basic Service Set (BSS). In the beaconing method, the wireless device (STA) informs the AP that it will put its network interface to sleep. While the interface is in sleep mode, the AP buffers the packet destined to it and sends a beacon packet periodically. The interface wakes up periodically to listen to the beacon from the AP and receive its packets. Some other sleep and wakeup scheduling methods for wireless networks are synchronized. An example is Fully Synchronized Pattern [60] in wireless sensor networks. In Fully Synchronized Pattern, all the nodes in the network are put to sleep at the same time for a fixed period of time. Upon expiration of this period, all the nodes wake up and stay on for a fixed period, and the process repeats. Although this approach assures that all the nodes are active at the same time, it has two caveats. First, turning on some of the nodes may not be necessary during all active periods and second, time synchronization among all the nodes is difficult. However, this approach is very trivial to implement. Since the next chapter will focus on improving two scheduling methods for LAN switches and Ethernet interfaces, reviewing scheduling methods in switches and individual Ethernet interfaces will be the subject of the rest of this section. Note that scheduling methods in servers such as PowerNap [72], are also used extensively.

#### 2.2.1.1 Scheduling Methods in Ethernet LAN Switches

Gupta et al. [39] investigated the possibility of putting the interfaces and other components of a LAN switch to sleep during periods of low utilization. The LAN traffic they collected showed that there are significant periods of inactivity which motivated them to propose an algorithm to put the link to sleep during these periods. Their algorithm estimated the mean time to arrival of the next packet. If the estimation shows a sleep opportunity that compensates for the energy wasted in transition from awake to sleep states, the link goes to sleep and is woken up upon arrival of the next packet. They addressed the possibility of losing the first arriving packet to a sleeping link in a LAN switch by using Hardware Assisted Buffered Sleep (HABS) in which an incoming packet wakes up the link and is buffered. Although HABS solves the problem of losing the first arriving packet, it consumes much more energy than when simple sleeping is used in which any packet received will wake the link up but is lost. They projected 1.8 to 2.5x energy savings using HABS with no loss. However, more energy saving of between 2x and 4.5x is possible using their method provided that simple sleeping was used which resulted in about 7.5 percent of loss.

Gupta et al. [42] also proposed a method and an improvement over it (On/Off-1 and On/Off-2) to determine when and how to turn off the links of an Ethernet LAN switch based upon expiration of a timer or when the load on the link exceeds a threshold. Based on their evaluation of the algorithm, 28% to 84% energy saving is possible using On/Off-2 method while resulting in additional packet delay and loss especially when the traffic is highly bursty. However, Tamura et al. [108] believed that the in-rush current caused by frequently changing the mode of the switch (as proposed by Gupta et al.'s On/Off algorithm) will destroy the circuitry of the switch and thus must be avoided as much as possible. They introduced their Extra Active Period (EAP) model according to which the link does not enter the sleep state even when the buffer is empty. Instead, it remains in active mode for another active period and if the buffer still remains empty after this extra period puts the switch to sleep. They evaluated their algorithm both numerically and analytically using M/M/1 and IPP/M/1 queueing models and concluded that assuming Poisson traffic on the link, their method improved the average number of turn-on instances while lowering the power reduction ratio.

Ananthanarayanan et al. [4] proposed a similar approach to Gupta et al.'s On/Off scheme which eliminates the need for the ports to be on during the idle periods and takes better advantage of extended idle periods. They proposed Time Windows Prediction (TWP) method which predicts the future traffic on each

of the ports of the switch by observing the number of packets in a sliding window of time. If the number of packets observed is below a defined threshold the switch powers down the port. The packets arriving to the powered-down ports are being buffered by means of shadow ports. They report about 30% energy savings using their approach.

Rodriguez-Perez et al. [97] built on the On/Off algorithm to remove its two shortcomings. Their algorithm postpones sleeping if the buffer is not completely empty when the On/Off algorithm decides to go to sleep. Instead, their algorithm waits until the buffer is drained and then determines the sleeping period. The second shortcoming they fixed was that as opposed to On/Off, their method does not try to maximize total sleeping time. Instead, it determines if entering sleep state for the calculated period compensated for the cost to wakeup again or not. If it is determined not to be a "worthy" sleeping interval, the method chooses to keep the link active and idle rather than to put it to sleep. With these two improvements, they show that their method improves both on average delay and energy savings compared to the On/Off algorithm. However, the degree of packet loss is slightly higher when their algorithm is used.

The Buffer-and-Burst (B&B) method proposed by Nedevschi et al. [82] shapes the traffic at network edge routers in order to enable some of the routers inside the network to sleep. In this method, all the edge routers shape the traffic into small bursts. Then, they transmit the bursts at the same time to ingress routers thus enabling the ingress routers to process all the incoming burst with one sleep and wake transition. In other words, all the edge routers synchronize the time of their burst transmissions. Since synchronizing all the edge routers is not practically feasible, the practical method they study is for each edge router to send its bursts at once as a single train of bursts. As a result, bursts disperse as they get further from the originating edge router. Practical B&B showed significant overall energy savings in the network of up to 80%.

The latest scheduling method to reduce the energy consumption of LAN switches is called Periodically Paused Switched Ethernet (PPSE) [76] which is first introduced by Blanquicet and Christensen with a different name (Pause Power Cycle) [13]. In PPSE, all the links connected to a LAN switch are stopped from sending any traffic at the same time for a fixed period of time. Therefore, the entire switch (or at least the majority of its components) can be turned off during this period of time. PPSE is specifically intended for edge network switches since these are the most lightly-utilized switches in the network with many idle periods. Therefore, there is a good opportunity of saving energy on these switches while keeping the adverse effect on the performance of the network as low as possible. PPSE seems similar to and is indeed motivated by Gupta et al.'s On/Off algorithm. However, the two have a fundamental difference. In

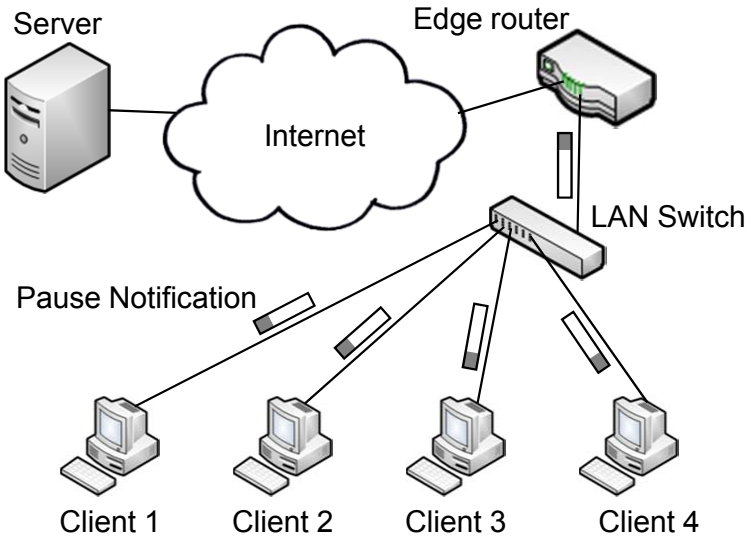



Figure 2.4: High-level illustration of PPSE in a LAN switch (taken from [13])

the On/Off algorithm, the future sleeping opportunities are predicted and the method tries to exploit them by putting the switch to sleep during these times. In PPSE, the sleeping opportunities are artificially made (through scheduling) by the switch by sending Pause Notifications. This difference leads to eliminating the event of packet loss during off times in PPSE since it is already made sure that none of the links sends any traffic during the idle periods. Moreover, since PPSE stops all the links at the same time, more components of the switch can be powered off compared to On/Off and other approaches described earlier. Figure 2.4 shows a high-level view of how PPSE works. A notification message (referred to as Pause Notification hereafter) indicating that the system connected to a link must not send any traffic for an arbitrary interval is sent by the switch on all the links connected to it. The switch then enters a low-power state in which a number of its components are powered off (off period). When the interval elapses, the switch resumes to the fully operational state and resumes servicing packets (on period). The fraction of time that PPSE enables the switch to sleep is determined by the duration of on and off periods. However, these periods are predetermined, which can cause instability in the buffers of systems connected to the switch. If the load on the switch is high (which happens rarely) or if there is sudden burst of packets (which can happen often due to the bursty nature of network traffic) the switch may not be able to transmit all the packets in an on period. This causes blocking of packets in the connected systems. and this may cause a very high delay for the packets. The root of the problem is that PPSE does not adapt to load. A part of the next chapter will be

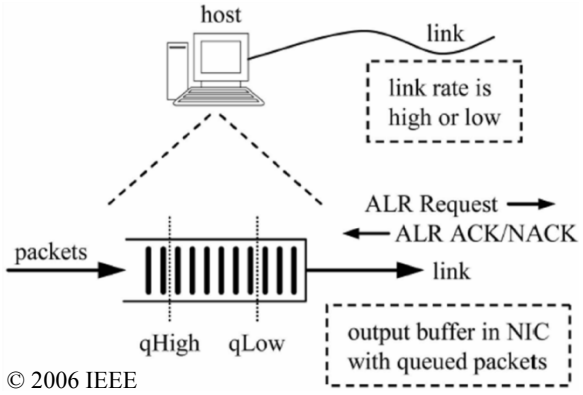



Figure 2.5: Adaptive Link Rate in an Ethernet link (taken from [37])

focused on demonstrating this inefficiency using simulation and proposing and evaluating an improvement over it to solve the problem.

#### 2.2.1.2 Scheduling Methods in Ethernet Links

In addition to Ethernet LAN switches, scheduling methods (using both sleep and wakeup and rate adaptation approaches) have been studied for Ethernet links. The first scheduling method for Ethernet links was proposed by Gupta et al. [40]. This sleep and wakeup method, called Dynamic Ethernet Link Shutdown (DELS), determines if putting the link to sleep is feasible or not based on the number of packets already buffered in the link's transmission queue as well as the mean interarrival time of the packets. If putting the link to sleep is determined to be feasible, the length of the sleep is computed based on a given maximum sleep interval and a maximum buffer size. Arriving packets to the link when in sleep mode are buffered to be transmitted when the sleep period is over. The method is shown to save significant energy for light link loads of less than 5%, but fails to save any significant energy for higher loads. The increase in delay caused by this method is reported to be of the order of less than 1 ms.

Scheduling methods using rate adaptation have also been explored for wired networks. The first such method was Adaptive Link Rate (ALR) method which was proposed in 2005 by Nordman and Christensen [89]. ALR is evaluated both analytically and by simulation for Ethernet links and switches in [37], [36], and [38]. ALR switches the rate of the link between high and low based on the queued packets in the link's buffer. The simplest rate change policy of ALR (Dual Threshold Policy [37]) uses two thresholds on the link's output buffer, low and high thresholds (qLow and qHigh in Figure 2.5). If the

number of packets in the queue exceeds the high threshold, the rate is changed to high. If it decreases below the low threshold the rate is changed to low provided that both sides of the link agree on the rate change. This policy causes rate oscillation under certain traffic loads which increased the response time and variability. To minimize link rate oscillation, another policy is proposed for ALR (Utilization-Threshold policy [38]) according to which the number of bytes counted during a time period is used in order to make the rate change decision. The ALR method is inspired by the power reduction approach taken in new versions of Asynchronous Digital Subscriber Line (ADSL), ADSL2 and ADSL2+. In these versions, three power modes are defined; L0 in which the ADSL link is fully functional and consumes the maximum power, L2 in which the link is still active but works at a reduced rate and consumes less power, and L3 in which the link is idle and powered off [35]. The link transitions from L0 to L2 when the link's transmission buffer is empty or almost empty and to L3 mode when the user is not online [111]. The most recent rate adaptation method, presented in [82], is called practRA. PractRA is a similar approach to ALR with a few differences. Most importantly, practRA uses just one threshold to determine rate change and avoids oscillation by using the packet arrival history to predict the rate in near future and only changes the rate when made sure that the future link utilization is not more than the high rate.

The most recent advancement in making Ethernet links more energy efficient is Energy Efficient Ethernet (EEE) [51]. EEE is a sleep and wakeup approach which makes Ethernet links close-to-energy-proportional. Estimates show that using EEE in all current 1 Gb/s edge links in both residential and commercial buildings and network equipment links within residences could save about $180 million/year in the US alone [24]. Two modes are defined in EEE; Active mode and Low Power Idle (LPI) mode. In Active mode the link is powered on to transmit packets. When there are no more packets to transmit, the link can enter the LPI mode in which the physical layer is powered off and elements in the receiver are stopped. Upon the arrival of a packet to the link, the link can wake up in a few microseconds to resume packet transmission (Figure 2.6). In Figure 2.6, $T_s$ is the time needed to enter the LPI mode and $T_w$ is the time needed to return to Active mode. During the $T_w$, $T_s$ and $T_r$ periods, the link consumes full power while during $T_q$ only almost 10% of the full power consumption of the link is needed [96]. The refresh cycle of duration $T_r$ is a periodic link activity to maintain the alignment of the receiver's elements to channel conditions. It can be assumed that the link uses the same power as in Active mode during transitions [96]. The minimum $T_w$ and $T_s$ for 10GBASE-T links are 4.48 and 2.88 $\mu$s respectively [51]. The transition times are high compared to the transmission time of 1.2 $\mu$s for a 1500-byte packet at 10 Gb/s. For instance, if the link wakes up to transmit a single 1500 byte

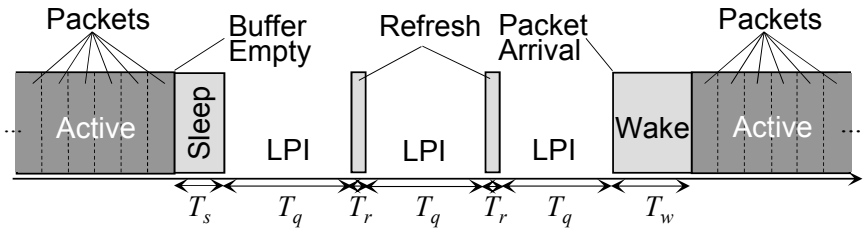



Figure 2.6: Transitions between Active and LPI modes in EEE

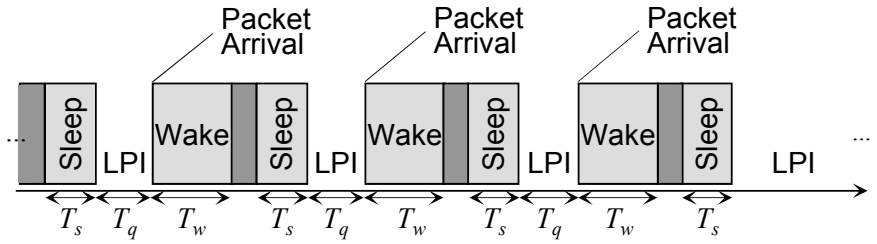



Figure 2.7: Close-to-worst case for EEE energy efficiency

packet, it would spend 1.2 $\mu$s transmitting the packet and 7.38 $\mu$s for transitioning from the LPI mode to the Active mode and back. This means that only about 14% of time is dedicated to transmitting the packet and the rest to the transitions. This inefficiency of EEE was first explored by Reviriego et al. [96] in 2009. EEE can be most efficient when packets arrive back to back in bursts. As a result, only one sleep and one wakeup transition is required per burst which makes the percentage of time the interface spends in Active mode close to the link utilization. This best case often occurs in the form of file downloads using TCP where large blocks of data are burst onto a link from a server to a client at a high rate. Conversely, the worst case happens when packets arrive with a fixed interarrival time and a spacing greater than the wake and sleep transition times. As a result, one wake and one sleep transition would be required for transmission of each packet resulting in inefficient operation. A close-to-worst case traffic scenario occurs when TCP ACKs are being returned from a client to a server. TCP ACKs are typically small packets and are spaced-out evenly (Figure 2.7). The inefficiency of EEE can be reduced by coalescing the outgoing packets into bursts thus decreasing the number of necessary transitions to one per burst. The trade-off is added delay to packets resulted by coalescing and the possible effects of this delay on user experience and Internet protocols such as TCP. Packet coalescing for EEE and its trade-offs will be studied in the next chapter.

### 2.2.2 Substitution

A fundamental problem with putting network-attached systems to sleep in wired networks is that the system will lose its presence in the network. Losing network presence is the inability to respond to certain types of traffic. For instance, if a network-attached system does not respond to ARP packets, the LAN switch will assume that the system is not in the network and will drop packets destined to it. As another example, during an established TCP connection (for instance for an SSH session), if the client does not send ACK packets to the server, the connection will be dropped after a defined timeout. This problem has resulted in a type of energy consumption of an ICT system called the induced consumption which is the energy consumption of a system only due to being present in the network. Early substitution methods were motivated by this problem – they put a system (a PC, for instance) to sleep but configured another system in the network to maintain the presence of the sleeping system by answering certain network traffic on its behalf. The system which maintained the presence was called the proxy and this approach was called proxying. Often the network presence of a system, or multiple systems, is maintained by a low-power proxy and the systems are put to sleep. The proxy wakes up the main system only when a message containing a request for a service provided by the main system is received. The energy savings are gained because of the difference between the power consumption of the systems and the proxy. Note that it is possible to put a network-attached system to sleep and the system still maintains presence in the network. However, such approaches often need a change in network protocols such as TCP, which makes them very difficult to implement. In [55], for instance, a modification to the TCP/IP protocol which adds the capability of sleep to the client is proposed. The TCP/IP protocol with this added capability is called Green TCP/IP. The development of this capability is done by adding a TCP header, called TCP_SLEEP, which notifies the server that the client is going to sleep. Upon receiving a packet with this header activated, the server disables all internal TCP/IP instructions which would normally drop the connection after a period of inactivity by the client. The client can then sleep without the connection being dropped. When the client wakes up by sensing user activity, it send a normal TCP packet to the server and resume normal activity.

The first proxying method for energy purposes was proposed by Christensen and Gulledge in 1998 for desktop PCs [22]. The proxy studied in [22] enables multiple PCs to sleep for extended periods of time by answering routine messages necessary for maintaining their presence in the network. Figure 2.8 shows a client and a proxy in a LAN. The proxy receives (1) and responds to (2) ARP packets on behalf of the client.

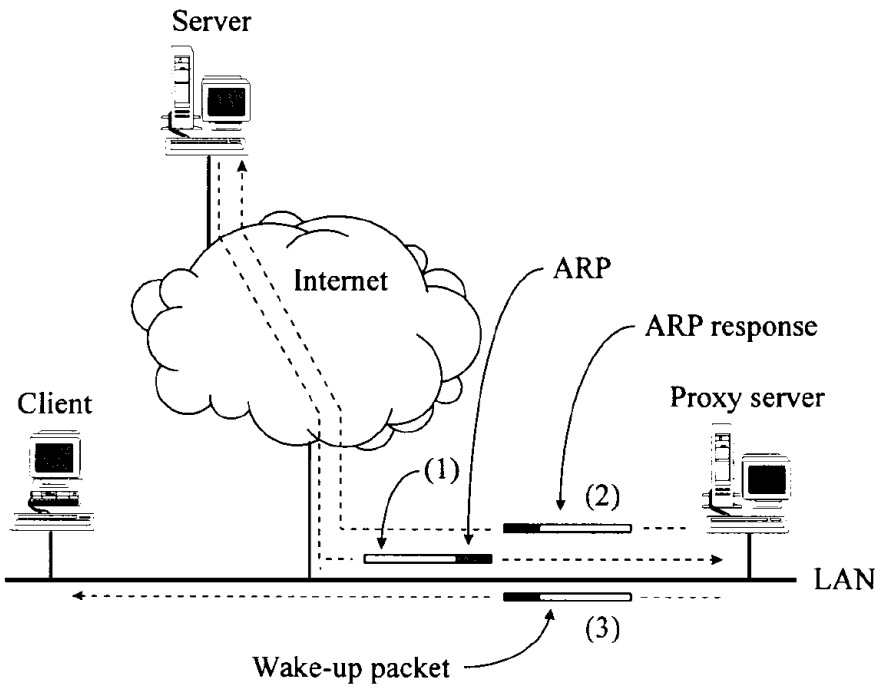



Figure 2.8: Proxy server covering for a client (taken from [22])

When the client needs to respond to a message, the proxy wakes it up using a wakeup packet (3). Proxying for other services such as the HTTP protocol [23], and the Universal Plug and Play (UPnP) protocol [62] were also studied. In all these proxying methods, the proxy responds to routine network messages such as ARP and SSDP discovery messages on the sleeping system's behalf.

Nedevschi et al. proposed four different classes of power-saving proxies based on the type of traffic they should be able to handle on behalf of the main system [81]. They identified three types of traffic that the proxy may encounter. The first type is ignorable traffic. These packets can be safely dropped (ignored) by the proxy without causing the main system to lose its presence in the network or appearing to clients as if it no longer provides the services it should. The second type is the traffic that can be handled via mechanical response. Traffic related to protocols for which the response (or requests) could be constructed with little of no information about the state of the server fall in this category. The third category is the traffic which requires specialized processing to be handled. A significant complexity in the design of the proxy will be required if this type of traffic is to be handled by the proxy. These traffic types are summarized in Figure 2.9. Based on the behavior of the proxy when encountering each type of the above traffic, the authors defined four types of proxy. The first proxy is the simplest where the ignorable traffic is ignored by the proxy and the main system is woken up for handling the rest. The second type is more complex. It ignores all ignorable

| Ignorable | HSRP, PIM, ARP (for others), IPX, LLC, EIGRP, DHCP | |
|---|---|---|
| Mechanical Response | Protocol | State |
| | ARP | IP address |
| | NBNS | NB names of machine and local services |
| | SSDP | Names of local plug-n-play services |
| | IGMP | Multicast groups the interface belongs to |
| | ICMP | IP address |
| | NBDGM | NB names of machine and local services. Ignores pkts. not destined to host, wakes host for rest |

Figure 2.9: Protocols that can be handled by ignoring or by mechanical response (taken from [81])

traffic and handles the ones which need mechanical responses. The third proxy maintains a small set of applications chosen by the user to be handled by the proxy. These application are chosen from the third category of traffic explained above. Finally, the forth proxy is the identical to the third, except that it wakes up the main system for some other applications such as virus updates.

Agarwal et al. proposed a design for a network proxy in enterprise LAN environment, called SleepServer [3]. Their design works as follows: one (or more) SleepServer is added to a subnet of an enterprise LAN. The SleepServer maintains an image (in the form of a virtual machine) for each proxied host that, when activated, maintains network presence and performs a portion or all services of the proxied host. The images are maintained by a Virtual Machine Monitor (VMM). While it is possible to run an image as a separate process of the SleepServer, the authors chose a VMM-based approach since VMs typically support existing operating systems with all their standard protocols. Average savings of 60% was shown to be possible in an enterprise using SleepServer. Agarwal et al. also proposed Somniloquy [3], an augmented Ethernet interface with a low-power secondary processor and storage which allows the a PC to sleep while some or its ongoing network connections are maintained by the second processor.

Substituting methods have also been used to move process and storage – which consume high energy – from battery-powered systems to plug-in systems ([33] and [58], for instance). These approaches, where some of all functionality of a system is transferred to another, have also been used for data servers to put servers to sleep and reduce their operational cost. The focus of Chapter 4 is a substitution method for data servers using two heterogeneous platforms. So, a brief history of such substitution methods for data servers follows.

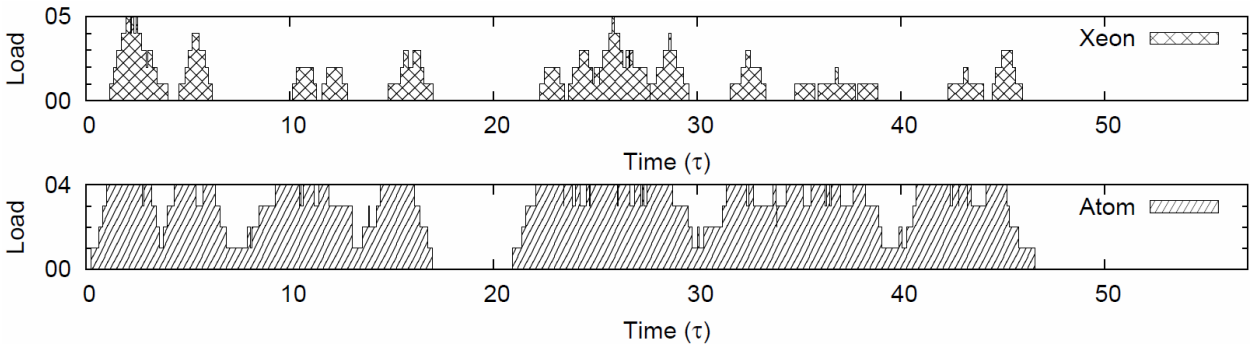



Figure 2.10: Hybrid server operation – top: Xeon load, bottom: Atom load (taken from [25])

The idea of using two heterogeneous platforms in a server for energy efficiency was first proposed by Chun et al. [25]. Prior to their work, substitution methods for energy efficiency in datacenters typically focused on heterogeneous architectures. For instance, Andersen et al. proposed the Fast Array of Wimpy Nodes (FAWN) architecture [6] which is a cluster comprised of embedded AMD Geode CPUs, and flash storage for each CPU. They showed that FAWN achieves almost two orders of magnitude higher queries per joules than conventional disk-based systems. Chun et al. proposed coupling a low-power low-performance Intel Atom platform with a high-power high-performance Intel Xeon platform and switching the serving platform between the two in response to load changes [25]. They used a threshold called $T_{atom}$ to determine when to switch between the platforms. They proposed two switching methods. The first is that the Atom platform wakes up the Xeon platform when the load exceeds $T_{atom}$, migrates the tasks that are currently running on it to the Xeon platform, and goes to the sleep state. The reverse process happens when the load falls below $T_{atom}$ where the tasks are migrated back to the Atom platform and the Xeon platform goes to the stand-by state. The second method they investigated was the same, except that instead of migrating current tasks, tasks are completed on the same platform on which they are started before the platform is powered down. Figure 2.10 shows a snapshot of the load on the two platforms for a sample run of the hybrid server. As can be seen in this figure, the Xeon platform only handles the excess load when the load increases to higher than the predefined threshold (4 concurrent processes on the CPU in this example). When the load decreases to below the threshold, the Xeon processor is put to sleep, as indicated by no load on the upper portion of the graph (note that this figure is for the case where tasks are always completed on the same platform on which they are started).

What is lacking in [25] and is a key challenge in a hybrid server is how to seamlessly switch between the platforms in such a manner as to ensure that there is no intervals during which a client cannot access the

server. Chapter 4 addresses this challenge. In Chapter 4, a new architecture for an energy-efficient hybrid web server, called SME Web Energy Efficient Platform (SWEEP), is proposed and evaluated.

### 2.2.3 Consolidation

Methods of consolidation for energy efficiency started in the area of wireless sensor networks. In sensor networks, nodes send their gathered data to a base station. It is possible for each node in a sensor network to directly transmit their data to the base station. However, if they do, the power needed for transmission – especially if the base station is far from the node – will soon drain the node's battery. Many consolidation methods in sensor networks aim to solve this problem by aggregating the data from a group of sensors (called a cluster) to one node that is preferably close to the base station (called a cluster head), and use that particular node to transmit all the data to the base station. This is energy-aware routing. For instance, a method called Low-Energy Adaptive Clustering Hierarchy (LEACH) [47] uses a fixed-size subset of the sensor network nodes as cluster heads. By rotating the role of cluster head among all nodes, LEACH ensures that the energy load is shared fairly. Other consolidation methods for sensor networks are studied in [46], [65], and [67]. PEGASIS [67] improves on LEACH by forming a chain topology to reduce the transmission energy when some nodes are very far from the base station. The method proposed in [46] assigns a higher probability of being selected as the cluster head based on the remaining battery life of a node. This method is useful in heterogeneous sensor networks. CODA [65] further divides the network into groups where each group consists of several clusters. The further the group to the base station, the more clusters will form in it, which results in a longer lifetime for the network compared to LEACH.

Energy-aware routing is also used in wired networks as a consolidation method. Wired networks were originally designed without energy consumption in consideration. So, there typically is a high degree of redundancy in the routes between two hosts in the network. Energy-aware routing seeks to put to sleep some parts of the network as a function of the traffic load, consolidate the traffic on the active nodes, and save energy in the network as a whole. While finding the minimum subset of resources in the network that maintains connectivity is the goal of these methods, guaranteeing an acceptable quality of service is their limiting factor. Energy-aware routing has been investigated for various networks such as ISP networks [20], backbone networks [21], and optical networks [110].

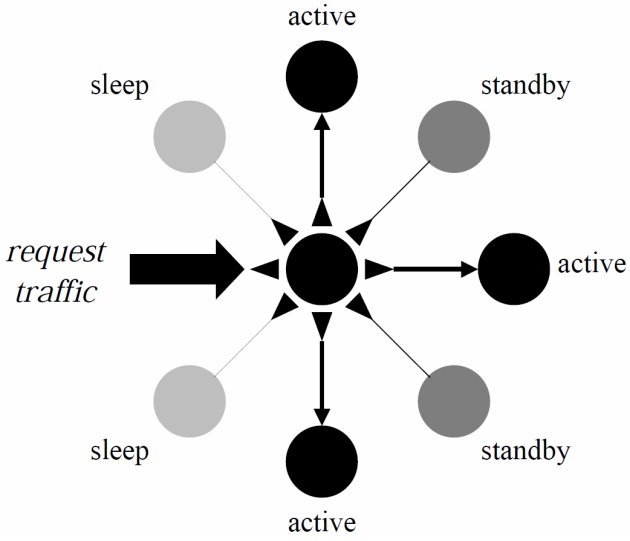



Figure 2.11: Energy-Conscious Server Switching (taken from [19])

Consolidation methods were also investigated in server clusters. A simple consolidation method for server clusters was investigated in [19]. Energy-Conscious Server Switching, proposed and evaluated in this paper changes the number of powered-on servers in response to load changes. Figure 2.11 shows the architecture used in this method. In the middle of the figure there is a server switch which is similar in nature to a load balancing switch. Traffic flows to the server switch where it is distributed to the active servers. The number of the servers is $N$, and $T$ is a defined threshold for the average traffic flow to the cluster. If the server switch detects an average load less than $TN/(N-1)$, it selects a victim server $V$, puts it to a lower-power state, and distributes its assigned load to the other $N-1$ servers. The opposite process is done when traffic increases to above $T$, in which case the server switch wakes up a server and moves some of the traffic to newly-activated server.

Chase et al. proposed an operating system for a server housing center which allocates resources to services with energy efficiency as the primary goal in consideration [18]. The operating system, called Muse, consists of four parts (Figure 2.12): 1) A server pool which is an array of generic and interchangeable servers that serve the requests of a service offered by the server center, 2) reconfigurable switches, which distribute the incoming traffic to the servers considered for serving the load, 3) load monitoring and estimation modules, which are embedded inside switches and server operating systems and continuously monitor the utilization levels on the machines and estimate future load, and 4) the executive, which is the entity that reconfigures the network and the server pool to handle the provisioned future load. Services bid for

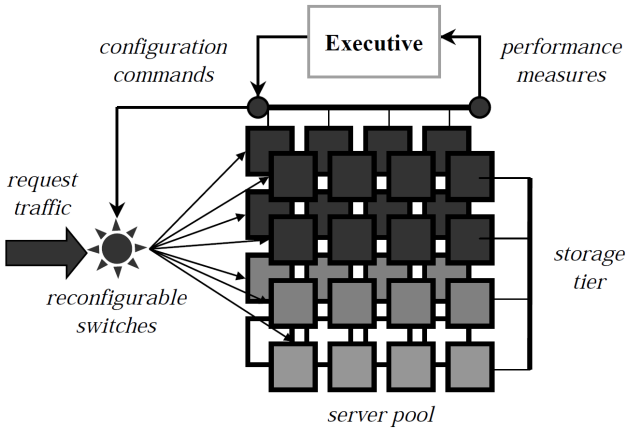



Figure 2.12: The four parts of Muse (taken from [18])

resources by offering to pay a price for getting a level of performance. The executive compares the cost of resources to allocate to the service (energy costs being one element in the overall cost) and allocates the resources which maximize the server center's benefit. They report up to 29% energy savings in the center if their approach is adopted.

Many datacenters use server virtualization to consolidate work on fewer physical machines. Server virtualization hides the hardware specifications of a physical server's resources such as the number of servers, CPU, operating systems, etc. from the server software. Using an extra application layer, a physical server machine can be virtually divided into multiple virtual environment, each of which looking exactly as an individual server environment to the server software (Figure 2.13). Virtualization can be supported at hardware level [83], or at software level using solutions such as Xen [9] or VMware [107]. While virtualization methods are not direct power management methods, they enable consolidation of work into fewer systems, which results in less physical servers and, consequently, less power consumption.

### 2.2.4 Combination of Methods

As can be seen in Figure 2.2, scheduling can be combined with consolidation and substitution. In fact, some sort of scheduling is typically needed in the two other methods in order to provide the opportunity of substitution or consolidation. For instance, in proxying methods for PCs such as the ones studied in [2] and [22], the flow of packets to the proxied systems is delayed until the system wakes up. In hybrid servers such as the one studied in [25], the service time of requests is changed by moving them to a lower, or higher,

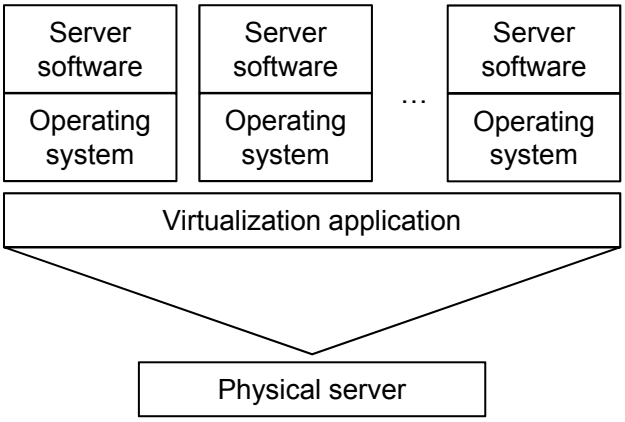



Figure 2.13: Server virtualization

performance platform. Scheduling plays an important role in consolidation methods as well. For instance, energy-aware routing methods can change the service time of requests by modifying the route on which they flow. Consolidation methods for server clusters, such as the method studied in [19], also increase the service time of the load on a server by consolidating more workload on it until a threshold is exceeded. Then, the service time is decreases by moving a portion of the load to another server.

Combination of methods can also be viewed from another perspective. In some cases, a combination of distinct methods can be used together to maximize energy savings. For instance, Mahadevan et al. explored a combination of Link State Adaptation (LSA), Network Traffic Consolidation (NTC), and Server Load Consolidation (SLC) for gaining better energy consumption in datacenter environments [71]. LSA is adjusting the state of a link (namely, disabled, 10 Mb/s, 100 Mb/s and 1 Gb/s) to best match the traffic on the link. NTC is a consolidation scheme in which traffic is routed on fewer links and switches in order to save power by turning off the unutilized links. SLC is a consolidation scheme for server load where jobs are migrated to a fewer number of servers and unutilized servers (and their ports and switches) are turned off. It is shown that only adjusting the active links by using LSA can results in 16% power savings with no performance penalty and up to 75% savings is possible is by also using traffic management and SLC.

## 2.3 Chapter Summary

In this chapter, the parts in the life cycle of an ICT system which consume energy were explained. The part on which this dissertation is focused is the energy consumed by a system while in operation.

A background and literature review of Dynamic Power Management (DPM) methods at the system level was then given and the major related DPM methods were reviewed and categorized in a new taxonomy comprised of three broad areas, scheduling, substitution, and consolidation. Table 2.1 summarizes the reviewed methods. Methods that follow in the next chapters of this dissertation belong to the scheduling and substitution areas. The rest of this dissertation addresses the following problems:

- How can the energy-inefficiency of EEE resulted by high transition overhead be reduced by coalescing the outgoing packets into bursts? What are the trade-offs and possible effects of this method on user experience and typical Internet traffic?
- In a heterogeneous hybrid server, how can the platforms be seamlessly switched in a manner so that there is no interval during which a client cannot access the server?
- How can coalescing of requests be utilized in a hybrid server to achieve energy-proportional operation?

Table 2.1: Summary of the reviewed DPM methods in ICT systems

| | Scheduling | Substitution | Consolidation |
|---|---|---|---|
| Ethernet Switches | - (Un)Coordinated Sleeping [41]<br>- HABS [39]<br>- On/Off 1 & 2 [42]<br>- Unnamed [108]<br>- TWP [65]<br>- Unnamed [4]<br>- Buffer & Burst [82]<br>- PPSE [13] and [76] | | |
| Ethernet Links | - DELS [40]<br>- ALR [89], [36] and [38]<br>- practRA [82]<br>- EEE [51] | | |
| Servers | - PowerNap [72] | - FAWN [6]<br>- Hybrid Server [25] | - Energy-Conscious Server Switching [19]<br>- Muse [18]<br>- Virtualization [83] |
| Wireless and Sensor Networks | - STEM [100]<br>- PAMAS [103]<br>- IEEE 802.11 Beaconing [52]<br>- Fully Synchronized Pattern [60] | - Unnamed [33]<br>- Unnamed [58] | - LEACH [47]<br>- PEGASIS [67]<br>- Unnamed [46]<br>- CODA [65] |
| PCs | - Sleep with inactivity timeout<br>- Green TCP/IP [55] | - Unnamed [81]<br>- Proxy for PC [22]<br>- Somniloquy [2]<br>- SleepServer [3] | |

## Chapter 3: Packet Coalescing for Energy Efficient Ethernet

As described in Section 2.2.1.2, EEE uses a Low Power Idle (LPI) mode to reduce power consumption between packet transmissions. EEE has transition times $T_s$ (wake-to-sleep) and $T_w$ (sleep-to-wake), which are significantly greater than a single packet transmission time for both 1 and 10 Gb/s EEE. By coalescing arriving packets into bursts, the overhead of $T_s$ and $T_w$ can be reduced and nearly energy-proportional operation can be achieved. The trade-off in coalescing is increased packet delay at the sender and, potentially, also in downstream switches or routers.

In packet coalescing, a FIFO queue in the Ethernet interface (in the host NIC and switch or router line card) is used to collect, or coalesce, multiple packets before sending them on a link as a burst of back-to-back packets. This FIFO queue is called a coalescer. Packet coalescing is already used in many high-speed Ethernet interfaces – mostly on the receive side – to reduce CPU overhead for packet processing [73]. Packet coalescing can be based on packet count and/or time from first packet arrival. In packet coalescing based on packet count (count-based coalescing), the coalescer collects a certain number of packets before sending them on the link in a single burst. In packet coalescing based on time from first packet arrival, the coalescer sets a timer, called the coalescing timer, to a certain predefined time upon the arrival of the first packet to an empty coalescer. The timer counts down to zero. When the timer reaches zero (or expires), the coalescer sends the packets which are collected in the coalescer on the link.

In this chapter, a deeper analytical understanding of the effects of coalescing on energy savings and delay (both in the coalescer and the downstream queues) is given first, and then two methods of packet coalescing in EEE links and LAN switches are studied.

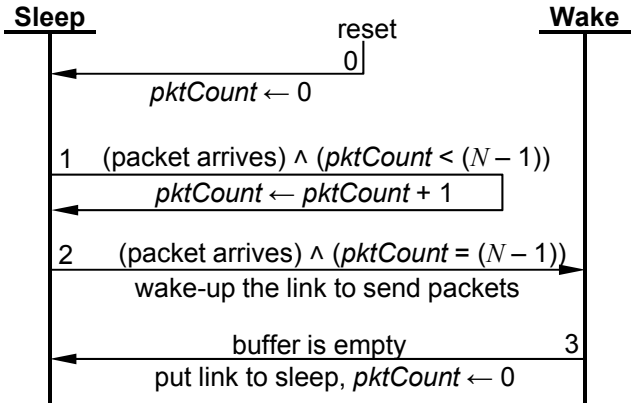



Figure 3.1: FSM for count-based packet coalescing for a sleep-capable link

## 3.1 An Analytical Energy-Delay Model for a Count-based Packet Coalescer

Figure 3.1 shows the Finite State Machine (FSM) for a packet count-based coalescer (or coalescing queue) as implemented in a sender. In the FSMs of this dissertation, the vertical lines represent states, the transitions are shown with horizontal arrows, the events which trigger a transition are listed on top of the transition arrows, and the resulting actions are listed below the transition arrows. The FSM in Figure 3.1 has the following states:

- Wake: The interface is fully powered-on and operational. Packets are queued in the link's buffer and transmitted in a FIFO order.
- Sleep: The link is sleeping. Packets are collected in the coalescer in this mode.

Two variables are defined as follows:

- $N$: Maximum number of packets that are allowed to be coalesced before transmission.
- *pktCount*: The variable which keeps the number of coalesced packets.

A detailed explanation for each of the FSM transitions follows:

- Transition 0: The coalescer starts in Sleep mode. *pktCount* is initialized.
- Transition 1: Upon generation of packets, *pktCount* is incremented until *pktCount* reaches $N$.
- Transition 2: This transition is made when *pktCount* has reached $N$. As a result, the FSM enters the Wake state.

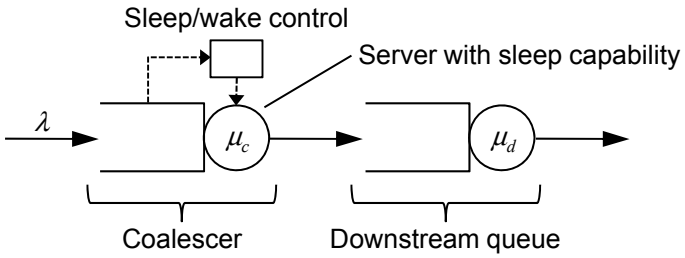



Figure 3.2: Queueing model of a packet coalescer connected to a downstream queue

- Transition 3: The interface has transmitted all the packets in the buffer. The buffer is empty, so the interface starts to power down by entering the Sleep state. *pktCount* is reset to zero.

Figure 3.2 shows the queueing model of a tandem coalescer with a downstream queue. The coalescer could be in a host, and the downstream queue could be in a switch connected to a sleep-capable link. The sleep/wake control unit implements the FSM in Figure 3.1. So, as explained above, the coalescer stores packets in its buffer (when the link is sleeping) until the $N$th packet arrival, and then the link is woken up, and the packets in the buffer are transmitted on the link in a contiguous burst. Packets arriving to the coalescer during the Wake state are included in the burst. When the buffer is empty the link is put to sleep (Sleep state), and the coalescer begins to store packets again. Packets arrive into the coalescer at a rate $\lambda$ and require a service time of $1/\mu_c$ when transmitted on the link and a second service time of $1/\mu_d$ on the outgoing link of the downstream queue. It is essential that $\lambda < \mu_c$ and $\lambda < \mu_d$, since the system becomes unstable otherwise. The following will be developed next:

- An exact expression for the end-to-end delay ($D$) which includes the delay caused by coalescing ($D_c$) and in the downstream queue ($D_d$).

- The total fraction of idle time that can be used for energy savings, $S_G$, which is indirectly a measure of energy-proportionality ($E_P$).

For tractability, deterministic interarrival and service times are assumed. As $S_G$ approaches 1, energy-proportional operation is achieved. In a system where $S_G = 1$, the system is capable of sleeping in all idle periods and for the entire duration of each idle period. In a system where the base power is zero and the system consumes exactly the peak power when active, $S_G$ is equal to $E_P$.

Table 3.1 describes the parameters and outputs in the analytical energy-delay model for a packet coalescer and a downstream queue, which will be developed next.

### 3.1.1 Energy-Delay Model for Coalescer

In EEE-enabled Ethernet links where coalescing is not used, the interval between packets (or gap of duration $t_{gap}$) may be less than $T_s + T_w$ making sleep infeasible (see Figure 1.5a). Although EEE-enabled links would power down even if the gap is shorter than $T_s + T_w$, the link would use more power than it would have if it had stayed powered on during the gap, if there is a power spike when waking up. At best, sleeping in these short gaps will result in no energy saving. With coalescing, the total idle time over a large period of time is the same as without coalescing, but it is now consolidated into larger gaps of duration $t_{gap}$ each (see Figure 1.5 for an illustration of consolidated idle periods). If $t_{gap} > T_s + T_w$, then sleep is possible. Coalescing thus achieves two goals simultaneously, 1) increasing the duration of idle intervals, and 2) reducing the number of idle intervals. Goal 1 enables sleep to be feasible in some cases when it otherwise would not be, and goal 2 reduces the total number of sleep/wake transitions thus reducing the overhead of these transitions. The trade-off is that coalescing increases packet delay. $S_G$ is a measure of how well a system can take advantage of sleeping in idle periods (gaps), and is defined as,

$$S_G = \begin{cases} 0 & \text{if } T_s + T_w \geq t_{gap} \\ 1 - \frac{T_s + T_w}{t_{gap}} & \text{if } T_s + T_w < t_{gap} \end{cases} . \tag{3.1}$$

If no coalescing is used (assuming that $t_{gap} > T_s + T_w$), then the two following cases are possible:

Table 3.1: Key parameters and outputs for the analytical model of coalescing

| | |
|---|---|
| $\lambda$ | Packet arrival rate (to the coalescer) |
| $\mu_c$ | Service rate of the coalescer |
| $\mu_d$ | Service rate of the downstream queue |
| $N$ | Packet count to open coalescer gate |
| $M$ | Total number of packets departing the coalescer |
| $T_s$ | Wake to sleep transition time |
| $T_w$ | Sleep to wake transition time |
| $t_{gap}$ | Duration of gap between bursts after coalescing |
| $t_{sleep}$ | Duration of sleep period after coalescing |
| $S_G$ | Fraction of total sleep time to total idle time |
| $D_c$ | Mean packet delay for the coalescer |
| $D_d$ | Mean packet delay for the downstream queue |
| $D$ | Mean end-to-end packet delay |
| $L_c$ | Mean queue length for the coalescer |
| $L_d$ | Mean queue length for the downstream queue |

- Case 1: the duration of the idle period and thus the arrival of the first packet in the next burst of packets can be predicted [50].
- Case 2: the link wakes up with the arrival of the first packet of the next burst of packets.

In Case 1, prediction of idle period duration is assumed to be possible and thus a wakeup can be scheduled to occur immediately before a packet arrival – the arriving packet does not incur a wakeup transition delay. In Case 2, predicting the duration of idle periods is not possible, so the arrival of a packet is used to trigger a wake up – the arriving packet now does incur an added wakeup transition delay. Case 1 is solved for here and it will be shown later that Case 1 is an approximation of Case 2 which happens in real EEE links.

When no coalescing is used,

$$D_c = \frac{1}{\mu_c}, \tag{3.2}$$

since no queueing can occur ($\lambda < \mu_c$) and the only delay a packet incurs is the transmission delay. Also,

$$t_{gap} = \frac{1}{\lambda} - \frac{1}{\mu_c} \tag{3.3}$$

(the difference between the time to the arrival of the next packet and serving the current packet).

When coalescing, the $N$ packets stored when in the Sleep state are immediately transmitted after the link wakes up and enters the Wake state. Packets that arrive during the transmission of these $N$ packets will also be transmitted, and only when the coalescer buffer is empty is a transition back to the Sleep state made. Then, coalescing begins again, and this pattern repeats. Each of these repetitions is called a coalescing cycle.

The following assumptions are made to keep the analytical model simple and avoid dealing with cases where $t_{on}$ and $t_{off}$ include fractions of time. Assume $\lambda < \mu_c$, normalize $\lambda = 1$ and restrict $N = b(\mu_c - \lambda)$ where $b = 1, 2, \ldots$. All parameters are assumed to be integers.

Let the time it takes to coalesce $N$ packets be $t_{off}$. The first packet arrives at time 0. So, $t_{off}$ ends at the arrival of the $N$th packet starting from time 0. So,

$$t_{off} = \frac{N-1}{\lambda}. \tag{3.4}$$

Let the time between the moment that $N$ packets are in the queue (transmission begins at this moment) until the time that the transmission ends plus the time until the arrival of the first packet of the next cycle be $t_{on}$.

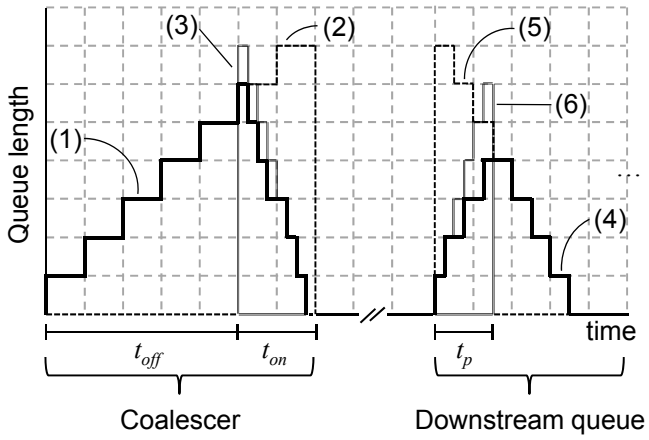



Figure 3.3: Coalescer queue length behavior for $\lambda = 1$, $\mu_c = 4$, $N = 6$, $\mu_d = 2$

Upon entering the Wake state, for each $\mu_c$ packets that are served one additional packet will arrive. This is equivalent to the $N$ packets in the coalescer buffer being served with the rate $\mu_c - \lambda$ until the end of the cycle. Based on this observation,

$$t_{on} = \frac{N}{\mu_c - \lambda}. \tag{3.5}$$

The left side of Figure 3.3 shows the evolution of queue length for a sample coalescer queue with $\lambda = 1$, $\mu_c = 4$, and $N = 6$. Starting at the beginning of a cycle, the queue length increases by one every $1/\lambda$ time increment until $N$ packets arrive. Then, starting from $t_{off}$, it drops by one every $1/\mu_c$ time increment, except every $1/\lambda$ time increment where it remains unchanged (one departure and one arrival occur at the same time). The variable $M$ is the total number of packets that depart the coalescer in each cycle,

$$M = \lambda(t_{on} + t_{off}), \tag{3.6}$$

which is the rate at which the packets arrive times the length of a cycle.

The coalescer average queue length is the average number of packets in the coalescer queue from the beginning of a cycle to the end of the cycle (duration $t_{on} + t_{off}$). The average queue length can be obtained by integrating for the length of a cycle. The area under the queue length function, (1) in Figure 3.3, can be calculated by subtracting the area under an independent queue length function for a queue with $M$ packets where the packets are served with rate $\mu_c$ with no additional arrivals (labeled by (3) in Figure 3.3), from the area under another queue length function with the arrival rate of $\lambda$ for $M$ packets and no departures (labeled by (2) in Figure 3.3). The average queue length, $L_c$, is then the area under the queue length function, (1) in

Figure 3.3, divided by the length of a cycle,

$$\begin{aligned} L_c &= \frac{1}{t_{on}+t_{off}}\left(\sum_{j=1}^{M}(\frac{j}{\lambda})-\sum_{j=1}^{M}(\frac{j}{\mu_c})\right) \\ &= \frac{1}{t_{on}+t_{off}}\left(\frac{M(M+1)}{2\lambda}-\frac{M(M+1)}{2\mu_c}\right) \\ &= \frac{M(M+1)(\mu_c-\lambda)}{2\lambda\mu_c\left(t_{on}+t_{off}\right)}. \end{aligned} \tag{3.7}$$

By Little's Law, $D_c = L_c/\lambda$,

$$D_c = \frac{M(M+1)(\mu_c-\lambda)}{2\lambda^2\mu_c\left(t_{on}+t_{off}\right)}. \tag{3.8}$$

The time $t_{gap} = t_{off} + t_{rem}$, where $t_{rem}$ is the time between the end of serving the last packet in each cycle and the moment the first packet of the next cycle arrives. So, $t_{rem}$ is the difference between $t_{on}$ and the time it takes to serve all the $M$ packets in a cycle, and thus,

$$t_{gap} = t_{on} + t_{off} - \frac{M}{\mu_c}. \tag{3.9}$$

So,

$$S_G = 1 - \frac{T_s + T_w}{t_{on} + t_{off} - \frac{M}{\mu_c}}. \tag{3.10}$$

So far, an expression for $S_G$ and for $D_c$ is derived. The applications of these will be shown later. Next, expressions for $D_d$ and $D$ will be derived.

### 3.1.2 Delay Model for Downstream Queue

The first packet to enter the downstream queue arrives at $t_{off} + 1/\mu_c$ and waits for $1/\mu_d$ time to be served. The next $M-1$ packets arrive $1/\mu_c$ times apart, and it takes $1/\mu_d$ time to transmit each packet. Here, it is assumed that $\mu_c > \mu_d$ (otherwise there would be no queueing in the downstream), and also $\mu_c = k\mu_d$, where $k = 1, 2, \ldots$. The right side of Figure 3.3 shows the evolution of queue length for the downstream queue with $\mu_d = 2$ where the coalescer queue is as shown in the same figure. The queue length grows with rate $\mu_c - \mu_d$ until all the $M$ packets have arrived from the coalescer at time $t_p$ (this from the time of the first arrival to the downstream queue), $t_p = (M-1)/\mu_c$. At $t_p$, the downstream queue reaches its peak

length in the cycle referred to as $p_d$, which is less than $M$ by the number of packets served in $t_p$ time at the rate $\mu_d$. So, $p_d = M - \lfloor t_p \mu_d \rfloor$. Before $t_p$, for each $k$ packets that arrive to the downstream queue, one packet is transmitted. So, the length of the queue increases by one every $1/\mu_c$ time increment, except at every $1/\mu_d$ time increments where the length remains unchanged (one departure and one arrival at the same time). When all the $M$ packets have arrived (at $t_p$), the queue length decreases by one every $1/\mu_d$ time until empty (no additional arrivals).

The average queue length is the average number of packets in the downstream queue from the beginning of a cycle to the end of the cycle, duration of which is $t_{on} + t_{off}$. The area under the queue length function, (4) in Figure 3.3, can be calculated by subtracting the area under an independent queue length function for a queue where packets arrive with rate $\mu_c$ for the time it takes for all the $M$ packets to arrive , $t_p$ (or $M-1$ packets, this is labeled by (6) in Figure 3.3), from the area under another queue length function for a queue with $M$ packets where packets are served at the rate of $\mu_d$ with no additional arrivals (labeled by (5) in Figure 3.3).The average downstream queue length, $L_d$, is then this area divided by the length of each cycle,

$$\begin{aligned} L_d &= \frac{1}{t_{on}+t_{off}} \left( \sum_{j=1}^{M} \left(\frac{j}{\mu_d}\right) - \sum_{j=1}^{M-1} \left(\frac{j}{\mu_c}\right) \right) \\ &= \frac{1}{t_{on}+t_{off}} \left( \frac{M(M+1)}{2\mu_d} - \frac{M(M-1)}{2\mu_c} \right) \\ &= \frac{M\big(\mu_c(M+1) - \mu_d(M-1)\big)}{2\mu_c\mu_d\big(t_{on}+t_{off}\big)}. \end{aligned} \tag{3.11}$$

By Little's Law, $D_d = L_d/\lambda$,

$$D_d = \frac{M\big(\mu_c(M+1) - \mu_d(M-1)\big)}{2\lambda\mu_c\mu_d\big(t_{on}+t_{off}\big)}. \tag{3.12}$$

### 3.1.3 Numerical Results

Figure 3.4 shows the Energy Proportionality ($S_G$) and the end-to-end delay ($D$) for parameters $\lambda = 1$, $\mu_c = 20$, and $\mu_d = 2$. The relationship between these parameters models a 10 Gb/s EEE interface connected to a 1 Gb/s uplink with 5% offered load ($\mu_c = 10\mu_d$ and $\lambda = 0.05\mu_c$). The burst size $N$ was varied between 19 and 513. These values for $N$ are the smallest coalescer size allowed by the assumptions ($b = 1$), and a very large coalescer size allowed by the assumptions ($b = 27$), respectively, to show the energy-delay trade-off of increasing the burst size from small to infinity. No significant energy savings are achieved by

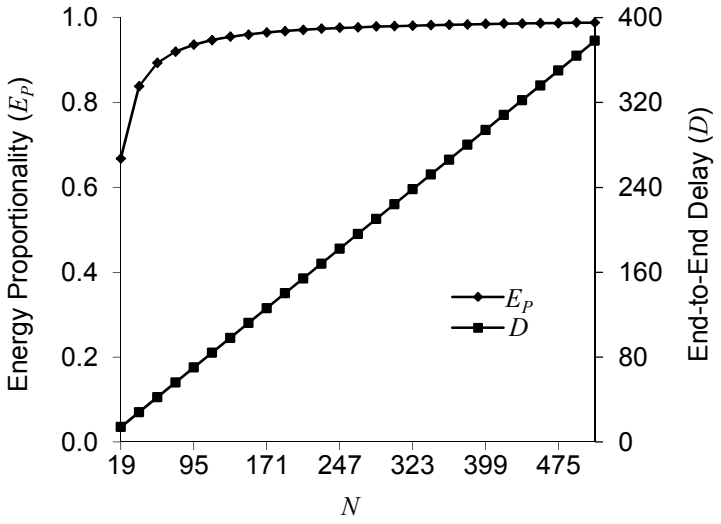



Figure 3.4: Plot of $S_G$ and $D$ for $\lambda = 1$, $\mu_c = 20$, and $\mu_d = 2$

increasing the burst size beyond about 130 packets while the end-to-end delay continues to increase linearly. This is due to the growth of $t_{on}$ with the burst size and an order of magnitude capacity difference of the coalescer and the downstream queue.

Using Equation (3.1), it is possible to directly calculate the value of $N$ to achieve a given $S_G$ value. Assuming that $t_{gap} > T_s + T_w$, from Equation (3.1),

$$t_{gap} = \frac{T_s + T_w}{1 - S_G}. \tag{3.13}$$

Using Equation (3.4), (3.5), and (3.6), and after simplification,

$$N = \frac{\left(\frac{(T_s+T_w)(\mu_c\lambda)}{(1-S_G)(\mu_c\lambda-1)}\left(\lambda(\mu_c-\lambda)\right)-\lambda\right)}{\left(\mu_c-1\right)}. \tag{3.14}$$

For instance, with the same above parameters Equation (3.14) yields $N \approx 125$ for an $S_G$ of 95%.

## 3.2 Reducing the Energy Consumption of EEE by Packet Coalescing

In this section, the simulation model used to simulate an EEE-enabled link which coalesces packets is explained. Experiments and results for evaluating the performance of packet coalescing will be explained.

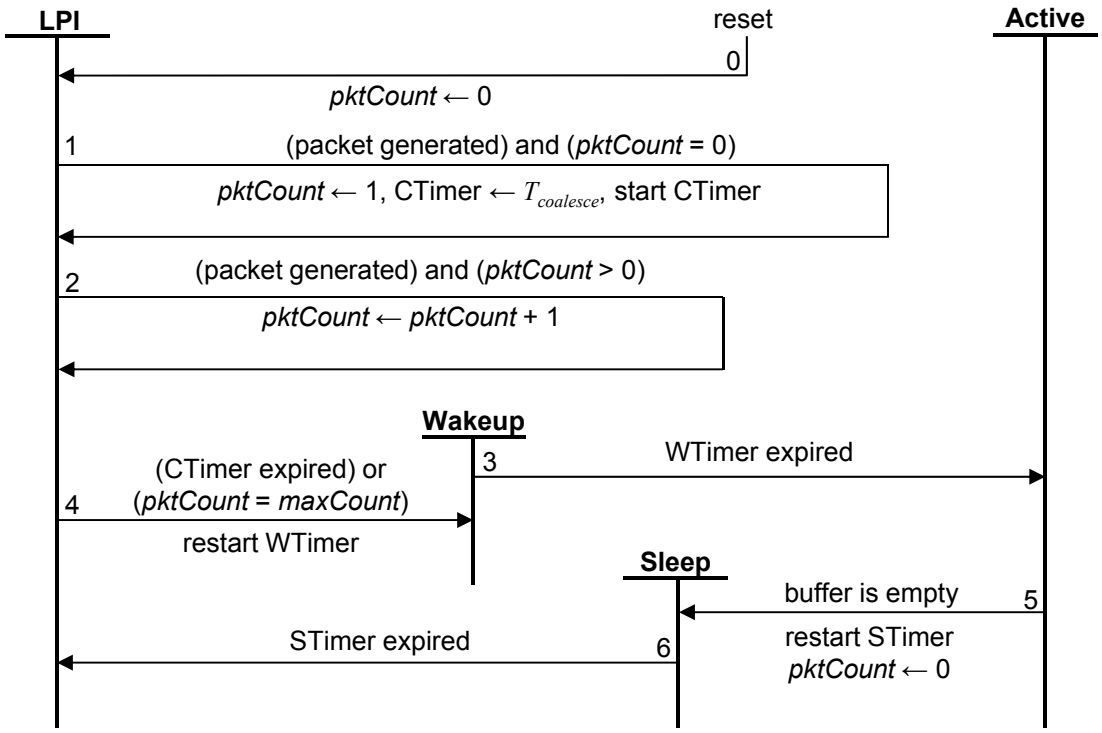



Figure 3.5: FSM for EEE with packet coalescing

### 3.2.1 Simulation Model of EEE with Packet Coalescing

Figure 3.5 shows the FSM for EEE with the ability of coalescing packets before transmission. The FSM has the following states:

- Active: The link is fully powered-on and operational. Packets are queued in the buffer and transmitted.
- LPI: The link is in the LPI mode.
- Wakeup: The link is powering on – it is in transition from LPI to Active.
- Sleep: The link powering down – it is in transition from Active to LPI.

Three timers are defined as following:

- STimer: Maintains the time spent in the Sleep state. Transitioning to LPI mode completes when this timer expires. STimer is set to $T_s$ when restarted.
- WTimer: Maintains the time spent in the Wakeup state. Transitioning to Active mode completes when this timer expires. WTimer is set to $T_w$ when restarted.

- CTimer: The timer which maintains the packet coalescing time. Coalescing resets when this timer expires. CTimer is set to $T_{coalesce}$ when restarted.

Two variables are defined as follows:

- *pktCount*: The variable which keeps the number of coalesced packets.
- *maxCount*: Maximum number of packets that are allowed to be coalesced before transmission.

Note that when a timer is restarted, it means that it is set to its initial value and starts to count down to zero. When the value of the timer reaches zero, it expires. Also note that buffer is a FIFO queue in which the packets are stored and served in the same order they arrived to the queue. A detailed explanation for each of the FSM transitions follows:

- Transition 0: The link starts in LPI mode. *pktCount* is initialized.
- Transition 1: When the first packet is generated, *pktCount* is set to 1 and STimer is restarted which means that STimer is set to $T_{coalesce}$ and starts to count down.
- Transition 2: When packets are generated, *pktCount* is incremented.
- Transition 3: WTimer has expired. The link enters the Active state.
- Transition 4: This transition is made when either *pktCount* has reached *maxCount* or STimer has expired. As a result, WTimer is restarted and the FSM enters the Wakeup state.
- Transition 5: There link has transmitted all the packets in buffer. buffer is empty, so the link starts to power down by entering the Sleep state. STimer is restarted to simulate the time spent to power the link down.
- Transition 6: STimer has expired, so the link enters the LPI state.

Note that the refresh cycles (explained in Section 2.2.1.2) are ignored in this FSM as well as the simulation models for simplicity. A link that works according to this FSM buffers packets in buffer until either STimer expires or *pktCount* reaches *maxCount* when it wakes up and start transmitting the buffered packets back-to-back in the same order they entered the buffer. The FSM remains in the Active state until all the buffered packets and the packets buffered during transmission of the buffered packets are transmitted and buffer is empty. Thus, more than *pktCount* packets can be sent each time the Active state is entered.

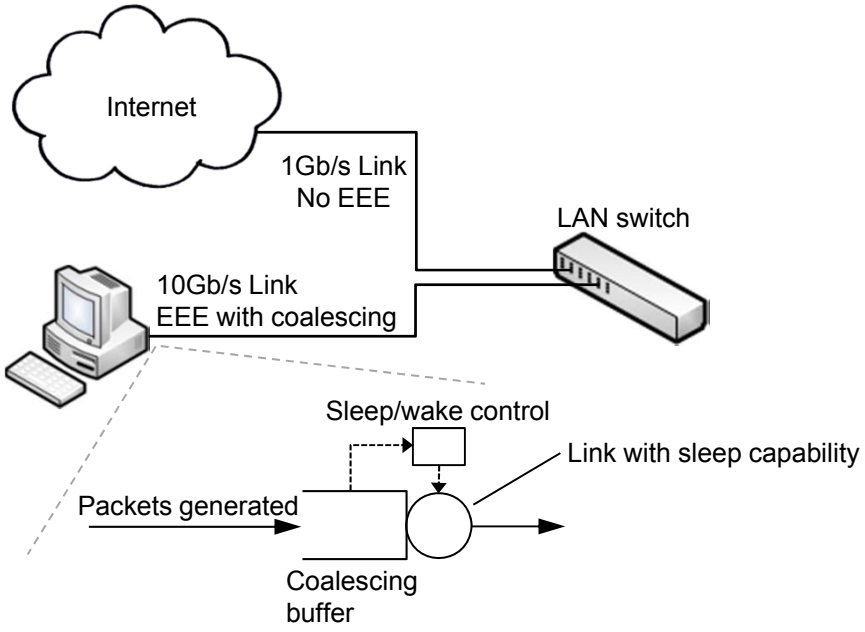



Figure 3.6: High-level illustration of EEE with packet coalescing

A high-level illustration of EEE with packet coalescing is depicted in Figure 3.6. In this model, a PC is connected to an Ethernet switch with a 10 Gb/s EEE with packet coalescing link. The switch is connected to the Internet with a 1 Gb/s Ethernet link. It is reasonable to assume that a 10 Gb/s LAN is used locally in homes or small offices in near future. Packets are generated at the PC and arrive to the 10 Gb/s EEE link with packet coalescing which works according to the FSM depicted in Figure 3.5. After being transmitted, packets are received by the switch and are then transmitted at 1 Gb/s (without EEE) to the Internet. Since the capacity of the link from the switch to the Internet is 10 times lower than the one connecting the PC to the switch, it is expected for the packets to be queued prior to transmission at the switch. This queue will be used to evaluate the increased burstiness caused by packet coalescing. In the simulation model, simulated packets arrive to a single server queue (the server simulates the link) which works according to the FSM in Figure 3.5. The downstream queue is modeled with another single server queue which services the packet in the order in which they are received.

### 3.2.2 Experiments

In the experiment designed to evaluate packet coalescing in this section, the response variables of interest are:

- Power consumption of the link; which is computed as a percentage of the link's consumption when working at full power using the following equation:

$$P_{link} = P_s t_{LPI} + P_a \left(t_a + t_{ws} + t_{sw}\right), \tag{3.15}$$

  in which $t_{LPI}$, $t_a$, $t_{ws}$ and $t_{sw}$ are the percentages of time spent in the LPI mode, Active mode, sleep transition and wakeup transition, respectively. The power consumption in the LPI mode, $P_s$, is assumed to be 10% according to the estimations made by different NIC manufacturers during the standardization process of EEE [96]. The power consumption during transitions is also assumed 100% ($P_a$) also based on estimations made by different NIC manufacturers [96]. The power consumption in Active mode is obviously 100% of the link's consumption.

- Average packet delay

The factors in these experiments are:

- Transition times, $T_s$ and $T_w$; set to their minimums, 4.48 and 2.88 $\mu$s respectively, as stated in [51].

- Load as a percentage of the link's capacity. This factor is varied between 0 and 95%.

- Distribution of packet arrivals and packet size; set to Poisson distribution with fixed packet size of 1500 B.

- Coalescing timer, $T_{coalesce}$ and coalescing buffer size, *maxCount*; two sizes of a coalescer, "small coalescer" and "large coalescer", are simulated here by setting these parameters accordingly. For the small coalescer, $12\mu s$ and 10 packets are used for these factors, respectively. For the large coalescer, $120\mu s$ and 100 packets are used. The value of *maxCount* is chosen to be equal to the number of 1500 B packets transmitted in one $T_{coalesce}$ time.

The load factor and $T_{coalesce}$ and *maxCount* are varied to determine the effects of them on the response variables. Two experiments are done with the small and large coalescer of which results will be presented and explained next. The experiments are done on large enough number of packet arrivals to achieve a 95% confidence level on the mean average delay of the packets. The confidence interval within which the mean point falls is smaller than 10% of the mean value in all cases.

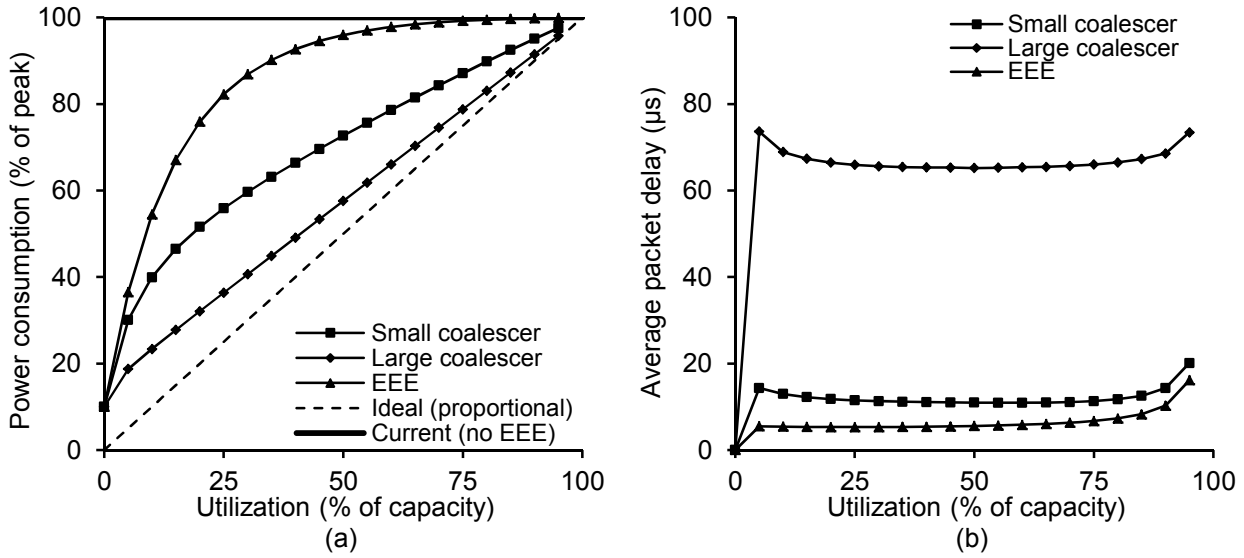



Figure 3.7: Power consumption and packet delay of EEE with packet coalescing

### 3.2.3 Results

Figure 3.7a shows the power consumption of the link as a function of load. The two traces labeled as "Small coalescer" and "Large coalescer" show the power consumption of the link when using the small and large coalescer. The current power consumption of the links (100% of the peak power all the time), the ideal consumption (proportional to load), and the power consumption of EEE without coalescing are also shown in this figure for comparison.

The use of the small coalescer improves the energy efficiency almost half-way between EEE and ideal (proportional). The large coalescer brings the power consumption very close to proportional. For instance, at 15% load, the power consumption using the small coalescer is about 45% of the full consumption, which is over 20% less than what EEE without coalescing yields. The large coalescer yields about 27% power consumption which is only about 4% more than the proportional power consumption.

This improvement in energy efficiency has two drawbacks: increase of packet delay, and increase of the relative burstiness of traffic sent by the Ethernet link.

Packet coalescing results in increased per-packet delay since each packet must remain in the coalescer for some time before being transmitted. Figure 3.7b shows the average packet delay for no EEE, EEE, EEE with small coalescer, and EEE with large coalescer. It can be seen in this figure that EEE adds a small delay to each packet, and the larger the coalescer becomes, the more will the packet delay be. For instance, at 15%

load, the average packet delay of EEE, EEE with small coalescer, and EEE with large coalescer are about 5 $\mu$s, 12 $\mu$s, and 67 $\mu$s, respectively.

The delay caused by EEE is clearly because of the sleep and wakeup transitions. As can be seen in Figure 3.7b, the average packet delay caused by EEE is slightly smaller than the sum of $T_s$ and $T_w$ which is 7.36 $\mu$s. The small difference is caused by the packets which are transmitted without a transition since they are already queued in the interface when the transmission of the current packet ends.

As seen in Figure 3.7b, the packet delay introduced by the small coalescer is between 10 $\mu$s and 14 $\mu$s for any load, which is almost the same as coalescing timer ($T_{coalesce}$). Instrumentation of the simulation model shows that the majority of the bursts occur due to expiration of STimer when the load is low (up to 40% of capacity). As the load increases from 5% to 40%, the number of single packet bursts decreases while the number of multiple-packet bursts increases which results in relatively lower delay. The delay is the lowest for moderate loads but increases with increasing load. When the load is higher (70% and higher), the opposite case happens; most of the bursts are due to the buffer being filled with packets before the timer expires. The delay in this case is caused by 10 packets being transmitted after a 4 $\mu$s wakeup time and 1.2 $\mu$s multiplied by the number of packets ahead of them in the burst. In the best case, all 10 packets arrive to the burst buffer at time 0, so each wait an average of 4.16 $\mu$s + 6.6 $\mu$s. In the worst case, 9 packets arrive at time 0 and the 10th arrives at time 12 $\mu$s. In this case, about 21 $\mu$s delay for each packet is possible.

In the case of the large coalescer which is labeled as "Large coalescer" in Figure 3.7, instrumentation of the simulation model shows that even for very high loads (90% and higher), the bursts occur due to timer expiration not the buffer being filled with packets. But the number of packets in each burst is high. Moreover, $T_{coalesce}$ is much higher than $T_s$ and $T_w$. Since there are a number of packets in each burst, some wait the entire burst timer to be transmitted whereas some that arrived just before the timer expires wait less. Therefore, on average each packet in a burst is delayed for $T_{coalesce}/2$ (60 $\mu$s) plus an additional 5 to 15 $\mu$s which is due to the time the packets waiting for the ones ahead of them be transmitted.

So far, it was shown that EEE with either small or large coalescer adds at most a few tens of microseconds of delay to a packet on average. The significance of a few microseconds per packets delay is very likely to be low compared to tens of milliseconds end-to-end delay of typical Internet connections. However, even this much increase in packet delay in a data center is considered significant, but the additional energy savings gained in a data server may be able to justify a reasonable per-packet delay.

Besides the average, the distribution of the packet delay is also one of the interesting characteristics of the delay. The distribution of packet delay determines how the delay is distributed among all the packets. These points were studied in detail in [74]. The results showed that in most cases, a high percentage of the packets are delayed for $T_{coalesce} + T_w + T_{pkt}$, and the rest of the delays are distributed uniformly. This lowers the chance of certain packets being delayed for too long while others are transmitted instantaneously.

### 3.2.4 Comparison Between the Analytical Model of Coalescing and the Simulation Model of EEE with Packet Coalescing

The analytical model from Section 3.1 was compared with the simulation model of EEE with packet coalescing from the previous section. The setup is the same as what depicted in Figure 3.6. The capacity of the Ethernet link is 10 Gb/s, and it is connected to a 1 Gb/s downstream link. The load on the link is 5% of capacity, and transition times are set to $T_s = 2.88$ and $T_w = 4.48$, as are suggested in the EEE standard [51]. In all the simulations in this section, the link only wakes up when the $N$th packet arrives to the coalescer (that is, Case 2 from Section 3.1.1). Two traffic distributions were used in the simulations; Poisson, and Interrupted Poisson Process (IPP). At large time scales, network traffic is likely to be bursty [66]. However, a Poisson distribution remains a reasonable first-order approximation in cases which the traffic is highly aggregated and the traffic sample is taken in sub-second (small) time spans [57]. The latter distribution, IPP, is among the distributions commonly used to approximate bursty traffic of different types [32]. With the restrictions, the analytical model can be used to simulate EEE links with light offered loads of up to 25%. This 25% load can be modeled as $\lambda = 1$, $\mu_c = 4$, and $\mu_d = 2$ ($b = 1$ and $k = 2$).

In Poisson traffic experiments, packets arrive to the coalescer according to a Poisson process with arrival rate $\lambda$. The coalescer's and downstream queue's service times are exponentially distributed with means of $1/\mu_c$, and $1/\mu_d$, respectively. The variables are normalized so that $\lambda = 1$, $\mu_c = 20$, and $\mu_d = 2$. Table 3.2 shows a comparison of $S_G$ (second and forth columns) and $D$ (third and fifth columns) computed from the model to the simulation model. Figure 3.8 shows the relative difference between the model and simulation as a percentage. It can be seen that as $N$ increases, the difference decreases and the model becomes a closer approximation. For high values of $N$, the difference becomes zero, and the model provides the exact values of $S_G$ and $D$. The most difference is seen when the coalescer capacity is small. This is due to the larger relative duration of transitions compared to transmitting the burst in the coalescer in smaller coalescers.

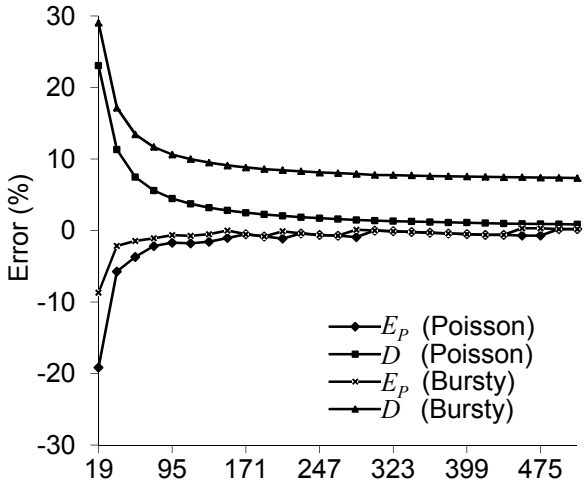



Figure 3.8: Relative error for analytical model versus EEE simulation

The trend of $D$ obtained by the simulation model shows that the end-to-end delay is almost linear even when the arrivals and service are randomly distributed.

In bursty traffic experiments, packets arrive to the coalescer according to an Interrupted Poisson Process (IPP). IPP has two states, On and Off. In the On state, packets are generated with the rate $\lambda$, and in the Off state no packets are generated. Transitioning from On to Off state and vice versa occur with rates $\alpha$ and $\beta$, respectively. The variables are set as $\lambda = 10$, $\alpha = 10$, and $\beta = 10$ for the IPP distribution to simulate short bursts of large number of packets. These parameters yield an overall arrival rate of approximately 1 in the long run. Table 3.2 shows a comparison of $S_G$ (second and sixth columns) and $D$ (third and seventh columns) computed from the analytical model to the simulation model. Figure 3.8 shows the relative difference between the model and simulation as a percentage. As can be seen from the table and the graph, the model also gives results close to that of simulation when the traffic is bursty. As the traffic becomes more bursty, the distribution of the packets distances more from deterministic, causing the model to be a less accurate approximation. Similar to the Poisson case, the highest difference is seen when the coalescer is small.

Based on the above observations, it seems that the analytical model generates results that are reasonably close to the results that can be obtained from lightly-loaded real EEE links for both Poisson and bursty traffic.

## 3.3 Extending Savings of Packet Coalescing Beyond Links in Ethernet Switches

In the previous section, it was shown that packet coalescing improves the energy efficiency of Ethernet interfaces, which results in energy savings in hosts as well as in Ethernet switches. The overall energy consumption of a switch is reduced by EEE as a result of the reduction in energy consumption of individual interfaces of the switch. However, it is possible to extend the energy savings by EEE beyond individual interfaces if all the switch ports enter the LPI mode at the same time, or synchronously. Synchronized LPI periods provide the opportunity for additional components of the switch to power down since it would be certain that there are no packets in any port buffer, or within the switch fabric, that need to be forwarded. In

Table 3.2: Comparison between the analytical and simulation model of coalescing

| | Model | | Sim (Poisson) | | Sim (Bursty) | |
|---|---|---|---|---|---|---|
| $N$ | $S_G$ | $D$ | $S_G$ | $D$ | $S_G$ | $D$ |
| 19 | 0.67 | 14.05 | 0.54 | 17.29 | 0.56 | 17.32 |
| 38 | 0.84 | 28.05 | 0.79 | 31.22 | 0.79 | 31.19 |
| 57 | 0.89 | 42.05 | 0.86 | 45.19 | 0.87 | 45.13 |
| 76 | 0.92 | 56.05 | 0.90 | 59.18 | 0.9 | 59.12 |
| 95 | 0.94 | 70.05 | 0.92 | 73.18 | 0.92 | 73.11 |
| 114 | 0.95 | 84.05 | 0.93 | 87.19 | 0.93 | 87.09 |
| 133 | 0.96 | 98.05 | 0.94 | 101.18 | 0.94 | 101.04 |
| 152 | 0.96 | 112.05 | 0.95 | 115.19 | 0.95 | 115.07 |
| 171 | 0.97 | 126.05 | 0.96 | 129.17 | 0.96 | 129.05 |
| 190 | 0.97 | 140.05 | 0.96 | 143.21 | 0.96 | 143.04 |
| 209 | 0.97 | 154.05 | 0.96 | 157.22 | 0.96 | 157.02 |
| 228 | 0.97 | 168.05 | 0.97 | 171.17 | 0.97 | 171.02 |
| 247 | 0.98 | 182.05 | 0.97 | 185.18 | 0.97 | 185.02 |
| 266 | 0.98 | 196.05 | 0.97 | 199.18 | 0.97 | 199.01 |
| 285 | 0.98 | 210.05 | 0.97 | 213.14 | 0.97 | 213.05 |
| 304 | 0.98 | 224.05 | 0.98 | 227.16 | 0.98 | 227.06 |
| 323 | 0.98 | 238.05 | 0.98 | 241.16 | 0.98 | 241.01 |
| 342 | 0.98 | 252.05 | 0.98 | 255.24 | 0.98 | 255.08 |
| 361 | 0.98 | 266.05 | 0.98 | 269.18 | 0.98 | 268.97 |
| 380 | 0.98 | 280.05 | 0.98 | 283.17 | 0.98 | 282.98 |
| 399 | 0.99 | 294.05 | 0.98 | 297.31 | 0.98 | 297.02 |
| 418 | 0.99 | 308.05 | 0.98 | 311.29 | 0.98 | 311.02 |
| 437 | 0.99 | 322.05 | 0.98 | 325.18 | 0.98 | 324.98 |
| 456 | 0.99 | 336.05 | 0.98 | 339.17 | 0.98 | 338.88 |
| 475 | 0.99 | 350.05 | 0.98 | 353.25 | 0.98 | 353.03 |
| 494 | 0.99 | 364.05 | 0.99 | 367.28 | 0.99 | 367.02 |
| 513 | 0.99 | 378.05 | 0.99 | 381.28 | 0.99 | 380.9 |

synchronized coalescing, the control of when to coalesce and for how long is moved to the switch (from the host interfaces) and the coalescing periods are synchronized on all the ports of the switch.

The target switches for the synchronized coalescing method are the ones mostly used in households and small offices. This type of switch, which is referred to as SOHO (Small Or Home Office) switches hereafter, typically includes 4 to 10 ports and costs less than $100. Two factors motivated the proposal of synchronized coalescing. The first is the typical low utilization of switches in general [39]. The second is that although SOHO Ethernet switches consume only a small amount of energy individually, the number of them deployed in the country is so high that makes their overall consumption significant. So, even small savings per switch would add up to significant overall savings. Using a Kill-A-Watt power meter, the power use of a Linksys EG005W Gigabit Ethernet switch with 4 connected active links was measured as 10 W. As a rough estimate, the current consumption of SOHO Ethernet switches is approximately 7.9 TWh/year based on the number of housing units in the US [112], assuming that about 70% have an Ethernet switch installed, and that each switch is powered on all the time. At the current average electricity cost ($0.10/kWh) this is a total of about $790 million per year in electricity use.

While current Ethernet links and switches are mostly 100 Mb/s and 1 Gb/s, they are likely to evolve to 10 Gb/s in the near future for several reasons including, 1) ever-decreasing prices [85], 2) fast adoption by vendors [80], and 3) increasing bandwidth requirements of multimedia applications within households (for example audio/video transfer between storage device and player, and LAN-based multi-player video games).

### 3.3.1 Switch Energy Use and Transition Times

To determine the possible energy savings from synchronization of LPI periods on all ports in a switch, it is necessary to answer the following three questions:

1. Which components of the switch can be powered down?
2. How much reduction in total switch power use can be achieved by powering down these components?
3. What are the required times to transition these components from fully-powered to powered-down mode and back?

The main component of a typical SOHO Ethernet switch is a single CMOS switch chip. D-Link DGS-1008G, Linksys EG008W, Netgear GS608, and Trendnet TEG S8 are common examples of such switches (all are Gigabit Ethernet switches). The first uses Vitesse VSC7388 SparX-G5 [114], and the rest use Broadcom BCM5398 chips [15], both of which are switch-on-a-chip ICs that include the switching fabric, Ethernet port blocks, interfaces to external CPUs, memories, and the layer 2 packet header processor. The chip is connected to the copper interfaces and optionally to external memories and CPUs. The switching fabric is a high performance bus, shared among all the port blocks and the processor. Each port block consists of a copper PHY, a MAC and ingress and egress packet queues. The packet forwarding tables are maintained in the processor's memory and registers. Packets enter through the PHY interfaces, are passed to the MAC, are put in the port block's egress queue, and are then put on the bus. The header of the packet is analyzed by the packet processor and either the forwarding port is determined, or it is filtered. The forwarding tables are modified accordingly at this point, if needed. The packet is then put in the ingress queue of the forwarding port. It then goes through the MAC and the PHY to the outgoing interface.

A synchronized idle period may allow the switch chip to sleep while maintaining internal state. It is important, however, to empty all the ingress and egress queues prior to power down in order not to lose any packets. This answers question 1.

To answer question 2, the power use of a 10 Gb/s SOHO Ethernet switch must first be determined. Since such switches are yet to be manufactured and marketed for SOHO use, their power use is estimated as follows. The average power use of a Linksys EG005W Gigabit switch with 4 active links was measured as 10 W. 10 Gb/s switches may become commonplace around 2016, roughly 10 years after the standardization of 10GBASE-T, which is the same time span from the standardization of Gigabit Ethernet in 1999 to 1 Gb/s becoming status quo in 2009. Using 100 W consumption as the base (linear relation with capacity increase) and 20% yearly improvement in router power efficiency [8], the power use of future status quo switches over 10 years would roughly be 10 W (that is, the same as today). The CPU and memories of the linecards of a high-end enterprise router consume more than 50% of the linecards power [28]. It is therefore assumed that the breakdown of the power use of a 10 Gb/s SOHO switch will roughly correspond to the breakdown of a single line card of an enterprise router with the CPU and memories embedded in the switch chip. Therefore, it is assumed that by powering down the chip the overall power use decreases to 50% overall.

To answer question 3, the transition times between C0 and C6 states of an Intel i5 multi-core processor (2 ms) were used as a conservative upper bound of the transition time for both the chip and possible external CPUs [14]. The synchronized LPI periods should be at least twice this time to allow the chip to transition to the low-power state and back to the full-power state.

### 3.3.2 The Synchronized Coalescing Method

In synchronized coalescing, all the links connected to a LAN switch are stopped from sending any traffic at the same time for a fixed period of time. The network interfaces at both ends then enter the LPI mode automatically, and the previously described components of the LAN switch can be turned off or put into a sleep mode. Synchronized coalescing is specifically intended for SOHO Ethernet switches since these are the most lightly utilized switches in the network with many idle periods. Therefore, there is a good opportunity of saving energy in these switches while minimizing any possible adverse effect on the performance of the network. Synchronized coalescing builds on the Pause Power Cycle (PPC) idea that was proposed and prototyped prior to the standardization of EEE in 2008 [13].

The same configuration used in [13] (Figure 2.4) is also used here for explaining the method and its evaluation. Synchronized coalescing is implemented in the LAN switch and works as follows: a notification message (referred to as Pause Notification hereafter) indicating that the interface connected to the port must not send any traffic for an arbitrary interval is sent by the switch on all the ports. The switch then enters a low-power mode in which the components mentioned earlier are powered off or put into a sleep state (the Off state). When the interval elapses, the switch powers up to a fully operational state and resumes servicing packets (the On state). The time that the switch spends in Off and On states are called $T_{off}$ and $T_{on}$ respectively. Based on $T_{off}$ and $T_{on}$, a parameter called the Duty Cycle (C) is defined as,

$$C = \frac{T_{on}}{T_{on} + T_{off}}. \tag{3.16}$$

By fixing $T_{off}$ and $C$, $T_{on}$ can be determined as $(C \cdot T_{off})/(1-C)$.

One of the mechanisms that can be used to notify the NIC to stop sending any traffic for a period of time is the flow control mechanism known as PAUSE frames defined in Ethernet standard. However, the traffic stopping part of synchronized coalescing could be implemented using other notification mechanisms as well.

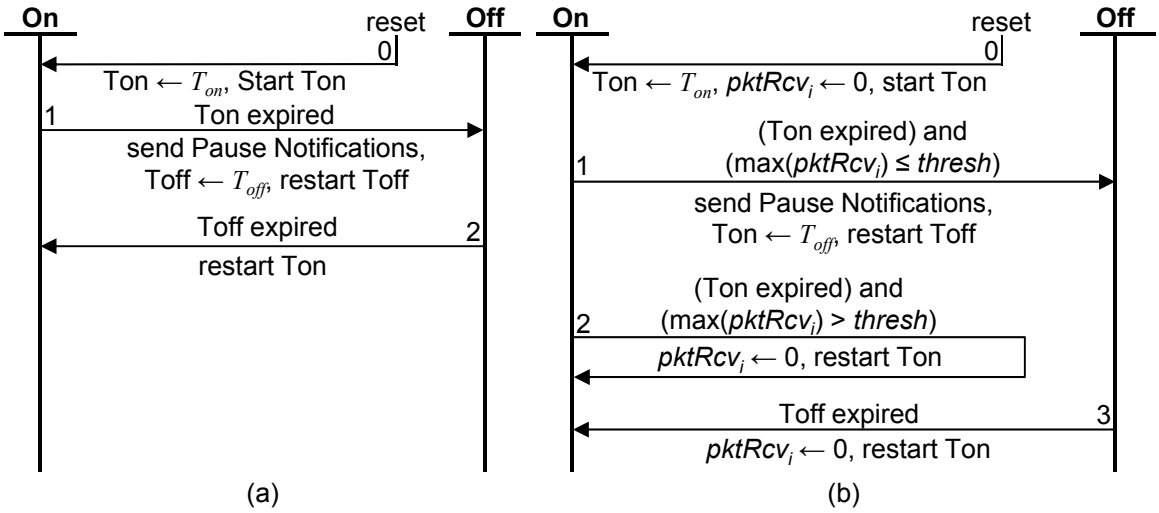



Figure 3.9: FSM for (a) simple synchronized coalescing, and (b) adaptive coalescing

The implementation of synchronized coalescing in real switches requires some sort of Pause Notification to be supported by MAC or PHY layers. PAUSE frames notify the NIC to temporarily stop the flow of traffic (except for MAC control frames) for a certain period of time [53]. PAUSE frames are intended to allow an end of a connection to recover from a congestion state by temporarily stopping the other end from transmitting more packets. By setting MAC Control Parameters field to the value of $T_{off}$, PAUSE frames can be used to make sure that no traffic will be received and powering off will not cause packet loss.

#### 3.3.2.1 Simple Synchronized Coalescing

The simplest version of synchronized coalescing is when the switch stays in the On state for a fixed period of time, enters the Off state, stays there for a fixed period of time, and the process repeats. Simple synchronized coalescing is described by an FSM in Figure 3.9a. This FSM has the following states:

- On: The switch is fully operational in this state.
- Off: The links connected to the switch are paused and the switch is powered down (or sleeping) in this state.

Two timers are defined as following:

- Ton: Maintains the time spent in the On state.
- Toff: Maintains the time spent in the Off state.

The initial values of the timers are $T_{on}$ and $T_{off}$, respectively. A detailed explanation for each of the FSM transitions follows:

- Transition 0: When the FSM starts, or resets, the switch enters the On state, the timers Ton and Toff are set to the initial timer values $T_{on}$ and $T_{off}$, and the Ton timer starts.
- Transition 1: When Ton expires, Pause Notifications are sent on all the links connected to the switch, Toff is reset and starts to count down, and the switch enters the Off state.
- Transition 2: Upon expiration of Toff, Ton is reset to its initial value, $T_{on}$, and starts to count down, the switch returns to the On state, and the entire procedure is repeated.

#### 3.3.2.2 Adaptive Coalescing

The use of simple synchronized coalescing results in large increases in per packet delay especially when the aggregate load on the link is high or when sudden bursts of packets flow to the switch. This effect will be demonstrated and explained further in Section 3.3.4. To reduce this effect, a modification to simple synchronized coalescing is made which regulates the transition to the Off state based on the number of packets received while in the previous On state. Simple synchronized coalescing with this modification is called adaptive coalescing hereafter. Adaptive coalescing is described in the FSM in Figure 3.9b. This FSM has the same states as the FSM in Figure 3.9a. A new array and a new variable are defined as following:

- *pktRcv*: An array of the size equal to the number of links connected to the switch. Each element of *pktRcv* stores the number of packets received from the corresponding link. Index *i* is for the *pktRcv* and ranges from 1 to the number of links.
- *thresh*: The threshold which is compared to the maximum of all $pktRcv_i$s to determine if transition to the Off state should be made.

A detailed explanation for each of the FSM transitions follows:

- Transition 0: Upon start or reset, the switch enters the On state, Ton is set to its initial value, $T_{on}$, and starts to count down. Also, all elements of *pktRcv* are initialized to 0.

- Transition 1: When Ton expires, two cases can happen: 1) the maximum over all elements of *pktRcv* is greater than or equal to *thresh* (this to handle uniformly distributed load among all ports, as well as heavy traffic or sudden burst on a single port), in which Pause Notifications are sent on all the links, Toff is reset, starts to count down and the switch enters the Off state, or 2) otherwise, which will be handles by the next transition.

- Transition 2: In the second case, Ton is reset to its initial value and starts to count down, all elements of *pktRcv* are set to 0 and the switch remains in the On state for another on period.

- Transition 3: Upon expiration of Toff, Ton is reset to its initial value and starts to count down, all elements of *pktRcv* are set to 0, the switch returns to the On state, and the entire procedure repeats.

### 3.3.3 Evaluation by Simulation

The evaluation of the energy savings and performance trade-offs of the methods was done using an ns-2 [84] simulation model. To model synchronized coalescing in ns-2, two timers were added to the wired PHY module of LAN networks to stop and resume passing packets to the lower layer for fixed periods of time. The On and Off periods were synchronized among all the links in a LAN by using static variables shared among all the PHYs.

The experimental factors of the system configuration were bandwidth and the RTT of the server-edge router, switch-edge router link, and switch-client links. The factor of the traffic flows was data flow type – arbitrary packet distribution or FTP. The factors of synchronized and adaptive coalescing were $T_{off}$, $C$, and *thresh*. The response variables were the switch on time (time spent in the On state), download time (for file transfer), and average per packet delay (for arbitrary packet distribution flows). From the switch on time, energy savings could be calculated.

The network configuration shown in Figure 2.4 was modeled in ns-2 for all the simulation experiments in this section. The server and the edge router were connected by a simulated link with 100 $\mu$s RTT. Wherever the transport protocol was TCP, a maximum congestion window size of 60 packets was used which corresponds to the default maximum TCP window size in most Microsoft Windows distributions. The switch and the edge router were in the same LAN with 2 $\mu$s RTT. The clients and the switch are in

another LAN with the same 2 $\mu$s RTT. This roughly corresponds to a LAN whose clients, switch and edge router are located in the same building.

The goal was to answer the following questions with an experimental evaluation of a simulation model of synchronized coalescing:

1. How does synchronized coalescing handle high traffic loads and sudden packet bursts?
2. How do the parameters of adaptive coalescing affect switch on time?
3. What are the performance trade-offs of adaptive coalescing?
4. How does adaptive coalescing affect real applications such as file download?

Three experiments were designed; they were 1) high load effect experiment, 2) threshold experiment, and 3) file download experiment. Each experiment was designed to answer the questions listed above.

The high load effect experiment was designed to study the effects of high loads and sudden bursts of packets on the performance of synchronized coalescing (Question 1). Packets flow from Client 1 to the switch over UDP. Packet interarrival time was exponentially distributed. At large time scales network traffic is likely self-similar [66], however a Poisson distribution remains a reasonable first-order approximation in cases which the traffic is highly aggregated and the traffic sample is taken in sub-second (small) time spans [57]. The size of the packets was fixed to 1500 bytes (maximum Ethernet MTU, compatible with bridged/switched 1 Gb/s systems). Simple synchronized coalescing was enabled on the switch with two duty cycles of 10% and 50%. $T_{off}$ was fixed to 100 ms. Since 100 ms is approximately the human response time, this is likely the upper bound of tolerable delay. A $T_{off}$ of 100 ms limits the relative time and consequent energy waste of transitions to less than 2% of the sleep time. The offered load was varied by manipulating the mean interarrival time of the packets. The response variables of interest were the switch on time and average per packet delay.

The threshold experiment was designed to study the effects of the threshold used in adaptive coalescing on packet delay and switch power use (Questions 2 and 3). A packet flow (with exponential interarrival times) over UDP from Server to Client 1 with a fixed packet size of 1500 bytes and variable offered load was modeled as in the previous experiment. $T_{off}$ was again fixed to 100 ms. Adaptive coalescing was enabled on the switch with 10% duty cycle. Two thresholds of 1000 and 5000 packets were used. The former (low threshold) corresponds to almost 10% of the link capacity (equal to the duty cycle). The latter

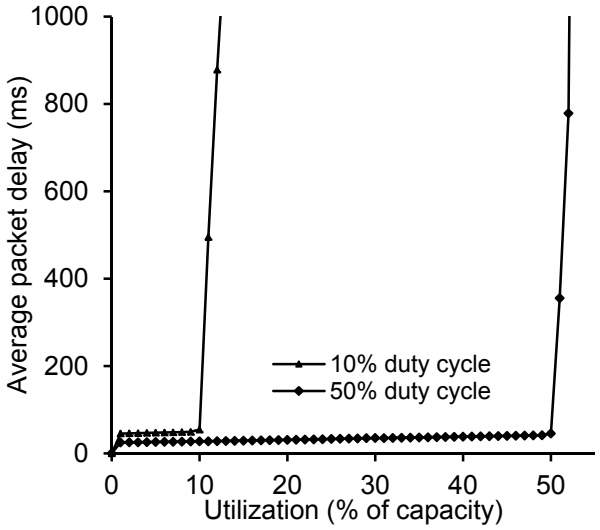



Figure 3.10: High load effect experiment results – packet delay

(high threshold) is almost half of the link capacity. The response variables of interest were the switch on time and average per packet delay.

The file download experiment was designed to study the effects of $T_{off}$ and the adaptive coalescing threshold on file download over TCP (Question 4). A 125 Megabyte file (corresponding to the file size of a small video clip) was downloaded by Client 1 from the Server using FTP. Adaptive coalescing was enabled on the switch with a 10% duty cycle. Three adaptive coalescing thresholds of 100, 500 and 1000 packets were used to show the effects of various thresholds on the response variables. $T_{off}$ was varied to cover the range from 0 to slightly more than the 100 ms. The response variables of interest were the file download time and the switch on time.

### 3.3.4 Results and Discussion

The experiment results were as follows. For the high load effect experiment, the average per packet delay as a function of offered load on the link is shown in Figure 3.10. It can be observed that when the load is more than the duty cycle, the average delay starts to rapidly increase due to instability of the buffer queue. Note that packets are delayed in the NICs connected to the switch, not inside the switch. This is due to halting of transmission from the NICs temporarily during $T_{off}$ periods while the packets are still being generated. The same phenomenon happens if there is a sudden burst in the traffic, even though the overall

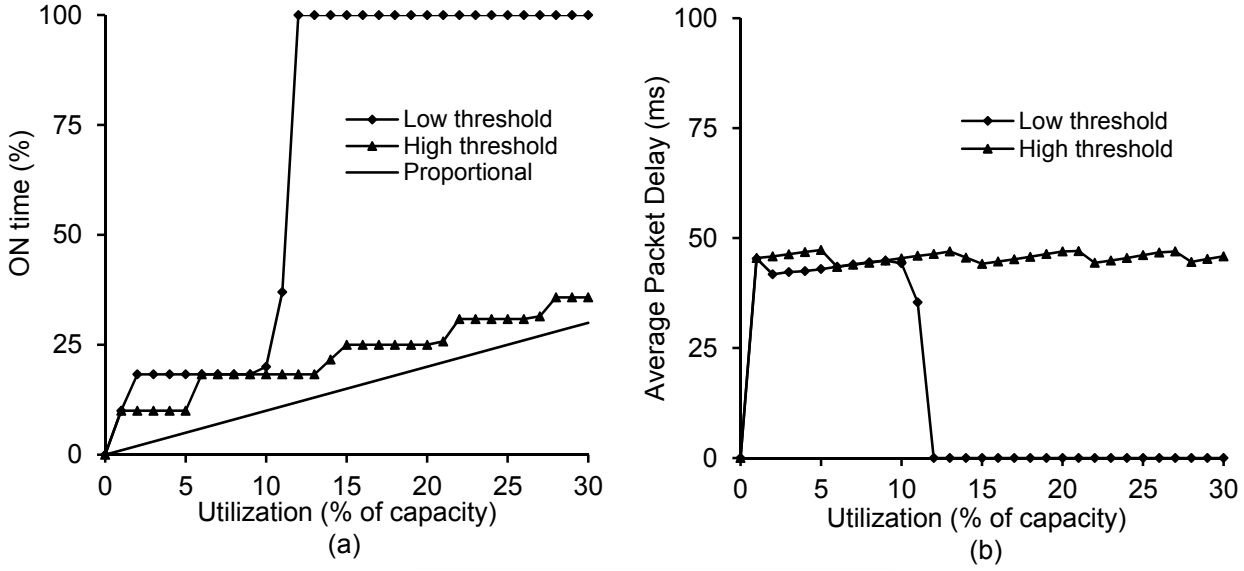



Figure 3.11: Threshold experiment results – (a) On time, and (b) packet delay

load is less than the duty cycle. The threshold for skipping Off periods in adaptive coalescing is an effective way to overcome this shortcoming. The switch on time is the same as the duty cycle regardless of load.

The results for the threshold experiment in Figure 3.11a show the percentage of time the switch spends in the On state as a function of load. Using the high threshold, the on time of the switch is less than 10% different than the load-proportional which is the ideal on time for the switch. However, the on time is more when the low threshold is used. Using the low threshold, when the load exceeds the duty cycle (10%) the on time ascends quickly to 100% and stays at this level as the load increases. This sudden increase is because the number of arriving packets during On periods exceeds the threshold. The "steps" seen in both traces are also because the number of arriving packets exceeds the threshold. At the points that the on time increases to the next step, the number of packets arriving during some of the On periods exceeds the threshold. After each increase to a given step, the on time stays roughly the same until the next increase to a new step. This is because while the number of arriving packets increases, the increase is not so high to cause any more On periods to have more arriving packets than the threshold. Therefore, the on time does not change in these intermediate loads. Although adaptive coalescing brings the on time close to load-proportional, it introduces a delay to some packets. The reason is that sometimes the switch is off and unable to service packets while packets keep arriving to the attached devices to the switch. These packets are queued and delayed. Figure 3.11b shows the average packet delay caused by adaptive coalescing as a function of offered load for the two thresholds. Steps are also seen in this figure as the load increases. Each

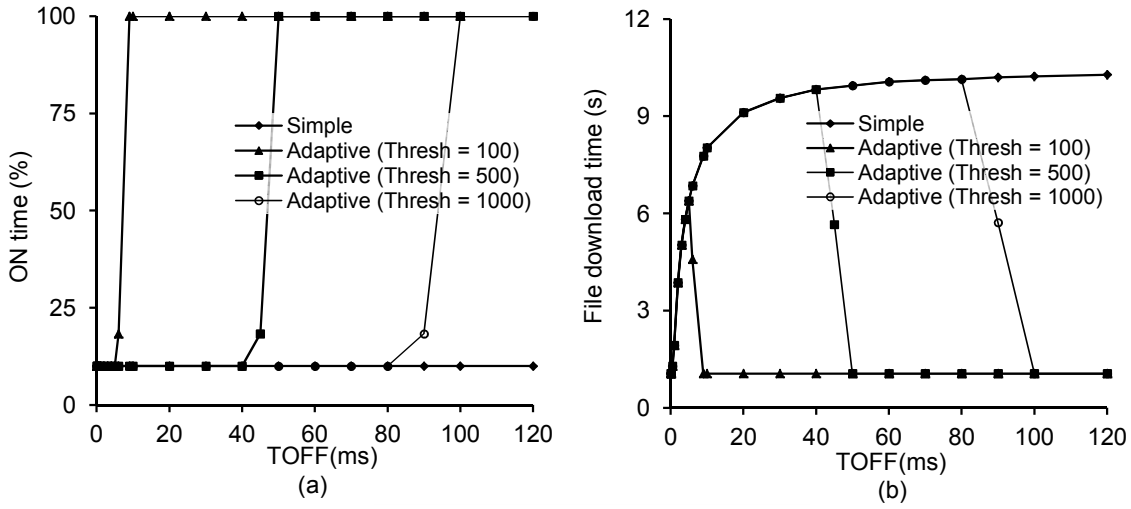



Figure 3.12: File download experiment results – (a) On time, and (b) download time

step corresponds to an on time step in Figure 3.11a. The reason is the same as what explained for on time. The average per packet delay is less than half of the human response time (50 ms) which has minor negative effect for non-delay-sensitive applications [93]. The sudden drop in the delay at 10% load by using the low threshold happens when the number of packets arriving to On periods exceeds the threshold constantly, which causes the switch to stay on and service the packets immediately.

If no coalescing were used, it would take about 1 s to download an entire 125-Megabyte file over the 1 Gb/s bandwidth (of the link between the edge router and the server through the Internet). Figure 3.12b shows the download time of the file for the file download experiment when using simple synchronized coalescing and adaptive coalescing with 10% duty cycle and three thresholds. Using simple synchronized coalescing with 10% duty cycle is as if the capacity of the switch is cut to 10%, which makes the capacity of the switch links equal to the bandwidth of the server through the Internet (1 Gb/s). This would suggest no increase in file download time. However, it is seen in Figure 3.12b that this is only the case when $T_{off}$ is very small (less than 3 $\mu$s). As $T_{off}$ increases, the download time increases exponentially until it becomes stable at about 10 s which is about one tenth of the capacity of the link between the edge router and the server. Instrumentation of the simulation showed that this counterintuitive effect is due to side effect with the TCP congestion control mechanism. If $T_{off}$ is longer than the time needed to transmit a full congestion window, the transmission pauses from the server until the window frees up in the next On period.

This effect can be controlled by reducing $T_{off}$. As explained in Section 3.3.1, however, $T_{off}$ cannot be set to less than 4 ms due to transition times. Too small $T_{off}$ times, although more than the minimum,

drastically increase the power waste due to relatively high transition times. Adjusting the threshold instead, can be used to control this effect temporarily while not sacrificing the beneficial length of $T_{off}$. As seen in Figure 3.12b, if the threshold is set to roughly the same number of packets that could be transmitted at full rate in a $T_{on}$, (for instance 100 packets for the duty cycle of 10%, $T_{off}$ of 10 ms and $T_{on}$ of 1.11 ms) the threshold is exceeded in case of a file download and makes the transfer time the same as it would be if no coalescing method were used. As a consequence, the switch would stay on for the entire download time as depicted in Figure 3.12a. Figure 3.12a shows the percentage of time the switch spends in the On state as a function of $T_{off}$ for the duration of the complete download of the file over FTP. The same three thresholds as in Figure 3.12b are shown. As expected, the switch stays on for 10% of the time when the threshold is larger than what transmitted in an Off period. When the threshold is exceeded, the switch stays on constantly.

From the results of the experiments, it seems that adaptive coalescing can save a significant amount of energy on a 10 Gb/s SOHO Ethernet switch with only small trade-off in reduced performance provided that the following conditions are met:

- The duty cycle ($C$) is set to a value which makes the switch spend most of the time in the Off state.
- The control variable $T_{off}$ is set to less than the human response time and reasonably more than the switch transition times.
- The threshold (*thresh*) is set to a value which detects file transfers and bursts well.

The values 10%, 100 ms, and 1000 packets for $C$, $T_{off}$, and *thresh* respectively, seem reasonable since it was shown that these values provide about 80% sleep time for the switch while introducing a small reduction in performance (less than 50 ms of average per-packet delay and almost no reduction in the case of a sudden traffic burst with the expense of less energy savings).

## 3.4 Chapter Summary

This chapter was comprised of three parts. The first part was dedicated to developing an energy-delay analytical model for a tandem queueing system with a coalescing queue and a single downstream queue. The model can help predict the performance-energy trade-offs of coalescing for links that support a sleep mode.

In the second part, the overhead of EEE transitions was studied which showed that the power consumption of EEE is much higher than proportional. Packet coalescing was studied as a means to decrease this inefficiency. The simulation model of EEE with packet coalescing was used to perform experiments to examine the energy efficiency of EEE with packet coalescing and study its drawbacks. The results showed that packet coalescing can bring the energy efficiency of EEE very close to proportional. Nevertheless, packet coalescing comes with two drawbacks. First, it increases the per-packet-delay. The results from simulation experiments showed that the average delay added to each packet is of the order of tens of microseconds at most. The significance of a few microseconds per packets delay is very likely to be low when compared to tens of milliseconds end-to-end delay of Internet connections. However, even this much increase in packet delay in a data center is considered significant, but the additional energy savings gained in a data server may be able to justify a reasonable delay for packets.

In the third part, it was shown how periodically powering down the components of a SOHO Ethernet switch after pausing the traffic from all the connected links in a synchronized manner can reduce the power use of the switch to almost proportional to the offered load. It was also shown that by using adaptive coalescing with a suitable threshold, energy-proportionality is achievable with an average delay of less than half that of the human response time (of 100 ms). Based on the presented results, it seems that adaptive coalescing is a viable method for significantly reducing the energy consumption of Ethernet switches without an excessive performance penalty.

## Chapter 4: An Energy-Efficient Hybrid Web Server Platform

As explained in Chapters 1 and 2, the electricity use of datacenters is increasing quickly. Previous work in reducing data center energy use has focused on powering up and down servers in clusters in response to demand ([19], for instance), server virtualization (Xen [9] and VMware [107], for instance), and using arrays of smaller processors to replace large processors (such as FAWN [6]). What has not been specifically addressed is the energy use of servers in small and medium enterprises (SMEs) where servers are often single computers with dedicated services and where clustering and virtualization methods used in data centers do not readily apply. It will be shown in this chapter that significant global energy savings can be achieved in this largely neglected area of SME web servers. The focus in this chapter is on SME web servers and a new hybrid web server architecture (based on a high-performance and high-power Master platform combined with a low-performance and low-power Assistant platform) to reduce energy use. While it is true that there is a trend towards cloud computing where enterprises move their server hosting to data centers (the "cloud"), it is likely that SMEs will continue to install and operate their own server computers for a variety of business and technical reasons. The ideas developed in this chapter can also apply to servers in data centers while this is not the main focus. IT energy consumption in SMEs comes from clients (primarily from desktop and laptop PCs), network infrastructure (including access points, switches, and routers), and servers. Controlling PCs to reduce energy use is a generally solved problem as seen by the many commercial offerings (such as the Verdiem Surveyor software [113]) and ongoing work in proxying-based solutions which were reviewed in Chapter 2. A hybrid system, as proposed in this chapter, can be seen as a natural evolution of proxying where the proxy platform now has the full capabilities, not just a subset of capabilities, of the higher-power Master host platform. The notion of a hybrid server was first proposed in [25], and was then expanded in [91]. A key challenge in a hybrid server is how to seamlessly switch between the platforms in such a manner as to ensure that there is no interval during which a client cannot access the server. This challenge is specifically addressed in this chapter. In this chapter, a new architecture for an energy-efficient SME hybrid

web server is proposed, which is named SME Web Energy Efficient Platform (SWEEP). Within the scope of this new architecture, the contributions include:

- A characterization of an SME web server log.
- A new method (macSwitch) to seamlessly switch between the two platforms by blocking ARPs and using a command/status handshake between the two platforms.
- An experimental evaluation of a prototype SME hybrid web server that uses seamless switching between platforms and prediction for when to switch.

## 4.1 Characterization of a Representative SME's Server Log

As explained before, it is a general understanding that ICT systems including servers, clients, and networks in general operate at very low utilization levels most of the time [90]. This is also true for servers in data centers [10]. To gain an understanding of the fraction of time an SME server operates at a low utilization (that is, the fraction of time that the request rate is low), an analysis on a one-month log of HTTP requests collected from the KETI main web server (found at http://www.keti.re.kr). KETI (Korea Electronics Technology Institute) is a government research laboratory in Korea with two major tasks; "to help small and medium-sized businesses and ventures secure innovative technology, and to develop advanced technologies and create new venture projects" [61]. KETI has about 450 employees and is, itself, an SME. The KETI main web server is hosted on a PC running Microsoft IIS and serves both static and active (ASP) pages. A one-month log, collected in November 2010, captured and time-stamped all HTTP requests to the KETI main web server. Figure 4.1 shows a snippet of the log stored in an Excel spreadsheet. The log contains 3,211,058 HTTP request records.

Table 4.1 shows the summary statistics for the type of request, interarrival time between requests, and the request rate per minute for successive one minute intervals for the entire month. It can be seen that 1) there is great variation in request rate (but, for over half the time the request rate is very low – 10 requests per minute or less), and 2) active page requests are less than 5% of all requests. Figure 4.2 shows the daily variation in request rate for (a) the first week of November (which started on a Monday), (b) a single weekday (the first Wednesday), and (c) a single weekend day (the first Sunday). A diurnal cycle for demand and also lower demand during the weekend can clearly be seen.

| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| 1 | #Software: Microsoft Internet Information Services 5.0 | | | | | | | |
| 2 | #Version: 1.0 | | | | | | | |
| 3 | #Date: | 11/1/2010 | 0:00:52 | | | | | |
| 4 | date | time | c-ip | s-ip | s-port | cs-method | cs-uri-stem | sc-status |
| 92 | 11/1/2010 | 0:02:31 | 211.214.88.50 | 203.253.128.3 | 80 | GET | /home/js/main_navi.js | 200 |
| 93 | 11/1/2010 | 0:02:31 | 211.214.88.50 | 203.253.128.3 | 80 | GET | /home/image/t_logo.gif | 200 |
| 94 | 11/1/2010 | 0:02:31 | 211.214.88.50 | 203.253.128.3 | 80 | GET | /home/image/menu/gnb_menu_1.png | 200 |
| 95 | 11/1/2010 | 0:02:31 | 211.214.88.50 | 203.253.128.3 | 80 | GET | /home/image/menu/gnb_menu_2.png | 200 |
| 96 | 11/1/2010 | 0:02:31 | 211.214.88.50 | 203.253.128.3 | 80 | GET | /home/image/menu/gnb_menu_3.png | 200 |
| 97 | 11/1/2010 | 0:02:31 | 211.214.88.50 | 203.253.128.3 | 80 | GET | /home/image/menu/gnb_menu_4.png | 200 |
| 98 | 11/1/2010 | 0:02:31 | 211.214.88.50 | 203.253.128.3 | 80 | GET | /home/image/menu/gnb_menu_5.png | 200 |
| 99 | 11/1/2010 | 0:02:31 | 211.214.88.50 | 203.253.128.3 | 80 | GET | /home/image/visual/all_bg.gif | 200 |
| 100 | 11/1/2010 | 0:02:31 | 211.214.88.50 | 203.253.128.3 | 80 | GET | /home/image/menu/sub_menubg01.gif | 200 |
| 101 | 11/1/2010 | 0:02:31 | 211.214.88.50 | 203.253.128.3 | 80 | GET | /home/image/menu/introTitle.gif | 200 |
| 102 | 11/1/2010 | 0:02:31 | 211.214.88.50 | 203.253.128.3 | 80 | GET | /home/image/title01_02.gif | 200 |

Figure 4.1: KETI server log showing time-stamp, IPs, port, HTTP method, URI, and response

The assumption here is that a low-power platform can serve a given request rate lower than that of the high-power platform. Later in this chapter, the request rate that a low-power platform can serve is explored and "low request rate" will more formally be defined. Table 4.2 shows the percentage of one minute intervals for the full month, the first week, a weekday, and a weekend day that are below a given threshold request rate. The notion is that the high-power platform sleeps during these lower-than-threshold periods and the low-power platform handles the requests. The percentage of time the high-power platform can sleep increases as the value of the threshold rate increases. Further in this chapter, the question of what

Table 4.1: Summary statistics for the KETI server log

| Parameter | Measure |
|---|---|
| Mean time between all requests | 0.81 s |
| Mean time between active page requests | 16.81 s |
| Percent of requests that are active pages | 4.8% |
| Mean request rate | 74 requests/min |
| 99% request rate | 524 requests/min |
| 95% request rate | 321 requests/min |
| 50% (median) request rate | 10 requests/min |
| 5% request rate | 1 request/min |
| 1% request rate | 1 request/min |

Table 4.2: Percentage of one-minute intervals with request rate to KETI server below threshold

| | threshold rate (requests/minute) | | | | | | |
|---|---|---|---|---|---|---|---|
| | 10 | 25 | 50 | 100 | 150 | 200 | 300 |
| Full month | 50% | 56% | 61% | 74% | 82% | 87% | 94% |
| First week | 54% | 59% | 63% | 75% | 82% | 87% | 94% |
| Week day | 44% | 48% | 53% | 66% | 76% | 82% | 92% |
| Weekend day | 70% | 76% | 80% | 91% | 95% | 97% | 99% |

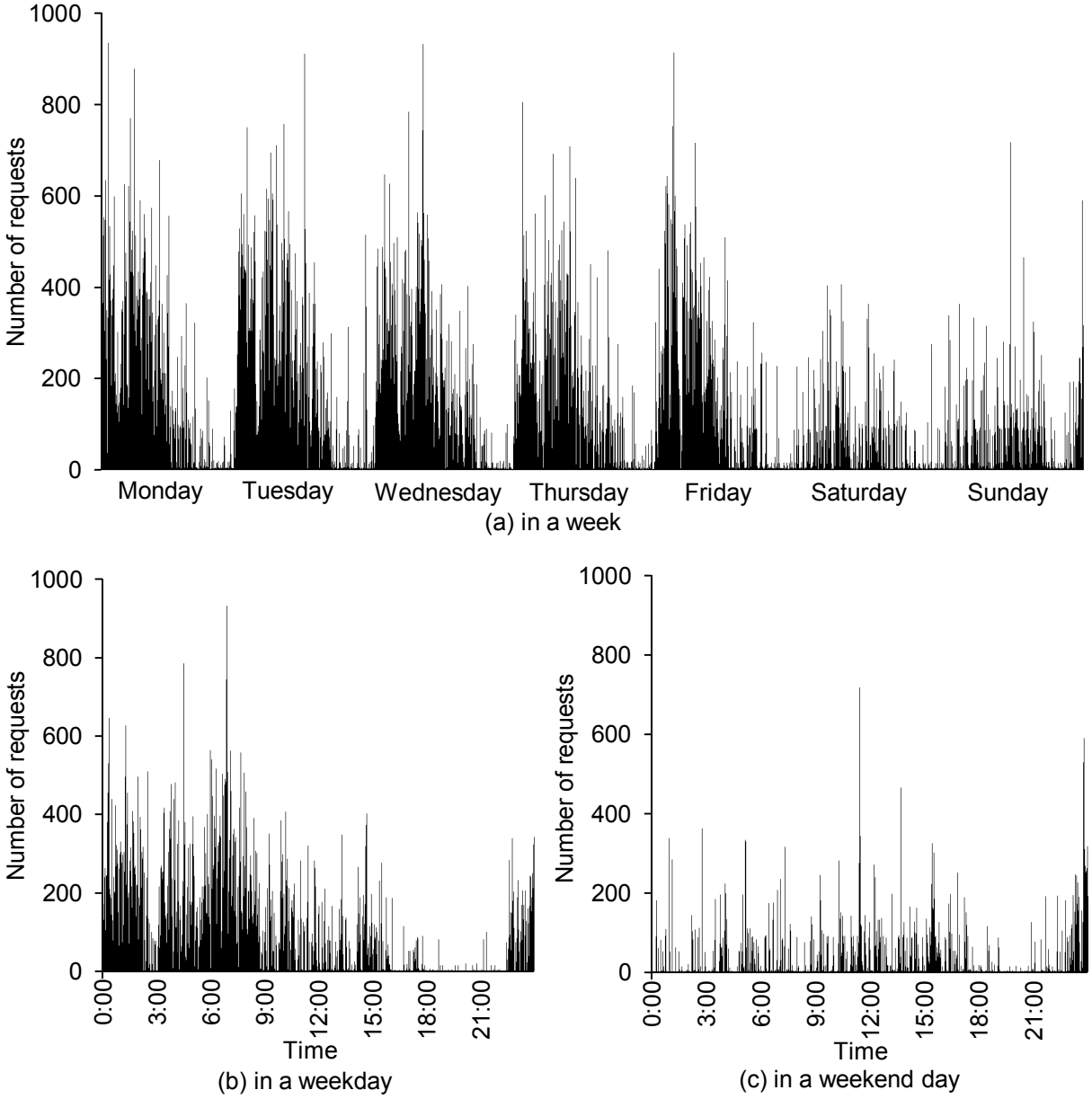



Figure 4.2: Variation in request rate to KETI main server

request rate a low-power platform can support will be answered (which is remarkably high – on the order of hundreds of requests per minute with acceptable response time for static pages). From Table 4.2 it can be seen that for over 60% of the total time for the full month, the request rate is less than 50 requests per minute. This suggests that a low-power platform capable of handling at least 50 requests per minutes could "cover" for the high-power platform for a large percentage of time.

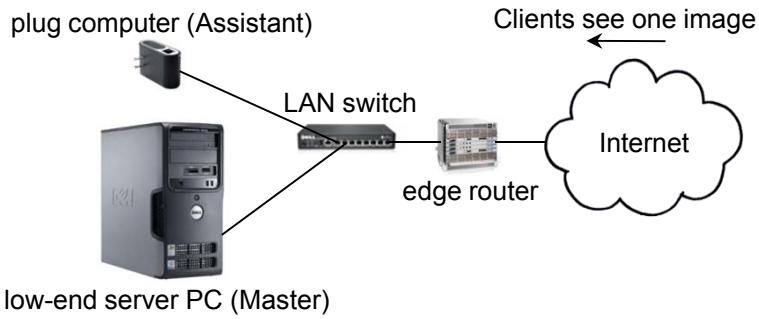



Figure 4.3: System configuration of SWEEP

## 4.2 The SME Web Energy Efficient Platform (SWEEP) Architecture

The basic idea in SME Web Energy Efficient Platform (SWEEP) is to have two heterogeneous platforms that support the same web server software (for example, Apache). The platforms are mirrored for content – both platforms are equally capable in what requests they can serve. However, there is a (potentially significant) difference in the delay with which they can serve client requests at different rates of incoming requests. The high-power platform is the Master and the low-power platform is the Assistant. Figure 4.3 shows the system configuration. The notion is that the low-power Assistant can be fully powered-on all the time (but can only serve the system workload when it is sufficiently low for the Assistant to handle) and the Master be powered-on only during rare periods of high request rate (the rest of the time the Master can sleep, and thus the power consumed for most of the time would be only that of the low-power Assistant). Various sleep modes for the Master can be explored. To maintain generality, the sleep mode is not specifically stated and it is noted that any sleep mode (stand-by, suspend, or hibernate) can be used in the method. Note, however, that the "suspend" state is likely the most suitable sleep mode to use for the Master. The reason is that the resulting power consumption in this mode is close to zero while the transition times to and from sleep are substantially lower than the length of intervals used for load sampling (4-5 seconds vs. 1 minute intervals).

The use of emerging extremely small plug computers for the Assistant platform is explored here. For the prototype system used in this chapter, a Sheeva plug computer manufactured by Marvel was chosen that costs about $100 for single quantities (in late 2011) and contained a 1.2 GHz ARM processor, 512 MB SDRAM, 512 MB Flash, and USB and 1 Gb/s Ethernet connectivity [94]. The plug computer ran Ubuntu 9.04. The power consumption was a constant 7 W independent of CPU load. An 8-GByte SDHC flash memory card

was used for storage (at an approximate cost of $10 in late 2011). For the Master, a Dell Optiplex GX620 PC with an Intel Pentium 4 2.80 GHz CPU running Ubuntu 11.04 was used. This PC (with the monitor off) consumed between about 65 W and 95 W depending on the CPU utilization.

### 4.2.1 The httpRedirect Method

A major challenge in the hybrid system is to design a method for switching between the Assistant and the Master as a function of request rate. When the request rate is low, the Assistant serves all requests while the Master sleeps. When the request rate is high, the Master is awake and serves all requests. That is, the requests are now "switched" to the Master. Switching between the platforms should be seamless such that no requests (whether in service or incoming from a client) are lost. The switching should be accomplished without the need for a separate load balancer since the need for a load balancer increases the cost and the complexity of design and operation of a hybrid server.

The Unix rsync utility can be used to synchronize, or mirror, the files in the two platforms. Synchronizing can occur during predetermined time periods during the day or during the switchover between the two platforms. In web servers that contain active pages, two scenarios can occur. First, the database resides on a separate machine, in which case both the Master and the Assistant can connect to the database separately. Second, the database is on the same machine where the web server resides. In this case, the database needs to be replicated on the Assistant and a syncing mechanism needs to be implemented to synchronize the two databases when changes to the data are made. A transactional replication between the Master and the Assistant can be used for this purpose. In the implementation used in this chapter, the database resides on a separate always-on machine, thus does not need syncing. Studying the second case and the implications of synchronizing the databases on the Master and the Assistant are future work. In the httpRedirect method, the Assistant has the IP address that is assigned to the website URL and the Master has an IP address known only locally. The Assistant and Master have different MAC addresses in all cases. The switching between the Assistant and the Master is accomplished with HTTP redirection in a manner first explored in the mid-1990s with SWEB [7]. The httpRedirect method is not new, it is directly based on HTTP redirection in SWEB and serves as a baseline in this chapter. When the request rate is above a pre-determined threshold value, the Assistant responds to all requests with an HTTP 307 response with the IP address of the Master to cause the

originating clients to resend their requests to the Master. The Assistant is fully powered-on at all times and is responsible for making the decision to switch between the platforms.

Table 4.3 shows the variables, one timer, and messages used for the httpRedirect method (and also used for the macSwitch method described next). There are two types of messages – command messages from the Assistant to the Master, and status messages from the Master to the Assistant. The command messages are used to put the Master to sleep, and make it start serving. A standard Magic Packet [69], sent by the Assistant, is used to wake up the Master from a sleep state. The status messages convey the state of the Master to the Assistant. Exchange of command and status messages can be achieved using UDP or TCP. Figures 4.4 and 4.6 show the design of the Assistant and the Master. For the httpRedirect method, the Assistant FSM is depicted in Figure 4.4. The FSM has 4 states:

- Serving: The Assistant is serving the incoming requests.

Table 4.3: Variables, timer, and messages used in httpRedirect and macSwitch methods

| | |
|---|---|
| **Variables used in the Master FSM** | |
| none | |
| **Variables used in the Assistant FSM** | |
| *prediction* | platform to use in next interval |
| **Variables used in sample timer FSM in the Assistant** | |
| *intervalTime* | sample time for prediction interval |
| **Variables used for rate prediction** | |
| *threshold* | Maximum request rate for the Assistant |
| *nextRate* | Predicted request rate next interval |
| *sampleRate* | Sample for EWMA |
| $\alpha$ | Weight for EWMA |
| **Timer in the Assistant** | |
| *sampleTimer* | Timer for sampling intervals |
| **Command messages from the Assistant to the Master** | |
| *GoToSleep* | Master to go to sleep |
| *StartServing* | Master to start serving requests |
| **Status messages from the Master to the Assistant** | |
| *OnServing* | Master is on and is serving |
| *OnNotServing* | Master is on but is not serving |
| *Sleeping* | Master is sleeping |

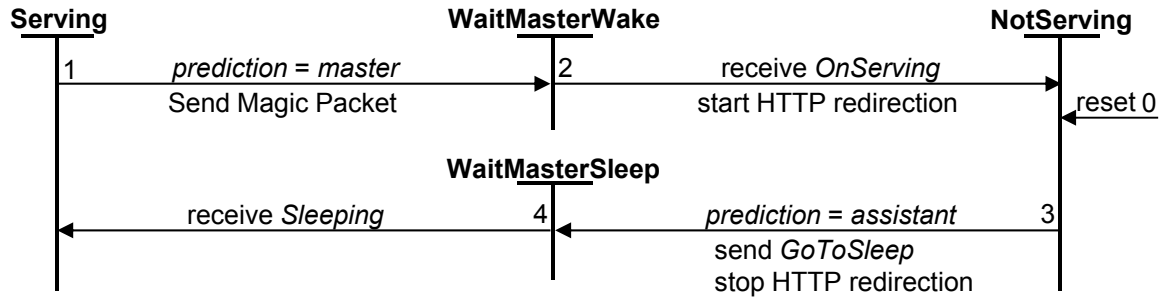



Figure 4.4: FSM for the Assistant in httpRedirect method

- WaitMasterWake: The Assistant is waiting for the Master to wake up. The Assistant is serving the requests in this state.
- WaitMasterSleep: The Assistant is waiting for the Master to finish serving its pending requests and go to sleep. The Assistant is serving the requests in this state.
- NotServing: The Assistant is not serving the requests; it is redirecting all the requests to the Master and sampling the load, which will be explained later in this section.

A detailed explanation for each of the FSM transitions follows:

- Transition 0: Upon start or reset, the system starts with the Assistant FSM in the NotServing state.
- Transition 1: This transition is triggered by the event of the prediction variable for the next sampling interval being set to *master*, in which case the Assistant wakes up the Master by sending a Magic Packet to it, and transitions to the WaitMasterWake state.
- Transition 2: Upon receiving an OnServing status message from the Master which indicates that the Master is woken up and ready to serve requests, the Assistant redirects the incoming requests to the Master.
- Transition 3: This transition is triggered by the event of the prediction variable for the next sampling interval being set to *assistant*, in which case the Assistant sends a GoToSleep command message to the Master, stops redirecting incoming requests to the Master, and transitions to the WaitMasterSleep state.
- Transition 4: Upon receiving a Sleeping status message from the Master which indicates that the Master is now sleeping, the Assistant transitions to the Serving state.

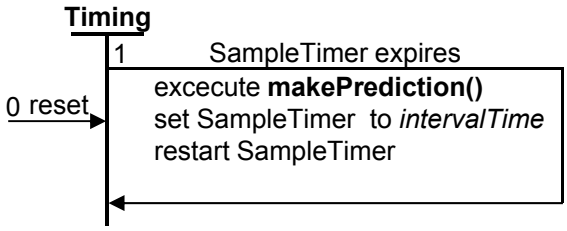



Figure 4.5: FSM for sampling in the Assistant

Figure 4.5 shows the design of the sampling capability within the Assistant. The variable *intervalTime* determines the minimum time period for which the Assistant or the Master platform must keep serving (and awake for the case of the Master). A single timer, SampleTimer, is periodically reset and restarted, as shown in Figure 4.5. The prediction of future request rates, implemented in the makePrediction() function, is described in Section 4.3. The Master FSM in Figure 4.6 has three states:

- OnServing: The Master is on and serving the incoming requests.
- WaitQEmpty: The Master is waiting for the request queue to become empty before sleeping.
- Sleeping: The Master is sleeping.

A detailed explanation for each of the FSM transitions follows:

- Transition 0: Upon start or reset, the system starts with the Master FSM in the OnServing state.
- Transition 1: Upon receiving a GotoSleep command message from the Assistant which indicates that the incoming request rate is low enough for the Assistant to be able to handle, the Master transitions to the WaitQEmpty state.
- Transition 2: When the request queue becomes empty, the Master sleeps by transitioning to the Sleeping state after sending a Sleeping status message to the Assistant.
- Transition 3: A Magic Packet sent from the Assistant to the Master triggers this transition and causes the Master to wake up. The Master sends an OnServing status message to the Assistant upon wakeup.

As can be seen in the FSMs, the httpRedirect method uses the GoToSleep, OnServing, and Sleeping messages to handshake between the Assistant and Master. Note that both FSMs should start, or reset, at the same time.

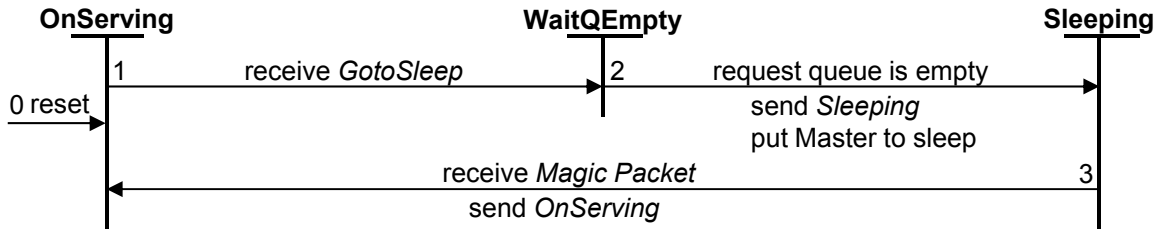



Figure 4.6: FSM for the Master in httpRedirect method

The httpRedirect method requires the Assistant to handle all incoming requests in all cases and requires a double round-trip delay during high request rate conditions (the HTTP redirect causes the second round trip delay). The following macSwitch method fixes this deficiency.

### 4.2.2 The macSwitch Method

In the new macSwitch method, the single IP address of the server is shared between the Assistant and the Master. Both the Master and the Assistant also maintain a second IP address that is only known locally. The second (local only) IP addresses are mainly used to exchange command and status messages between the two platforms. When the Assistant is serving requests the IP-MAC address association in the ARP cache of the edge router (shown in Figure 4.3) associates the MAC address of the Assistant with the IP address. Conversely, when the Master is serving requests the ARP cache associates the MAC address of the Master with the IP address. Switching the IP-MAC address association is accomplished by using a Gratuitous ARP (GARP) message. This method of changing IP-MAC address association is well known and is used in failover methods [43]. The challenge is to make this switch such that no requests are lost.

Figures 4.7 and 4.8 show the FSMs for the macSwitch method in the Assistant and the Master, respectively. The Assistant FSM adds one new state:

- WaitQEmpty: The Assistant is waiting for its request queue to become empty.

The Master FSM also adds one new state:

- OnNotServing: The Master is woken up and ready to serve requests. However, it will not start serving until it receives a StartServing command message from the Assistant which indicates that the Assistant's request queue is empty and switching the platforms will not cause request loss.

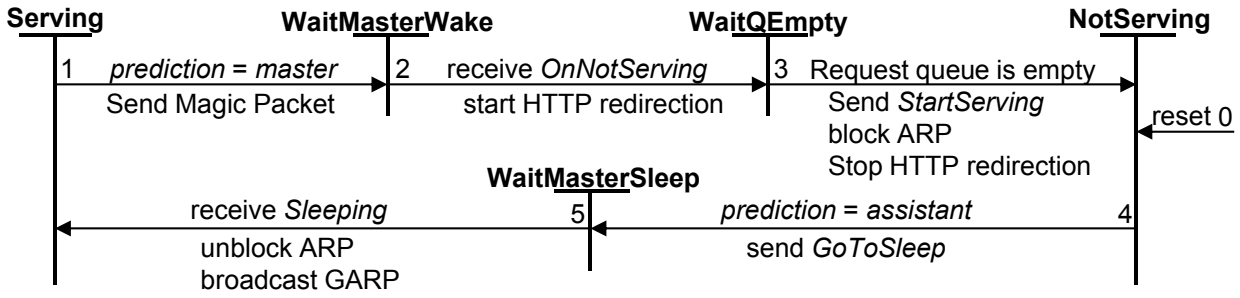



Figure 4.7: FSM for the Assistant in macSwitch method

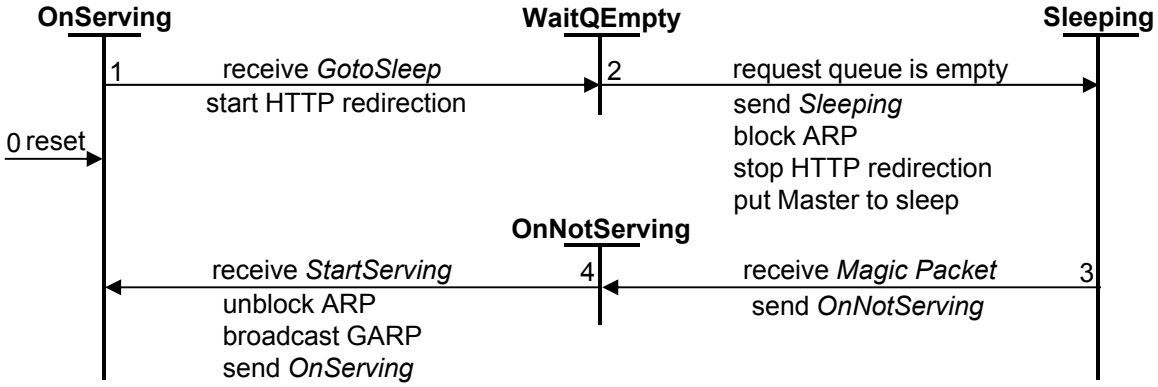



Figure 4.8: FSM for the Master in macSwitch method

Exit from the WaitQEmpty state in transition 3 of Figure 4.7 occurs only when the request queue of the Assistant is empty (after HTTP redirection has been initiated in transition 2 to redirect requests to the local IP address of the Master). Temporary HTTP redirection prior to switching the platforms with ARP blocking and sending Gratuitous ARP is needed temporarily to ensure that the request queue of the Assistant (or the Master) will indeed become empty at some point of time. This, however, causes a vulnerability which will be described later. When the request queue is empty, the StartServing command message is sent to the Master and the Master then exits the OnNotServing state in transition 4 of Figure 4.8 and broadcasts a GARP packet to associate the IP-MAC address with itself. At this point, all client requests will be forwarded by the end router to the Master. The receiving of ARP packets is blocked in the platform currently not serving requests to prevent a duplicate address error condition.

The macSwitch method has three known vulnerabilities:

- The first vulnerability occurs when the request rate is so high that the request queue is never cleared in the WaitQEmpty state (Figure 4.7). In this case, the switch to the Master will never occur and

requests (during the high request rate period) will have a very high response time and/or be dropped by the Assistant due to overflow.

- The second vulnerability occurs when the Assistant and the Master are in states WaitMasterSleep and WaitQEmpty, respectively (in this case, new requests are being redirected to the Assistant). Suppose that there is a request in Master's queue which requires service time greater than one sampling interval to finish. If makePrediction() predicts that the request rate in the next interval will be above the threshold, the GoToSleep message should be withdrawn and the Master and the Assistant should go back to the OnServing and NotServing states, respectively. However, there is currently no means of withdrawing or canceling a previous GoToSleep message. As a result, the Master can only resume serving new requests after it is powered down and woken up (back to the OnServing state through the Sleeping and the OnNotServing states). This vulnerability causes high response time for the redirected requests and some requests can be dropped due to overflow. To solve this vulnerability, a new message, Abort, can be used to cancel a previous GotoSleep message to the Master and abort the power-down process. Both of the above vulnerabilities are explored in the evaluation of the SWEEP prototype described later in this chapter.

- The third vulnerability is due to the limited CPU capacity of the Assistant. It is possible that a sudden, very high request rate can delay the sampling process in Figure 4.5. To minimize the effect of this condition, a second sampling capability (as an additional transition to the FSM of Figure 4.5) can be added to monitor the CPU load (where load is the number of runnable processes on the CPU) of the Assistant and use this as a trigger for switching to the Master earlier than when SampleTimer expires. This sampling capability can be included in the design of the macSwitch method by modifying the triggering event of transition 1 in the FSM of the Assistant (Figure 4.7) to $(prediction = master) \vee (cpuLoad \geq maxCPULoad)$ where *cpuLoad* is the current CPU load for the Assistant and *maxCPULoad* is a predefined threshold which serves as an early notification of a possible CPU overload condition. The variable *cpuLoad* is updated with the current load of the CPU. Another timer with shorter intervals than *intervalTime* triggers the reading of the CPU load (the CPU load can be read using w or top Linux tools) and updates *cpuLoad* with the current average load.

```
makePrediction()
   nextRate ← HTTP request rate in next interval
   if (nextRate < threshold)
      prediction ← assistant
   if (nextRate ≥ threshold)
      prediction ← master
   return prediction
```

Figure 4.9: Oracle prediction

```
makePrediction()
   sampleRate ← HTTP request rate in last interval
   nextRate ← α · lastRate + (1 – α) · sampleRate
   lastRate ← nextRate
   if (nextRate < threshold)
      prediction ← assistant
   if (nextRate ≥ threshold)
      prediction ← master
   return prediction
```

Figure 4.10: Exponentially Weighted Moving Average prediction

## 4.3 Prediction of Future Request Rates

The Assistant should switch the serving platform to the Master in periods of time wheres the request rate is higher than it can handle. A threshold request rate is defined below which it is deemed that the Assistant can serve requests with a satisfactory response time. Prediction is used to determine if in the next sample interval the request rate is expected to be above the threshold. The prediction of *assistant* or *master* is made in the makePrediction() function, two versions of which are shown in Figures 4.9 and 4.10. Two prediction methods were considered – the ideal case of an oracle that can "see" into the future, and prediction based on Exponentially Weighted Moving Average (EWMA). Figure 4.9 shows the oracle where the request rate of the next sampling interval is known (it can be read directly from the KETI server log in the experimental evaluation of SWEEP). Figure 4.10 shows the use of EWMA to predict the future request rate as a function of previously known request rates including the current sample. The key experimental factors, or "tuning parameters", are sampleInterval and $\alpha$ (for EWMA).

## 4.4 Implementation and Performance Evaluation of a Prototype SWEEP System

This section describes the implementation of a prototype SWEEP system and its evaluation. The capabilities of the Master and Assistant platforms were evaluated using the Apache ab benchmark [1]. Prediction methods were evaluated using a simulation model. Finally, the SWEEP prototype system was experimentally evaluated by a workload derived directly from the KETI server log characterized in Section 4.1.

### 4.4.1 Implementation of a Prototype SWEEP System

The implementation of the Master and Assistant processes that comprise the SWEEP system was based on multithreaded C programs which readily available Linux and Apache utilities. The following were used:

- arptables to block ARP packets,
- ether-wake to send a Magic Packet to the Master,
- arpsend to send Gratuitous ARP (GARP) packets,
- mod-status module status page for calculating the request rate for the current interval
- mod-rewrite module rules in Apache for the redirection of HTTP requests, and
- scoreboard string reported by mod-status module of Apache to keep track of the number of in service requests and to determine when the Master request queue becomes empty.

The web server used in the prototype was Apache 2.2 (running in both the Master and Assistant platforms) with mod-rewrite and mod-status activated.

The Assistant process consisted of two threads; one thread would listen to status messages from the Master and update an internal variable which maintained the current state of the Master. The second thread would read the request rate of the current serving platform (by subtracting the current total number of page requests reported by mod-status module of Apache from that of the previous interval) and execute makePrediction() at sampling intervals. For the httpRedirect method, if the prediction was to switch to the Master, the Assistant would wake up the Master (by sending a Magic Packet using ether-wake) and redirect

new requests to it (this wass done by modifying Apache mod-rewrite module rules). For the macSwitch method, the Assistant would wake up the Master, wait for the request queue to become empty by monitoring the Apache scoreboard while redirecting new requests to the Master by modifying Apache mod-rewrite module rules, and finally block ARP packets by modifying arptables rules. For the httpRedirect method, if the prediction was *assistant*, the Assistant would stop redirecting requests to the Master, and send it a GoToSleep command message which would cause the Master to power down and enter an energy-saving sleep state. For the macSwitch method, the Assistant would send a GoToSleep message to the Master, wait for it to power down, unblock ARP packets (again, by modifying arptables rules), and finally broadcast a GARP using arpsend with the MAC address of the Assistant.

The Master process would listen to command messages from the Assistant and act accordingly. For the httpRedirect method, upon receiving a GoToSleep message, the process would wait for the request queue to become empty and power down the Master. Upon receiving a Magic Packet, the Master would wake up and resume serving new requests. For the macSwitch method, upon receiving a GoToSleep message, the Master would wait for the request queue to empty by monitoring Apache scoreboard while redirecting new requests to the Assistant by modifying Apache mod-rewrite module rules, block ARP packets by modifying arptables rules, and power down the Master using the pm-suspend system call. Upon receiving a StartServing command message after being woken up by a Magic Packet, the Master would unblock ARP packets (again, by modifying arptables rules) and broadcast a GARP with the MAC address of the Master using arpsend.

### 4.4.2 Benchmarking of the Prototype SWEEP Using Apache ab

Using the Apache Benchmark (ab) the response time of the Master and Assistant platforms for static and active pages were measured. The Master-Assistant switching method was also evaluated. The purpose of this evaluation was to determine the capabilities of the Master and Assistant platforms and to determine the effect of switching on request response time. ab benchmark works as follows. A single client PC sends requests to a server (the Master or Assistant platform running Apache) as fast as the server can respond to the requests. ab benchmark allows a single client to emulate multiple clients by running concurrent processes – one process for each emulated client thus allowing the server to service requests in parallel. The total number of requests and the request URL can be set in the client. The total number of requests sent to the

platform was chosen to be large enough to take a few minutes to complete. The static page was a 100 KB text file (95% of files on KETI server are 100 KB in size or less), and the active page was a PHP page with 2 database lookups, and 4 images (this represents the active pages on KETI main web server – the majority of the active pages on KETI main server have less than 5 database lookups and less than 10 images). The switching between the Master and Assistant platforms was hardwired for 30-second intervals (that is, every 30 seconds the Master would switch the request stream to the Assistant and vice versa).

Tables 4.4 and 4.5 show the benchmarking results for static and active pages respectively. The first column, labeled as C, is the number of concurrent processes sending requests. The results are reported for up to 100 concurrent connections to the platform. For each run, the mean, median and the 99th percentile is reported. For delivering static pages, the results show that the Master and Assistant (and consequently also SWEEP) have about the same response time for low to moderate loads (up to about 10 concurrent connections). For higher loads, the Master platform has a slightly lower response time than the Assistant platform. As expected, the performance of the hybrid system falls almost in the middle of that of the Master and Assistant. The Assistant refused new connections when the load was beyond 350 concurrent connections, whereas the Master was able to handle higher loads. The response time for delivering active pages, however, is significantly different between the two platforms. The Assistant shows 3 to 6 times longer

Table 4.4: Apache ab benchmark results for a static page

| | Assistant (ms) | | | Master (ms) | | | Hybrid (ms) | | |
|---|---|---|---|---|---|---|---|---|---|
| *C* | Mean | Med | 99% | Mean | Med | 99% | Mean | Med | 99% |
| 1 | 13 | 12 | 23 | 12 | 11 | 17 | 13 | 12 | 21 |
| 2 | 19 | 18 | 28 | 19 | 18 | 29 | 19 | 18 | 30 |
| 3 | 28 | 26 | 41 | 27 | 26 | 43 | 27 | 26 | 44 |
| 5 | 45 | 43 | 72 | 45 | 43 | 70 | 47 | 43 | 73 |
| 10 | 90 | 87 | 131 | 90 | 87 | 126 | 93 | 87 | 128 |
| 100 | 932 | 922 | 1114 | 894 | 880 | 1128 | 910 | 924 | 1121 |

Table 4.5: Apache ab benchmark results for an active page

| | Assistant (ms) | | | Master (ms) | | | Hybrid (ms) | | |
|---|---|---|---|---|---|---|---|---|---|
| *C* | Mean | Med | 99% | Mean | Med | 99% | Mean | Med | 99% |
| 1 | 140 | 140 | 160 | 41 | 40 | 50 | 67 | 42 | 143 |
| 2 | 271 | 270 | 290 | 51 | 59 | 70 | 104 | 54 | 287 |
| 3 | 402 | 400 | 440 | 67 | 60 | 100 | 122 | 69 | 423 |
| 5 | 665 | 660 | 740 | 110 | 110 | 180 | 214 | 116 | 705 |
| 10 | 1319 | 1320 | 1500 | 210 | 200 | 400 | 384 | 219 | 1204 |
| 100 | 13344 | 13140 | 22170 | 2184 | 2150 | 3850 | 3163 | 2136 | 18295 |

response time. Again, the response time of the hybrid system falls between the two platforms, but it is closer to the response time of the Master. The reason is that for a fixed number of requests and fixed intervals of service from each platform, a much higher number of requests are served by the Master than served by the Assistant. This makes the mean and median closer to that of the Master. Despite the difference in response time, the point at which both platforms close new connections is almost the same at about 250 concurrent requests. These results suggest that active pages benefit from a more powerful platform whereas static pages seem to be delivered equally well from the Master and the Assistant.

### 4.4.3 Evaluation of Prediction Methods

The prediction method, implemented in the makePrediction() function, predicts the request rate for the next sampling interval. This predicted request rate is then used to determine the platform to be used for the next interval (which may be the current platform or a switch to the other platform). The predicted request rate is compared to a set *threshold* value to make a decision on which platform should be used. A prediction method can have two types of mispredictions:

- Underestimation, in which the method predicts a request rate lower than the *threshold* for the next interval, but a higher request rate occurs instead. As a result of switching to (or staying in) the Assistant based on this prediction, the Assistant will be overloaded and the response time of serving requests will increase. Increased response time results from this type of misprediction.

- Overestimation, in which the method predicts a rate higher than the *threshold* for the next interval, but a lower request rate occurs instead. As a result of switching to (or staying in) the Master based on this prediction, the Master will be powered-on and serving when it could otherwise have been sleeping. A waste of energy results from this type of misprediction.

The ideal prediction method would be an oracle that can "see" into the future. For the same workload, no prediction method results in a longer sleep time and a shorter response time at the same time than the oracle. For a trace-based evaluation where the KETI server log is used to create a workload, it is possible to see into the future. For a feasible prediction method, Exponentially Weighted Moving Average (EMWA) was implemented. The oracle and EMWA prediction methods as implemented in makePrediction() are shown in Figures 4.9 and 4.10, respectively.

```
SleepSimulation()
   sleepCount ← overCount ← underCount ← 0
   nextSample ← first sample from processed trace file
   for (all entries in processed trace file) do
      prediction ← makePrediction()
      nextSample ← next sample from trace file
      if (prediction = assistant) then
         sleepCount ← sleepCount + 1
         if (nextSample > maxRequestRate) then
            underCount ← underCount + 1
      if (prediction = master) then
         if (nextSample ≤ threshold) then
            overCount ← overCount + 1
   return (sleepCount, underCount, overCount)
```

Figure 4.11: Simulation model to evaluate prediction

Table 4.6 shows the variables used in the trace-driven simulation, and Figure 4.11 outlines the simulation procedure. The variables *underCount* and *overCount* are used to determine the percentage of energy waste due to overestimation and the time where the Assistant is performance limited due to underestimation, respectively. The variable *sleepCount* counts the number of intervals the Master is put to sleep. In each iteration of the loop, the request rate of the next interval is read from the processed KETI server log and assigned to *nextSample*. The processed KETI server log was the KETI server log from the KETI main server reduced to the number of requests per one minute intervals. The HTTP request rate in the first line is the request rate of the current interval that is read from the processed KETI server log file. The variable *maxRequestRate* is the maximum request rate that the Assistant can handle. The variable *maxRequestRate* was set to 300 requests per minute in this simulation. If a request takes one second to be served, a *maxRequestRate* of 300 requests per minute is on average 5 simultaneous requests on the Master or the Assistant in a given interval. One second service time for each request is very conservative; about 95% of the files on the KETI server are below 5 KB in size and take much less than one second to download. The Assistant is capable of serving such a request load without any measurable difference in response time compared to the Master. The variable *threshold* (*threshold* must always be less than *maxRequestRate*) is a "tuning parameter" and is used to determine which platform to use in the next interval.

Table 4.7 shows the results of the simulation. In all cases, $\alpha = 0.25$, *maxRequestRate* = 300, and the values of *threshold* are shown in the first column. Experimenting with different values of $\alpha$ showed that $\alpha = 0.25$ gave the overall least mispredictions for different threshold values. The fourth column shows

the performance impact as the percentage of requests that fall within the intervals where the request rate on the Assistant is more than *maxRequestRate*. It can be seen that the EWMA prediction mispredicts the request rate of next interval in 7% to 20% of intervals for different values of *threshold*. For a larger value of *threshold* (150 request per minute), 15% of predictions were underestimated. For these 15% incorrectly predicted intervals, 8.4% of all requests arrive in these intervals and are likely to experience an increased response time. The impact of this misprediction is further described in the next section. An interval with more than 600 requests per minute does not exist in the one-month KETI server log. With the same earlier assumption of one request per second, this rate is equivalent to about 10 simultaneous connections to the Assistant, which, as shown earlier, does not impose a significant increase in response time for static pages. For active pages (which constitute only about 5% of the overall requests in the KETI server log), however, the impact on response time is much more (about six times increase).

### 4.4.4 Performance Evaluation of the Prototype SWEEP

In the evaluation of the SWEEP prototype, answering the following four questions is of interest:

1. How does the threshold in the prediction method affect the overall Master sleep ratio?

Table 4.6: Variables for simulation evaluation of prediction method used in SWEEP

| | |
|---|---|
| *threshold* | Request rate to switch to Master |
| *maxReqRate* | Maximum request rate for Assistant |
| *prediction* | Prediction from makePrediction() |
| *nextSample* | Sample rate in next interval |
| *sleepCount* | Number of intervals that Master sleeps |
| *underCount* | Number of underestimated intervals |
| *overCount* | Number of overestimated intervals |

Table 4.7: Results from prediction simulation evaluation

| threshold | Sleep | Overestimation | Underestimation | Performance hit |
|---|---|---|---|---|
| 10% | 32% | 15.2% | 7.0% | 0.5% |
| 25% | 50% | 12.5% | 7.8% | 0.8% |
| 50% | 61% | 11.6% | 10.2% | 1.7% |
| 100% | 77% | 9.5% | 8.8% | 4.5% |
| 150% | 82% | 7.5% | 7.8% | 8.4% |
| 200% | 90% | 5.8% | 6.5% | 13.1% |
| 300% | 95% | 3.1% | 4.0% | 21.9% |

2. How does the response time change as the sleep time increases?

3. How does the sleep time and response time of the EWMA prediction differ from the ideal prediction (the oracle) in practice?

4. How does the macSwitch method compare to the httpRedirect method with respect to the request response time and overall sleep ratio?

The performance evaluation of the prototype system was done by replaying the KETI web server log against the SWEEP prototype to determine the energy savings and performance penalty (increase in response time). Although the requests in the KETI server log are made by various clients, in these evaluations all the requests were made from a single client PC connected to the edge router (as shown in Figure 4.3). The capacity of the LAN switch and edge router was 100 Mb/s. The process in the client that replayed the KETI server log was a multi-threaded C program which read the requests from the KETI server log line by line and retrieved the corresponding page by a wget request to the SWEEP prototype at the corresponding time of each request. The program recorded the completion time (server response time) for each request. The Master process was also instrumented to record the percentage of time the Master was powered down (time spent in the Sleeping state in Figure 4.8) out of the total time of the experiment. All the files in the KETI server log were first downloaded from the actual KETI website to the Master and Assistant. The active page requests in the KETI server log were replaced by a request to a PHP page on SWEEP system with two database queries and four images.

To answer questions 1 to 3 (above) the experimental factors were chosen to be 1) the prediction method, and 2) the threshold. The ideal and EWMA predictions were selected with thresholds ranging from 10 to 300 requests per minute. Thresholds above 300 or below 10 would result in the Master being powered on all the time or not at all, respectively, and are thus not interesting. The response variables were 1) request response times and 2) sleep time of the Master. From the sleep time, energy savings could be estimated. To answer question 4 (above), all experiments were performed for both the httpRedirect and macSwitch methods and compared the response variables were compared.

To avoid impractically long experiment times (replaying the entire KETI server log for each factor level would take more than one month to complete) the experiment times were scaled by a factor of 10 as follows. A sample weekday and a weekend (Wednesday, November 3, 2010 and Sunday, November 7, 2010, respectively) were chosen as representative samples of relatively high and low periods of traffic on

the system and experiments were performed for these two days only. All the times in the KETI server log for these days were also divided by 10 (So, replaying a full day of KETI server log would take 2.4 hours instead of 24 hours). The file download times were also scaled by cutting the size of all the files by a factor of 10. The sample time for the policies in the time-scaled experiments was 6 seconds (one minute non-scaled sample time). Two things did not scale well, 1) active pages, and 2) the Master wakeup and sleep transition times. For scaling the active page, one of the queries was removed and the other was modified to include fewer records from the database. As a result, retrieving the active page from the Master took about 10 ms (as opposed to 50 ms for the original) and about 30 ms from the Assistant (as opposed to 150 ms for the original) when the active page request was the only request handled by the platform. These times were still more that one tenth of the original download times and would likely be higher when the server is busier, but they can be considered as reasonably conservative assumptions for the scaled experiments. To scale the transition times, the Master process did not actually put the Master to sleep. Instead, it recorded the time in the Sleeping state after subtracting one tenth of the Master's normal wakeup and sleep transition times (5 seconds was the normal wakeup and sleep transition time as measured on the Master platform) from it. The validity of the scaled experiments was confirmed by comparing a few sample non-scaled experiments on the prototype against the corresponding scaled experiments. The Master sleep times for scaled experiments were within 5% to that of non-scaled experiments. The average request response times were much higher in the scaled experiments most likely because of the higher number of concurrent requests on the platforms when the requests were squeezed into ten times shorter time intervals. However, the relationship between the response times in different non-scaled experiments remained the same as that of for scaled experiments. Since this relationship was of interest (and not the absolute response time values), it was concluded the scaled experiments were adequately valid for this evaluation.

Figure 4.12 shows the Master sleep time and the average request response time for the weekday. Figure 4.13 shows the same for the weekend day. For both days, the prototype shows significant sleep time for the Master with more sleep time for the weekend day, as expected. For a moderate threshold of 150 requests per minute, it can be seen that a sleep time of over 75% is achievable. The sleep times of the httpRedirect and macSwitch methods are almost the same since the method of switching the platform between the Master and the Assistant neither affects the number of requests on the Master in each interval nor causes any change in the fluctuation of the load on the system. So, the intervals in which the Master is powered down remain the same regardless of which switching method is used. However, the prediction

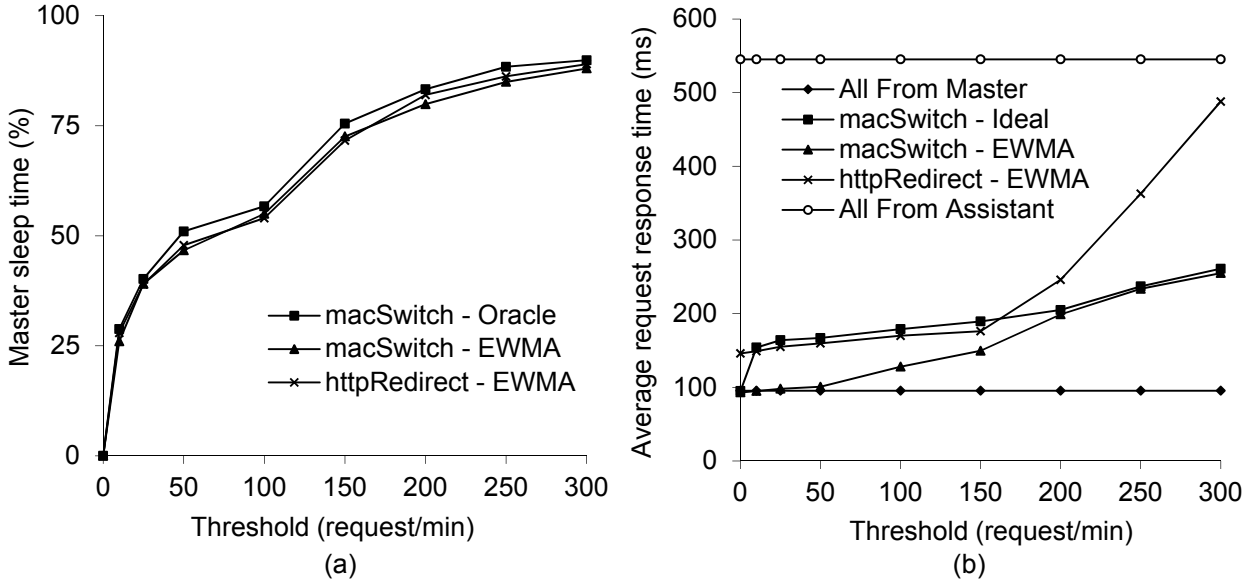



Figure 4.12: Weekday (a) Master sleep time, and (b) response time

method affects the sleep times. As can be seen in Figures 4.12b and 4.13b, the oracle achieves a higher sleep time (although by a very small margin) compared to EWMA. This is due to the oracle not making any overestimation mispredictions. This shows that EWMA can be considered as an accurate prediction method with a low number of overestimation mispredictions, which achieves close-to-ideal sleep times for the Master. It can also be seen that the sleep time grows almost exponentially with the growth of the threshold until it becomes steady beyond moderate (150 requests per minute and above) thresholds. This suggests that a moderate threshold can achieve high sleep times.

Request response times, however, showed a quite different behavior from sleep time. The two traces labeled “All from Master” and “All from Assistant” are given as the performance boundaries, where the former is the average response time where all the requests of the KETI server log are made to the Master, and the latter is the average response time where all the requests of the KETI server log are made to the Assistant. The low boundary, “All from Master”, was almost the same for both days. This is due to the fact that the performance of the Master does not change significantly with different number of simultaneous connections (shown in Section 4.4.2). However, the high boundary is more than two times higher for the weekday. This may be due to the much higher degree of parallel connections (higher load) on the Assistant during a weekday which affects its performance more than the Master (as also shown in Section 4.4.2). The macSwitch method showed lower response times than httpRedirect for any threshold. This is due to the extra roundtrip time caused by the redirection in httpSwitch and the process imposed on the Assistant by handling

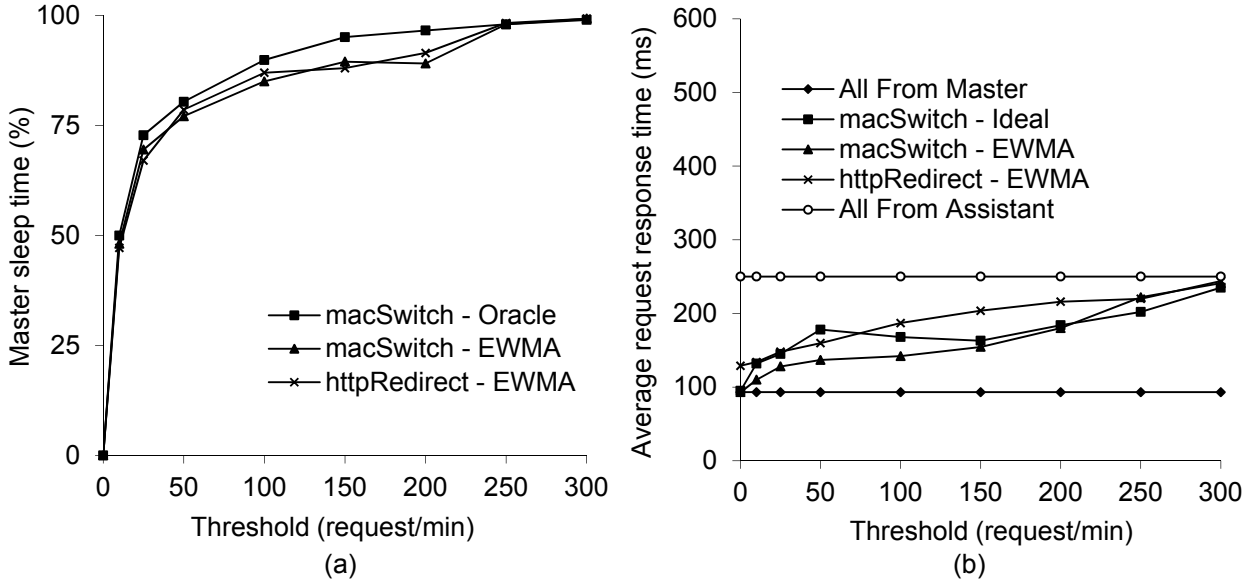



Figure 4.13: Weekend day (a) Master sleep time, and (b) response time

a high number of simultaneous redirection responses during high loads. This effect was more severe in the weekday because of the generally higher load. As the *threshold* increased above 150 requests per minute in the weekday (causing more redirections from the Assistant), the response time increased much faster than those of the macSwitch method. The request response times of the macSwitch method were close to the lower boundary especially for low to moderate thresholds which showes limited performance degradation from using the Assistant. The response time of the macSwitch with EWMA prediction was also tightly bounded by the response time of the same method with oracle prediction. Overestimation mispredictions (which EWMA seems to make more than underestimation) result in lower response times at the expense of less sleep time, as can be seen in Figure 4.12. The increase in response time was limited. For delay-sensitive web content, this increase in delay may cause a noticeable effect to the user. For less delay-sensitive web content, the delay may not adversely affect user experience.

Based on the above discussion the questions considered at the beginning of this section can be answered as follows. For question 1, a moderate threshold (150 requests/min) results in significant sleep time. For question 2, the energy savings come at the expense of relatively higher request response times. However, this increase in response time is limited and may not adversely affect user experience for some lessdelay-sensitive web content. For question 3, EWMA based on past history can predict the intervals of high or low demand with good accuracy. The results showed minor difference between EWMA prediction

and the oracle. Finally, for question 4, the macSwitch method showed better performance than httpRedirect with almost the same sleep times. The reasons are likely to be due to the extra roundtrip time of http redirection and the extra load on the Assistant to handle redirection responses especially when the request load is high.

### 4.4.5 Evaluation of SWEEP Vulnerabilities

In order to evaluate SWEEP's vulnerabilities (see Section 4.2.2), a few scenarios under which the macSwitch vulnerabilities occur were produced. This also helped in determining the likelihood of their occurrence in a real system. These scenarios were as follows:

- For the first vulnerability, producing such a scenario requires an extremely high request rate to the Master due to the small size of an HTTP redirection response and the small amount of processing required for generating these responses independent of the request size. Such scenario could be created in the prototype system when the Master was flooded by about 50 concurrent processes sending requests as fast as the Master can respond. This was done by ab. This request rate translates to about 15000 requests per minute. Such a high request rate was not identified in the KETI one-month web server logs.

- For the second vulnerability, a sequence of requests was crafted to exploit this vulnerability to flood the Assistant. First, the system was given a request rate above *threshold* in order to make it switch the serving platform to the Master. Then, a 40 GB file was downloaded from the Master which takes more than a few sampling intervals (1 minute) to complete. As stated in Section 4.2.2, any request that takes more than a sample interval (1 minute) can potentially reveal this vulnerability. In the setting used for the prototype, downloading files larger than 6 GB takes more than 1 interval (the link from the hybrid system to the client is 100 Mb/s). Then, the system was given no load for the next few sampling intervals which made the Assistant send the *GoToSleep* command to the Master and the Master to redirect new requests to the Assistant while waiting for the download to finish. Then, 100 concurrent connections started sending active page requests to the system (all of which were redirected to the Assistant) and caused the Assistant to drop new requests after responding to about 300. However, again, such a request sequence in KETI's one-month web KETI server logs was not found. In other

experiments used for SWEEP evaluation, a high enough load on the Assistant to cause any delay of the sampling process was not identified.

- With the current setting of the system prototype, crafting a scenario that causes the third vulnerability to occur was not possible. The PC which emulated multiple clients was not capable of producing a high enough load to delay the sampling process in the Assistant. As a result, further investigation of this vulnerability was left to future.

## 4.5 Chapter Summary

In this chapter, a hybrid web server for an SME was designed, prototyped, and evaluated. The SME Web Energy Efficient Platform (SWEEP) is a heterogeneous system based on two platforms – a high performance (and high power) Master and a low performance (and low power) Assistant. No additional hardware is needed for load balancing. In the prototype, a PC for the Master and a plug computer for the Assistant were used with mirrored content. During periods with low request rates, the Assistant had sufficient capacity to handle all requests allowing the Master to sleep. When the request rate was predicted to increase beyond the capacity of the Assistant, the stream of HTTP requests was seamlessly switched to the Master (which was woken-up by a Magic Packet). To a client, the hybrid server appears as a single system image with a a single IP address. The prototype system was evaluated by replaying KETI server logs to the prototype system. It was possible for the Master to sleep about 50% of the time without increasing the response time of the requests. Depending on the maximum tolerable delay, or response time, higher sleep times are achievable (that is, compared against the response time for the Master only). For instance, for a busy weekday, for about 75% total Master sleep ratio the average response time of the requests increased about 100 ms. Based on these results, a system was created with a roughly 10x reduction in power for periods of time when the request rate is low, which occurs about 50% of the time.

## Chapter 5: Timed Redirection to Reduce Energy Use of Hybrid Web Servers

The previous chapter was dedicated to the use of hybrid web servers that contain both Pentium and ARM processors where the smaller (and lower-power) ARM processor can handle a significant fraction of the workload. In Chapter 4, a hybrid server architecture that could reduce the electricity use of Small and Medium Enterprise (SME) web servers was explored where an ARM-based plug computer would proxy or "cover" for a sleeping Pentium-class Master server during periods of low load. In Chapter 4, the Assistant plug computer was a perfect mirror of the Master server. The Assistant would serve content during periods of low load with the Master allowed to sleep; the Master would serve content during infrequent periods of high load. As a next step, a different approach to exploiting the hybrid server architecture to save energy will be explored in this chapter. For this next step, existing Internet protocols are modified to enable the Assistant to effectively schedule the sleep time of a Master where the Master serves all content and the Assistant acts as a redirector of requests [77]. The Assistant does not store or serve content.

HTTP is the key protocol for transferring data between servers and clients. In this chapter, a mechanism for existing HTTP redirection whereby client requests can be redirected not only in space (that is, directly to another server) but also in time [77]. This HTTP timed redirection enables requests for multiple clients to be coalesced and delayed in a controlled fashion which allows many short idle periods in servers to be coalesced into fewer, longer idle periods. During these periods, the servers can sleep and save energy. As demonstrated in this chapter, timed redirection can be used for reducing energy use for a growing class of delay-tolerant services such as application updates and file backup. This is a new way to use the hybrid web server architecture to reduce energy use of web-based services.

HTTP redirection has long been used for load balancing of incoming requests to multiple web servers (for example, [7]). HTTP redirection works in "real time" – time shifting a redirected request is not possible. In this chapter, a new method for time shifting incoming requests to a later time is proposed and evaluated. Such time shifting can save energy by coalescing requests into pre-determined awake periods for a given

server. Time shifting will increase the response time for a given request. For some applications – such as web surfing – such an increase in response time would likely be unacceptable. For a growing number of applications, however, there is a degree of delay tolerance that can be exploited. For example, automatic app updates for smart phones are delay tolerant. Large file downloads may also be delay tolerant to the extent that their download start time could be delayed for a short period of time. Online backup services such as Carbonite [16] may also be able to benefit from time shifting of their requests. Backing up a recently changed file does not take place instantly; there is a few seconds of delay between when a file is changed and the time it is synchronized with the version stored in the user's online backup space. This delay could be increased without significantly changing the performance or reliability of the backup service. The idea of a "just noticeable difference" (also known as Weber-Fechner law) has been studied by psychophysicists and suggests that changes on the order of 10% may be unnoticeable [98]. So, for example, increasing a file download time by 10% by delaying the start of the download may be acceptable. It may also be possible that customers of mobile data plans would be willing to accept an occasional increased delay for a lower service cost.

The contributions of this chapter are the proposal, system design, implementation, and evaluation of HTTP timed redirection to achieve energy savings in a hybrid web server.

## 5.1 Modifications to HTTP Design to Enable Timed Redirection

HTTP already supports redirection. However, HTTP does not support a delay mechanism useful for scheduling purposes. In HTTP, a web server can issue a redirect response (for example, in response to a GET request) to indicate that the requested resource should be accessed at a different location than the URI specified in the request [31]. Different status codes in a redirection response notify the browser of the proper action to take in subsequent requests. For instance, response code 301 means that the resource is permanently moved to the new location, whereas code 307 means the request is temporarily moved and future requests need to still be made to the original URI. The HTTP redirection capability can be used for load balancing (SWEB [7], for instance). The work in this chapter builds upon SWEB to add redirection in time (and not only space). Methods of delaying HTTP requests already exist. A field called Retry-After exists in a 503 response (Service Unavailable) which can be set by website administrators to inform users of when to try again when the website becomes temporary unavailable. The Retry-After field implies that

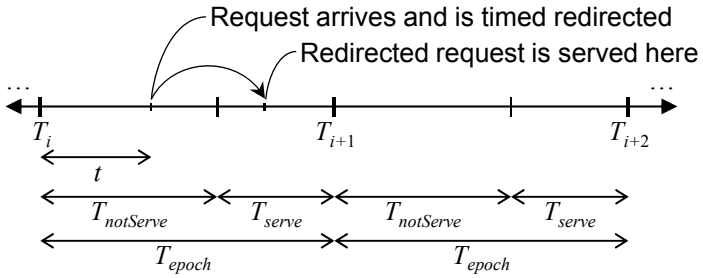



Figure 5.1: Arrival of requests and epochs of the Master

the resource will be available to the client anytime after the given delay. The proposed delaying method in this chapter (described next) is different in that the client is required to retry the request at exactly the current time plus the given delay – this pre-determined delay is given to the client by a new field in the HTTP response called Retry-Delay. Thus, Retry-Delay can be used for scheduling purposes (as used here) where Retry-After cannot. Another means of delaying client requests is refresh element in the metadata of an HTML page. The client must be capable of interpreting HTML (as a browser is capable of, for example). An alternative URI can be coupled with refresh meta element to automatically redirect users to another location after a certain time, but not with the precision required for scheduling applications (such as the one explored in this chapter).

HTTP timed redirection is a means of coalescing HTTP requests to allow the Master in a hybrid server to periodically sleep. In this implementation of a hybrid server, the Assistant is fully powered-up (awake) all the time while the Master is put to sleep periodically in sleep-wake epochs as shown in Figure 5.1 and is thus awake only during defined periods of time. Figure 5.1 shows a series of not serving (or asleep) and serving (or awake) periods for the system. Each epoch of duration $T_{epoch}$ consists of a not serving period of predetermined duration $T_{notServe}$ and a serving period of variable duration $T_{serve}$. During the $T_{notServe}$ period, the system is sleeping or transitioning between wake and sleep. During the $T_{serve}$ period, the system is awake and serving. There is no web content available on the Assistant. The Assistant implements an HTTP server that only does redirection. The redirection notifies a client that it has to make another request to the new URI of the resource on the Master. All client HTTP requests are sent to the Assistant – the Assistant IP address is associated with the domain name of the website. When the Master is awake, requests sent to the Assistant are redirected by the Assistant to the Master with no delay. However, when the Master is sleeping, these requests are delayed in time (using timed redirection) for redirection to the Master at a later time when

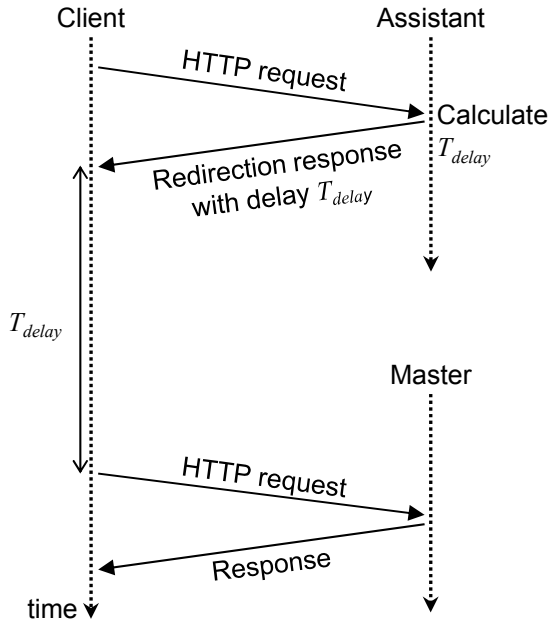



Figure 5.2: Timed redirection message flow

it is awake. This delay for requests allows for short server idle periods to be coalesced into fewer and longer idle periods during which the Master can sleep and energy use can approach energy proportional.

### 5.1.1 Description of the Changes in the HTTP Protocol

The new HTTP timed redirection requires a new HTTP method, a new response field, and a new behavior for a web client that uses HTTP to get files from, or put files to, a web server, as follows:

- A new Retry-Delay field similar to the existing Retry-After field in HTTP 3xx responses. The Retry-Delay field contains a retry delay in seconds with a fraction to the precision of the system (Figure 5.3).
- The capability for an HTTP server to assign delays (in the new Retry-Delay field) to HTTP 3xx redirection responses based on a given policy.
- The capability for an HTTP client to delay a redirected request by the Retry-Delay value of a received HTTP 3xx response.

The HTTP message flow of a timed-redirected request is shown in Figure 5.2 in the context of a hybrid web server. The initial resource request from the client is made to the Assistant. The Assistant receives the

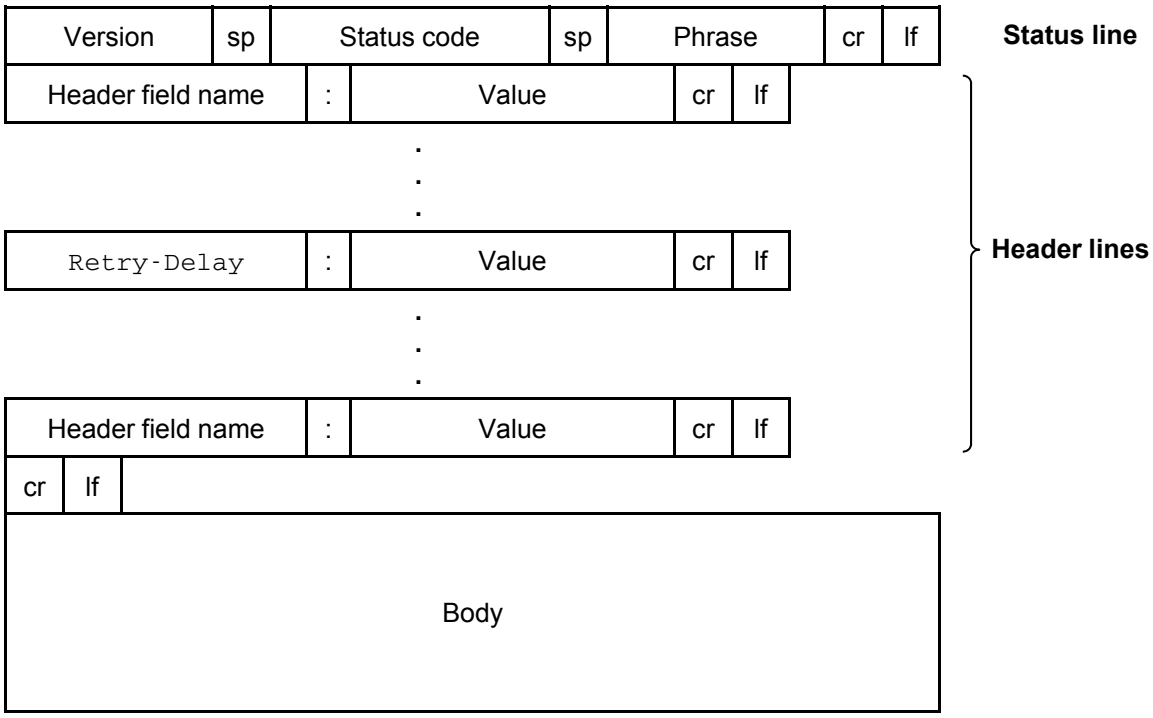



Figure 5.3: Format of a generic HTTP response message with the Retry-Delay field added

request and responds with a redirection response with delay, $T_{delay}$, in the Retry-Delay field. The redirection response from the Assistant to the client contains the URI of the resource on the Master as the new temporary location of the resource. Upon receiving the response, the client delays for $T_{delay}$ time and then automatically resends the request to the Master. The request is finally served by the Master.

### 5.1.2 Using the Protocol with the Modifications for Timed Redirection

In the hybrid web server, HTTP timed redirection is used to redirect in time requests that arrive during a Master sleep period to a later Master awake period within each epoch. At the beginning of each epoch, the Master sends a message to the Assistant indicating that it is transitioning to sleep, and then it transitions to sleep. This begins a fixed-duration Master sleeping period which is known by the Assistant. The Assistant then starts redirecting the incoming requests with a delay. This delay is calculated so that all the delayed requests are redirected to the start of the next awake period. When the Master's queue becomes empty, the current awake period (and epoch) ends and a new epoch begins with the Master entering a sleep period again, and this sequence continues. Figure 5.4 shows the scheduling method executed in the Assistant. The

```
loop forever
  if (a request is received in epoch T_i at time t)
    if  (t < T_notServe)
      send a standard redirect message
    else if (t ≥ T_notServe)
      send a timed redirect message with Retry-Delay = T_delay
```

Figure 5.4: Scheduling method executed by the Assistant

duration of sleep periods is fixed and is determined by the type of application being served by the server. Thus, all requests that arrive while the Master is sleeping, plus the ones arriving while the Master is awake, are served in the awake period. This effectively increases the utilization of the Master when measured during awake periods only, but allows for headroom for high load periods by canceling sleep periods when necessary. The duration of sleep periods can be determined such that the requests arriving during sleep periods can be fully served in the awake periods without overloading the Master.

## 5.2 Analytical Model of Timed Redirection

Timed redirection used in a hybrid server coalesces requests in a distributed fashion – the requests are held in the requesting clients. From a queueing point of view, the Master can be considered as a single-server queueing system where the server takes vacations – a vacation period starts whenever the queue is empty. During a vacation the server does not serve requests. The queueing model is the same as Figure 3.2. However, the analysis in Section 3.1 was for deterministic arrivals and service times. The model in this section assumes Poisson arrivals and general distribution for service times. The server vacations are controlled by the server control unit which monitors the number of requests queued in the request queue. Requests arrive at a rate $\lambda$ and are served by the server (when active) at a rate $\mu$. The utilization of the server is thus $\rho = \lambda/\mu$ (this is the ratio of busy time). When requests arrive according to a Poisson distribution and requests are served first-come first-served with an arbitrary service time distribution, the system becomes an M/G/1 queue with vacations. Such a system has a well-known formula for solving for the mean response time, $W$, which is derived directly from the Pollaczek Khinchin (P-K) formula. Let $X$ and $V$ be random variables for the service and vacation times, respectively. Then,

$$W = \frac{\lambda E[X^2]}{2(1-\rho)} + \frac{E[V^2]}{2E[V]}. \tag{5.1}$$

## 5.3 Implementation and Performance Evaluation of Timed Redirection

The performance of timed redirection for a hybrid web server was evaluated using both analysis and experiments on a prototype implementation. Request arrivals that follow a Poisson distribution were considered, since it seems reasonable given independently generated requests from a large number of independent clients. For example, requests for application updates or file backup uploads will likely follow a Poisson distribution given the independence of the generated requests. For the request sizes, both fixed length and highly variable length were considered (for this latter case, a Bounded Pareto distribution was used to model heavy tailed distributions for file size [27]). Request size models the size of an application update to be downloaded to a client.

### 5.3.1 Implementation of a Prototype Hybrid Server with Timed Redirection

To experimentally evaluate timed redirection for a hybrid server, a test bed was built with a configuration as shown in Figure 4.3. For the Assistant, a Sheeva plug computer that contained a 1.2 GHz ARM processor was used. For the Master, a Dell Pentium4-based PC was used. Both the Assistant and Master ran Apache on Ubuntu.

Software was developed for the Assistant and Master to implement the hybrid server operation where the Master periodically sleeps. A messaging protocol between the Assistant and Master was implemented to allow the Master to notify the Assistant that it would go to sleep and for how long it would sleep ($T_{notServe}$). The awake time, $T_{serve}$, as described, was not fixed – it was as long as it would take to clear the Master queue of all requests. Then, a new epoch (and sleep period) would begin. The Assistant HTTP redirection service was implemented as a multithreaded process written in C. One thread listened for messages from the Master. Upon receiving a message indicating the beginning of a new epoch, another process would modify the mod-rewrite rules of Apache to redirect requests to the Master with a delay such that all the requests would be retried to the Master at the beginning of the next awake period in the current epoch. Rather than implementing a new field, the new Retry-Delay field was emulated in the location field of the 3xx response, separated from the URI with a special character. The Master service was also implemented as a multithreaded process in C. The process kept track of the server's request queue by monitoring pending requests in Apache's mod-status module scoreboard. When the request queue became empty, the process

would notify the Assistant of the beginning of a new epoch (that is, that the Master would go to sleep) and put the Master to sleep for a $T_{notServe}$ period of time. A client load generator was also a part of the test bed. The client load generator was a single PC that emulated multiple clients as a multithreaded C process. Each thread was capable of sending GET requests to the hybrid server at a given rate with request interarrival times generated from a specified distribution. The load generator timed each request response time to calculate the mean response time for all requests made.

### 5.3.2 Evaluation of the Prototype Hybrid Server with Timed Redirection

For the experimental evaluation, the response variables of interest were the percentage of time the Master was sleeping and the average delay to service a GET request. The experimental factors were:

1. $T_{notServe}$
2. request arrival rate ($\lambda$)
3. file size
4. Master multi-processing mode

The service rate ($\mu$) was the data rate of the outgoing link (100 Mb/s) divided by the mean file size. The factor levels were as follows. The factor $T_{notServe}$ was fixed to 30 seconds. The sleep-to-wake and wake-to-sleep transition times of the Master were about 4 seconds each, and inclusive in $T_{notServe}$. The choice of 30 seconds for $T_{notServe}$ was to enable the Master to serve the full coalesced load in a sleep period in the subsequent awake period. When $T_{notServe}$ was set to more than 30 seconds, the Master appeared to be dropping requests at high loads. How this can be addressed is discussed in the next section. Although this value for $T_{notServe}$ seems very short relative to the transition times (roughly 26% of a $T_{notServe}$), as argued in [72], transition times in the order of 100s of milliseconds or less may be possible in future commercial grade servers. Therefore, it was assumed that in practice, such a $T_{notServe}$ would be reasonable compared to conventional server transition times. The client and the hybrid server were in the same room, and the link between them was a 100 Mb/s Ethernet link. The interarrival time of requests was exponentially distributed (Poisson arrivals) in all experiments. Two distributions were used for file size; fixed-size at 1 MB and Bounded Pareto with mean size 1 MB and minimum and maximum files sizes of 100 KB and 100 MB,

respectively, to represent a small and a mid-size file on the Internet. The request rate was determined such that the load on the server varied between 5% and 95%. Requests were sent to the hybrid server for at least 3 hours per factor level while sleep and response time measurements were made.

Two sets of experiments were performed. One set of experiments used the default multi-processing mode of Apache. In this mode, Apache creates a thread pool and serves requests in parallel, one by each thread. In this mode, service can be closely approximated as processor sharing with multiple requests being served at one time. This set of experiments was referred to as "threaded". A second set of experiments was based on changing the multi processing mode of Apache to a non-threaded mode (this set was called "non-threaded"). The non-threaded mode more closely models first-come first-served service discipline and allows for a comparison with the M/G/1 analytical model. Experimental results are compared directly with analytical results for M/D/1 for fixed, or deterministic file size and M/BP/1 for Bounded Pareto-distributed file sizes. That is, the "G" for General service distribution in the M/G/1 model was modeled as Deterministic (D) and Bounded Pareto (BP).

Figures 5.5 and 5.6 show the results from non-threaded and threaded experiments, respectively. Figure 5.5a shows the average fraction of time in sleep as a function of load ($\rho$), which is $1-\rho$ by theory. Figure 5.5b shows the average request delay as a function of load. It can be seen that the actual sleep time is slightly less than theory, but is very close. It can also be seen that the average delay is dominated by the sleep time at low loads and then increasing at higher loads. Actual and theory match very well. For the threaded experiments the average delay in Figure 5.6b is significantly greater than that of the non-threaded case. This is because in a processor sharing model, all jobs in the system at a given time receive service, thus increasing the average service time in many cases. For both threaded and non-threaded experiments (Figures 5.5a and 5.6a), it can be seen that the sleep time is very close to $1-$ load (especially for low and medium load levels of up to 50%), which shows that timed redirection can bring the server very close to energy-proportional. In the testbed, setting $T_{notServe}$ to longer than 30 seconds resulted in the Master dropping requests when the load was more than 75%. This appears to be due to the number of requests redirected to be queued at the start of an awake period being too large and some client connections being dropped. For future work, this potential problem needs to be addressed with, possibly, a notion of batching of request redirections. The Assistant will batch redirection times and inform the Master at the start of each awake period how many batches to expect and not to power down until the last batch is complete. The current implementation is thus a single batch system.

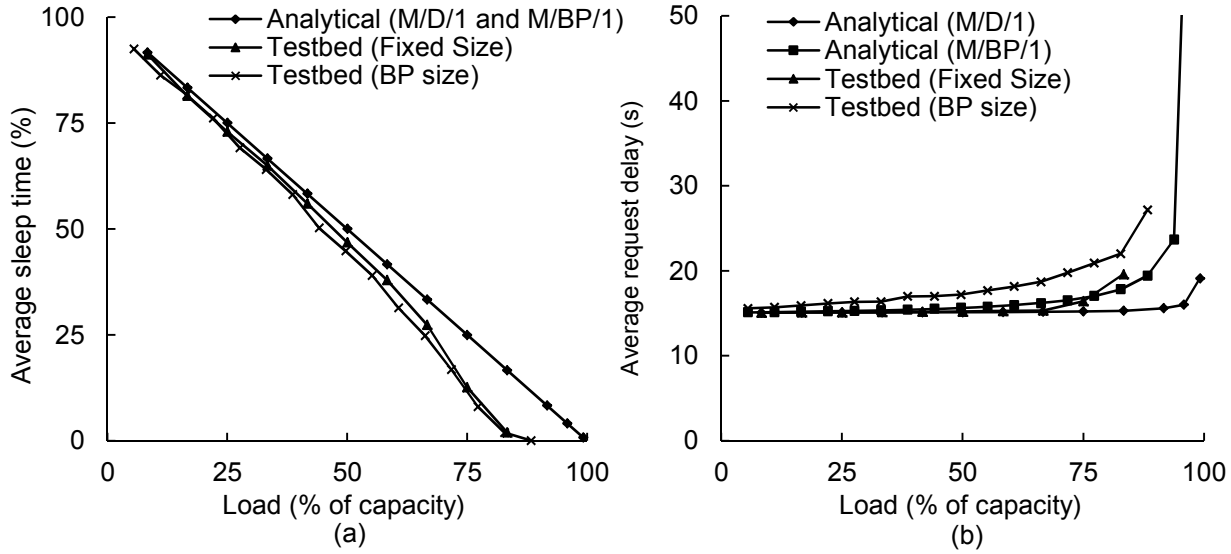



Figure 5.5: Non-threaded experiment results – (a) fraction of time in sleep, and (b) request delay

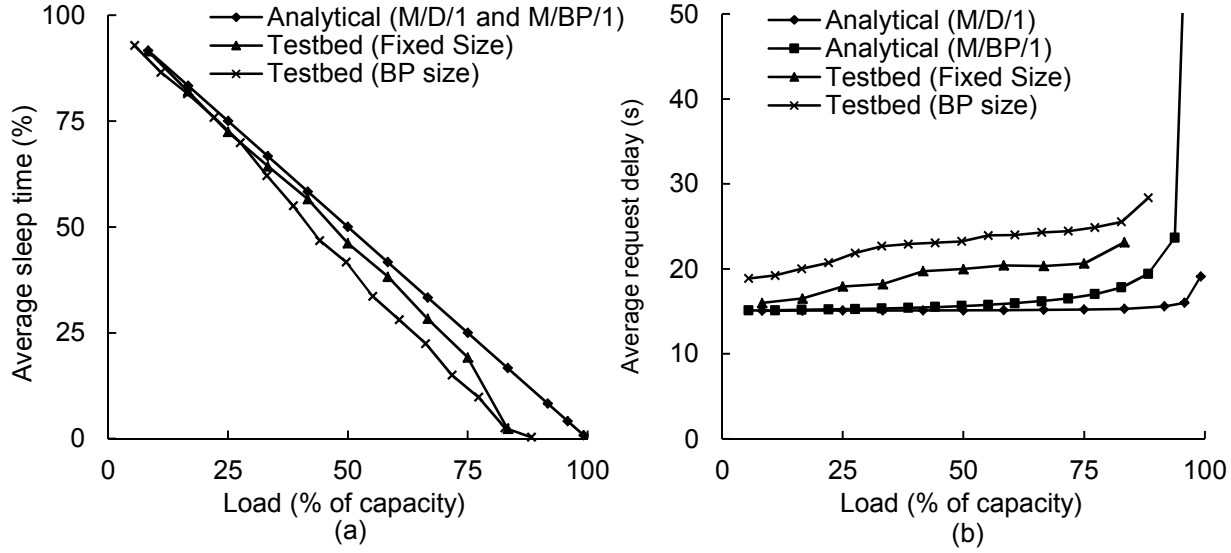



Figure 5.6: Threaded experiment results – (a) fraction of time in sleep, and (b) request delay

## 5.4 Chapter Summary

The hybrid web server is another possible approach to reducing energy use for delay-tolerant web-based services in Small and Medium Enterprises and in data centers. In this chapter, it was shown how HTTP timed redirection can enable very-close-to-energy-proportional operation of a hybrid web server for delay-tolerant services. The proposed change to HTTP permits redirection to be precisely delayed by a predetermined Retry-Delay. This added delay can be used to coalesce client requests to enable servers to sleep. Adding

delay to a web-based service seems counter-productive given than most efforts are focused on reducing delay. However, for a growing class of delay tolerant applications such added delay may not have any measurable effect on user quality of experience and the reduced energy costs can result in lower service cost to the user. In particular, application update and file back-up can be two popular Cloud Computing services that can reduce their energy use by employing HTTP timed direction in a hybrid web server to enable longer idle periods for servers to sleep.

# Chapter 6: Summary and Future Work

The electricity consumption of ICT systems costs about $15 billion per year in the US. Generating this electricity also contributes to climate change by emitting almost 2% of all $CO_2$ generated by human activities and industries worldwide. There is a great opportunity to reduce this consumption; most ICT systems are lightly loaded which means there are idle time periods where they are not utilized. New methods are needed to exploit these idle periods for putting systems to sleep in order to save energy while still maintaining an acceptable level of performance. This dissertation studied three such methods for Ethernet links, Ethernet LAN switches, and data servers using analytical modeling, simulation modeling, and prototyping. These methods seek a common goal; to bring the system closer to energy-proportionality. Energy-proportionality is a notion of minimizing energy waste by utilizing idle periods for sleeping. The challenge is that sleeping may not be possible in some idle periods since the transition times associated with sleep and wakeup can be more than the length of an idle period, making sleep impossible. A solution, applied in the methods studied in this dissertation, is to consolidate idle periods to longer ones suitable for sleeping. This was done by coalescing jobs by scheduling them to be served in batches of back-to-back jobs.

## 6.1 Summary of the Investigated Methods

The first method studied was packet coalescing for EEE. In packet coalescing, multiple packets are coalesced in the sending buffer of an EEE-enabled Ethernet interface in order to extend idle periods between packet arrivals and maximize the opportunity for sleeping in these extended periods. Packet coalescing was modeled analytically. Packet coalescing was also modeled using simulation. Performance evaluation using the simulation model showed that packet coalescing could add significantly to EEE savings with an additional delay overhead to the packets that may be negligible compared to the normal end-to-end delays in the Internet. The concept of packet coalescing was then applied to SOHO Ethernet switches where the

entire switch could be put to sleep periodically after stopping traffic on all ports. The method was called synchronized coalescing, since traffic would pause on all the connected ports in a synchronized fashion before putting the switch to sleep. An extension of synchronized coalescing, called adaptive coalescing, was then developed to enable the switch to adapt to high loads by extending the periods of staying on. It was shown that synchronized and adaptive coalescing can bring an Ethernet LAN switch very close to energy-proportionality.

The second method was SWEEP, a hybrid web server architecture in SME environments. SWEEP is a hybrid web server with two platforms which appear as a single server with a single IP address to outside users. The first platform is a low-power, low-performance Plug PC which serves the server load most of the time as the server load in SME environments tends to be low for the majority of time. The second platform is a high-power high-performance server which takes over serving requests in rare periods of high load. Performance evaluation of SWEEP using a system prototype and web server logs from a representative SME (KETI) showed that the hybrid server operates very close to energy-proportional.

The third method was timed redirection in hybrid servers. Timed redirection is coalescing of HTTP requests in clients in a distributed fashion. In timed redirection, requests are delayed to the future. The delay occurs in the clients. The delay is calculated so that no requests arrive to the server for a period of time, thus enabling the server to sleep, and then the requests are retried to the server at the same time, thus enabling the server to either work at full utilization, or sleep, at all times. This makes the server operate very close to energy-proportional. The delay caused by timed redirection is not acceptable for some applications (such as real time services). However, for a growing class of delay-insensitive applications, such as online backup and app updates on handheld devices, the delay may be unnoticeable or of no significance to the user, or justifiable by some sort of business incentive.

## 6.2 Future Work

The research done in this dissertation can be continued in several directions, including the following: The analytical model for coalescing can be extended. Future work includes extending the model to a general case system comprised of multiple coalescer outputs of which "fan-in" to a downstream queue. This system represents, for example, an Ethernet LAN switch with multiple EEE-enabled interfaces connected. Packet coalescing may cause an increased burstiness of the traffic, which can have adverse effects on the

downstream network equipment such as routers and switches. The possible effects of increased burstiness of the traffic need to be evaluated. The effects on downstream queues (for example, in downstream routers) depend on three factors:

- The characteristics of the traffic stream generated by the application in the system that implements coalescing.
- The capacity, or service rate, of the downstream queue.
- The setting of the coalescer parameters.

Packet coalescing, as implemented in this work, generates bursts that last in time scales in the order of a few microseconds. If the original traffic stream in the system that implements coalescing is already bursty, then packet coalescing may not significantly alter the characteristics of the original stream. As a first step, the burstiness of the original traffic stream and the outgoing traffic from the downstream queue can be compared. One way to do so is to measure the CoV of the interarrival times to the coalescer and interdeparture times of the outgoing packets from the downstream queue and compare them. If the CoVs are not significantly different, it may be an indication that the burstiness of the original traffic "covers" the microbursts caused by the coalescer. On the other hand, if the original traffic stream is smooth, then coalescing would make it bursty at a small time scale. It is unclear if the time scale here is large enough to cause significant effects to a downstream queue. The time scale of the bursts generated by a coalescer is generally much smaller than that of the burstiness caused by applications and users. EEE with packet coalescing should also be studied more in the context of the Internet protocols. Its effects on the TCP/IP congestion control mechanisms should be further investigated. Also, the effects of synchronized coalescing on interactive applications such as gaming and web surfing should be measured and studied. An analytical and simulation evaluation of synchronized and adaptive coalescing when it is extended to multiple hops at the same time would be useful. Hybrid systems show great potential for reducing energy use of ICT systems and services; additional work is needed in this promising direction. Installing a hybrid server in a real SME – possibly at KETI – and hosting a real website and workload is possible future work. Open questions that were uncovered during the study of hybrid servers in this dissertation include:

- How can SWEEP be extended to other Internet services that SME servers may host including email, DNS, file back-up, and so on?

- What additional services specifically related to energy use reduction can SWEEP implement (this might include new forms of scheduling of requests and other form of shaping the workload to increase sleep time)?
- Can the Assistant be more closely integrated with the Master, for example as multiple processors on the same motherboard or as heterogeneous cores within a single processor and achieve both lower cost and better energy efficiency?

Changing the hybrid server architecture to suit various applications, and possibly increasing the energy savings, is another direction of future work regarding hybrid servers. Such alternative architectures can include the following:

- As opposed to the current architecture where the Master's content is replicated on the Assistant, the Assistant can only be capable of serving some content available on the Master. The Assistant can be used as a cache for the Master's content and only serve the content that is requested most frequently. A challenge which should be addressed would then be the synchronization of the contents of the Assistant and the Master. If such architecture is to be explored, the FSMs for the Assistant (Figure 4.7) should be modified. The modification would be to add a transition similar to Transition 1 from the Serving state to the WaitMasterWake state where the triggering condition should be the unavailability of a resource on the Assistant.
- As opposed to the current architecture where there is one high-power high-performance platform handling the peak loads, an array of low-power low-performance platforms can be used to handle the peak loads. Such architecture has been investigated for massively parallel processing (FAWN [6], for instance). However, one important characteristic of SWEEP which is eliminating the need for a separate load balancer may not be possible for this alternative architecture. The reason is that a switch does not normally associate more than one MAC address to one IP address, so at any time it would not be possible for more than one platform to serve requests destined to an IP address in the network (unless the switch broadcasts packets to all connected ports, which would require modifications to the switch). But using low-power platforms may significantly reduce the overall consumption of the hybrid server. New architectures are possible future work.

Lastly, timed redirection can be further investigated. Next steps involving timed redirection include:

- Identifying more application types to which timed redirection is applicable and the tolerable delay for each type.
- Investigating how the scheduling method can be adaptively set to maximize energy savings (or minimize delay) for a given delay (or energy) constraint.
- Implementing and evaluating batching of request redirection to handle overload situations.
- Gaining a better analytical understanding of mean response time for the threaded mode of operation.

## 6.3 Broader Impact and Intellectual Merit

Widespread use of the methods studied in this dissertation can bring broad and tangible benefits to society. These methods can enable energy savings – in the form of reduced energy consumption of Ethernet links and data servers – of hundreds of millions of dollars and reduction in $CO_2$ generation of billions of lbs in the US [24]. Estimations leading to these potential savings follow.

There are currently more than 65 million active active 10 Gb/s Ethernet links in the US, most of which are data center links in rack switches. Assuming 3% utilization (slightly higher than an edge link) on each link, 5.5 W power consumption, and being powered-on all the time, about 2.25 TWh of energy per year can be saved on these links using EEE (a 72% reduction at this level of utilization based on the results presented in Figure 3.7a). An additional 18% savings can be achieved if packet coalescing with the large coalescer is used on these links (again, Figure 3.7a). This is an additional 0.41 TWh of energy per year. A total of about 260 TWh of energy per year can be saved by utilizing EEE with packet coalescing on all these links. At the current average electricity rate, these savings translate to about $266 million per year in the US. This is also equivalent to about 3.4 billion lbs of $CO_2$ emissions per year.

Similar estimates can be made for synchronized and adaptive coalescing too. Using the power consumptions in the On and Off states, and the total number of SOHO switches in the US (Section 3.3), it can be estimated that the potential energy savings that can be obtained by deploying adaptive coalescing in all the future SOHO switches would be approximately 3.5 TWh per year in the US. This is about $350 million in monetary savings and about 5.3 billion lbs of reduction in $CO_2$ emissions.

Savings that can result from utilizing SWEEP can be estimated using the consumption of a typical SME server and the number of them in the US. Consider a 200 W non-hybrid server (this is a conservative

assumption, SME web servers typically consume more power) that is fully powered-on 24/7 versus a hybrid server with a 200 W Master and 10 W Assistant. With 50% sleep time for the Master, the hybrid server becomes a 110 W always-on server. This is 45% reduction in the overall power consumption which translates to over 780 kWh per year in energy savings. These savings are roughly equivalent to $70 in monetary savings (using current average electricity price in the US, $0.10/kWh) per server once the cost of the Assistant is recovered by the savings. Based on the US Census Bureau 2008 statistics on small businesses, there were 526,307 firms with 20 to 99 employees in the US in 2008. Information on exactly how many of these firms use a dedicated web server was not available. As a first-order rough estimate, it can be said that savings of between $1 million and $5 million per year nationwide are possible if between 3% and 13% of these firms use a dedicated web server and utilize SWEEP.

As another impact, applications of coalescing of requests in other disciplines – potentially outside ICT – can be considered. In general, coalescing can potentially apply to domains where queueing models apply such as manufacturing, transportation, amusement parks, etc. In these domains, coalescing can be done on requests for a highly valuable resource (equivalent to energy in the context of this dissertation) that can be better utilized in longer time periods rather than in smaller time periods.

The intellectual merit of this dissertation is in introducing a new way of reducing energy use of network links and data servers by the coalescing of packets (for network links) and server requests (for data servers). Coalescing enables idle periods to be better exploited for sleeping and thus allows for energy proportionality to be achieved. A better understanding of the trade-offs of coalescing on network quality of service and user quality of experience is an intellectual direction to be explored.

## List of References


[1] "ab – Apache HTTP Server Benchmarking Tool," 2011. URL: http://httpd.apache.org/docs/2.0/programs/ab.html.

[2] Y. Agarwal, S. Hodges, J. Scott, R. Chandra, P. Bahl, and R. Gupta, "Somniloquy: Augmenting Network Interfaces to Reduce PC Energy Usage," *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pp. 365–380, April 2009.

[3] Y. Agarwal, S. Savage, and R. Gupta, "SleepServer: Energy Savings for Enterprise PCs by Allowing Them to Sleep," *Proceedings of the USENIX Annual Technical Conference*, pp. 22–36, June 2010.

[4] G. Ananthanarayanan and R. H. Katz, "Greening the Switch," *Proceedings of the Conference on Power Aware Computing and Systems*, pp. 1–5, December 2008.

[5] G. Anastasi, M. Conti, M. Di Francesco, and A. Passarella, "Energy Conservation in Wireless Sensor Networks: a Survey," *Ad Hoc Networks*, vol. 7, no. 3, pp. 537–568, May 2009.

[6] D. Andersen, J. Franklin, M. Kaminsky, A. Phanishayee, L. Tan, and Vijay Vasudevan, "FAWN: a Fast Array of Wimpy Nodes" *Proceedings of the ACM SIGOPS Symposium on Operating Systems Principles*, pp. 1–14, October 2009.

[7] D. Anderson, T. Yang, V. Holmedahl, and O. H. Ibarra, "SWEB: Towards a Scalable World Wide Web Server on Multicomputers," *Proceedings of the 10th International Parallel Processing Symposium (IPPS)*, pp. 850–856, April 1996.

[8] J. Baliga, K. Hinton, and R. Tucker, "Energy consumption of the Internet," *Proceedings of the Joint International Conference on Optical Internet and the 32nd Australian Conference on Optical Fibre Technology (COIN-ACOFT)*, pp. 1–3, June 2007.

[9] 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," *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*, pp. 164–177, December 2003.

[10] L. A. Barroso and U. Holze, "The Case for Energy-Proportional Computing," *IEEE Computer*, vol. 40, no. 12, pp. 33–37, December 2007.

[11] L. Benini, A. Bogliolo, G. A. Paleologo, and G. De Micheli, "Policy Optimization for Dynamic Power Management," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 18, no. 6, pp. 813–833, June 1999.

[12] L. Benini and G. De Micheli, "Dynamic Power Management: Design Techniques and CAD Tools," Kluwer Academic Publishers, Norwell, MA, USA, 1998.

[13] F. Blanquicet, "PAUSE Power Cycle: A New Backwards Compatible Method to Reduce Energy Use of Ethernet Switches," White Paper, Version 1.0, *Ethernet Alliance*, April 2008.

[14] R. Bolla, R. Bruschi, A. Carrega, and F. Davoli, "Theoretical and Technological Limitations of Power Scaling in Network Devices," *Proceedings of the Telecommunication Networks and Applications Conference (ATNAC)*, pp. 37–42, October/November 2010.

[15] Broadcom Corporation, "8-GE port switch with integrated SerDes," BCM5398 Datasheet, 2006.

[16] "Carbonite Online Backup Software | Cloud Backup for Home and Bussiness," 2013. URL: http://www.carbonite.com.

[17] A. Chandrakasan, S. Sheng, and R. Brodersen, "Low-Power CMOS Digital Design," *IEEE Journal of Solid-State Circuits*, vol. 27, no. 4, pp. 473–484, April 1992.

[18] J. Chase, D. Anderson, P. Thakar, A. Vahdat, and R. Doyle, "Managing Energy and Server Resources in Hosting Centers," *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP)*, pp. 103–116, October 2001.

[19] J. Chase and R. Doyle, "Balance of Power: Energy Management for Server Clusters," *Proceedings of the 8th USENIX Workshop on Hot Topics in Operating Systems (HotOS)*, May 2001.

[20] L. Chiaraviglio, M. Mellia, F. Neri, "Energy-aware Backbone Networks: a Case Study," *Proceedings of Green Communications Workshop in Conjunction with IEEE ICC*, pp. 1–5, June 2009.

[21] L. Chiaraviglio, M. Mellia, F. Neri, "Reducing Power Consumption in Backbone Networks," *Proceedings of IEEE International Conference on Communications*, pp. 1–6, June 2009.

[22] K. Christensen and F. Gulledge, "Enabling Power Management for Network-Attached Computers," *International Journal of Network Management*, vol. 8, no. 2, pp. 120–130, March/April 1998.

[23] K. Christensen, C. Gunaratne, B. Nordman, and A. George, "The Next Frontier for Communications Networks: Power Management," *Computer Communications*, vol. 27, no. 18, pp. 1758–1770, December 2004.

[24] K. Christensen, P. Reviriego, B. Nordman, M. Bennett, M. Mostowfi, and J. A. Maestro, "IEEE 802.3az: The Road to Energy Efficient Ethernet," *IEEE Communications*, vol. 48, no. 11, pp. 50–56, November 2010.

[25] B.-G. Chun, G. Iannaccone, G. Iannaccone, R. Katz, G. Lee, and L. Niccolini, "An Energy Case for Hybrid Datacenters," *Proceedings of ACM Symposium on Operating Systems Principles (SOSP) Workshop on Power Aware Computing and Systems (HotPower)*, pp. 76–80, October 2009.

[26] E.-Y. Chung, L. Benini, and G. DeMicheli, "Dynamic Power Management Using Adaptive Learning Tree," *Digest of Technical Papers, IEEE/ACM International Conference on Computer-Aided Design*, pp. 274–279, November 1999.

[27] M. Crovella and A. Bestavros, "Self-Similarity in World Wide Web Traffic: Evidence and Possible Causes," *IEEE/ACM Transactions on Networking*, vol. 5, no. 6, pp. 835–846, December 1997.

[28] G. Epps and A. Baldini, Talk in *Cisco Green Research Symposium*, March 2008.

[29] "Electric Power Monthly," U.S. Energy Information Administration, February 2012. URL: http://www.eia.gov/electricity/monthly/.

[30] M. Ellis, "Gadgets and Gigawatts: Policies for Energy Efficient Electronics," International Energy Agency, 2009.

[31] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Hypertext Transfer Protocol – HTTP/1.1," IETF, RFC 2616. URL: http://tools.ietf.org/html/rfc2616.

[32] W. Fischer and K. Meier-Hellstern, "The Markov-Modulated Poisson Process (MMPP) Cookbook," *Performance Evaluation*, vol. 18, no. 2, pp. 149–171, September 1993.

[33] J. Fryman, C. Huneycutt, H. Lee, K. Mackenzie, and D. Schimmel, "Energy-Efficient Network Memory for Ubiquitous Devices," *IEEE Micro*, vol. 23, no. 5, pp. 60–70, September/October 2003.

[34] "Gartner Estimates ICT Industry Accounts for 2 Percent of Global $CO_2$ Emissions," Gartner Group Press Release, April 2007. URL: http://www.gartner.com/it/page.jsp?id=503867.

[35] G. Ginis, "Low Power Modes for ADSL2 and ADSL2+," Broadband Communications Group, Texas Instruments, January 2005. URL: http://focus.ti.com/lit/an/spaa021/spaa021.pdf.

[36] C. Gunaratne, K. Christensen, and B. Nordman, "Managing Energy Consumption Costs in Desktop PCs and LAN Switches with Proxying, Split TCP Connections, and Scaling of Link Speed," *International Journal of Network Management*, vol. 15, no. 5, pp. 297–310, September/October 2005.

[37] C. Gunaratne, K. Christensen, and S. Suen, "Ethernet Adaptive Link Rate (ALR): Analysis of a Buffer Threshold Policy," *Proceedings of the IEEE Global Communications Conference (GLOBECOM)*, pp. 448–461, November 2006.

[38] C. Gunaratne, K. Christensen, S. Suen, and B. Nordman, "Reducing the Energy Consumption of Ethernet with an Adaptive Link Rate (ALR)," *IEEE Transactions on Computers*, vol. 57, no. 4, pp. 448–461, April 2008.

[39] M. Gupta, S. Grover, and S. Singh, "A Feasibility Study for Power Management in LAN Switches," *Proceedings of the 12th IEEE ICNP*, pp. 361–371, October 2004.

[40] M. Gupta and S. Singh, "Dynamic Ethernet Link Shutdown for Power Conservation on Ethernet Links," *Proceedings of the IEEE International Conference on Communications*, pp. 6156–6161, June 2007.

[41] M. Gupta and S. Singh, "Greening of the Internet," *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pp. 19–26, August 2003.

[42] M. Gupta and S. Singh, "Using Low-power Modes for Energy Conservation in Ethernet LANs," *Proceedings of the IEEE International Conference on Computer Communications*, pp. 2451–2455, May 2007.

[43] P. Higginson, R. Hinden, P. Hunt, S. Knight, A. Lindem, D. Mitzel, M. Shand, D. Weaver, and D. Whipple, "RFC 2338: Virtual Router Redundancy Protocol," Network Working Group, April 1998.

[44] W. Hang-Sheng, P. Li-Shiuan, and S. Malik, "A Power Model for Routers: Modeling Alpha 21364 and InfiniBand Routers," *Proceeding of the 10th Symposium on High Performance Interconnects*, January/February 2002.

[45] C. R. Hannemann, V. P. Carey, A. J. Shah, and C. Patel, "Lifetime Exergy Consumption of an Enterprise Server," *Proceedings IEEE International Symposium on Electronics and the Environment*, pp. 1–5, May 2008.

[46] W. Heinzelman, "An Application-Specific Protocol Architecture for Wireless Microsensor Networks" *IEEE Transactions on Wireless Communications*, vol. 1, no. 4, pp. 600–670, October 2002.

[47] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan, "Energy-efficient Communication Protocol for Wireless Microsensor Networks," *Proceedings of the Hawaii International Conference on System Sciences*, pp. 10–19, January 2000.

[48] Y. Huang, C. Weber, and H. Matthews, "Carbon Footprinting Upstream Supply Chain for Electronics Manufacturing and Computer Services," *Proceedings of the IEEE International Symposium on Sustainable Systems and Technology*, pp. 1–6, May 2009.

[49] P. Huber and M. Mills, "Dig More Coal –' The PCs Are Coming," *Forbes*, May 1999.

[50] C.-H. Hwang and A. C.-H. Wu, "A Predictive System Shutdown Method for Energy Saving of Event-driven Computation," *Proceedings of IEEE/ACM International Conference on Computer-aided Design*, pp. 226–241, November 1997.

[51] IEEE P802.3az Energy Efficient Ethernet Task Force. URL: http://www.ieee802.org/3/az/index.html.

[52] "IEEE 802.11 Wireless LAN Medium Access Control and Physical Layer Specification," URL: http://standards.ieee.org/getieee802/802.11.html.

[53] "IEEE 802.3x, Standards for Local and Metropolitan Area Networks," 1997.

[54] "Internet Host Count History," Internet Systems Consortium. URL: http://www.isc.org/solutions/survey/history.

[55] L. Irish and K. Christensen, "A "Green TCP/IP" to Reduce Electricity Consumed by Computers," *Proceedings of IEEE Southeastcon*, pp. 302–305, April 1998.

[56] C. E. Jones, K. M. Sivalingam, P. Agrawal, and J. C. Chen, "A Survey of Energy Efficient Network Protocols for Wireless Networks," *Wireless Networks*, vol. 7, no. 4, pp. 343–358, August 2001.

[57] T. Karagiannis, M. Molle, M. Faloutsos, and A. Broido, "A Nonstationary Poisson View of Internet Traffic," *Proceedings of the 23rd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, pp. 1558–1569, March 2004.

[58] R. Kehr, A. Zeidler, and H. Vogt, "Towards a Generic Proxy Execution Service for Small Devices," *Presentation in Future Services For Networked Devices Workshop*, November 1999.

[59] C. Keller, "Global Warming: A Review of this Mostly Settled Issue," *Stochastic Environmental Research and Risk Assessment*, vol. 23, no. 5, pp. 643–643, July 2009.

[60] A. Keshavarzian, H. Lee, and L. Venkatraman, "Wakeup Scheduling in Wireless Sensor Networks," *Proceedings of the ACM International Symposium on Mobile Ad hoc Networking and Computing*, pp. 322–333, May 2006.

[61] KETI (Korea Electronics Technology Institute), 2011. URL: http://www.keti.re.kr.

[62] J. Klamra, M. Olsson, K. Christensen, and B. Nordman, "Design and Implementation of a Power Management Proxy for Universal Plug and Play," *Proceedings of the Swedish National Computer Networking Workshop*, September 2005.

[63] J. Koomey, "Growth in Data Center Electricity Use 2005 to 2010," Analytics Press, August 1, 2011. URL: http://www.analyticspress.com/datacenters.html.

[64] J. Koomey, "Sorry, Wrong Number – How to Separate Fact From Fiction in the Information Age," *IEEE Spectrum*, vol. 40, no. 6, pp. 11–12, June 2003.

[65] S. Lee, J. Yoo, and T. Chung, "Distance-based Energy Efficient Clustering for Wireless Sensor Networks," *Proceedings of the 29th Annual IEEE international Conference on Local Computer Networks (LCN)*, pp. 567–568, November 2004.

[66] W. Leland, M. Taqqu, W. Willinger, and D. Wilson, "On the Self-Similar Nature of Ethernet Traffic," *IEEE/ACM Transactions on Networking*, vol. 2. pp.1–15, February 1994.

[67] S. Lindsey and C. Raghavendra, "PEGASIS: Power-efficient Gathering in Sensor Information Systems," *Proceedings of the IEEE Aerospace Conference*, pp. 1125–1130, March 2002.

[68] J. Lorch and A. Smith, "Software Strategies for Portable Computer Energy Management," *IEEE Personal Communications*, vol. 5, no. 3, pp. 60–73, 1998.

[69] "Magic Packet Technology," Advanced Micro Devices, White Paper, Publication# 20213, Rev: A, Amendment/0, November 1995.

[70] P. Mahadevan, A. Shah, and C. Bash, "Reducing Lifecycle Energy Use of Network Switches," *Proceedings of the IEEE International Symposium on Sustainable Systems and Technology*, pp. 1–6, May 2010.

[71] P. Mahadevan, P. Sharma, S. Banerjee, and P. Ranganathan, "Energy Aware Network Operations," *Proceedings of the IEEE Global Internet Symposium (in conjunction with IEEE INFOCOM)*, April 2009.

[72] D. Meisner, B. Gold, and T. Wenisch, "PowerNap: Eliminating Server Idle Power," *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 205–216, March 2009.

[73] S. Makineni, R. Iyer, P. Sarangam, D. Newell, L. Zhao, R. Illikkal, and J. Moses, "Receive Side Coalescing for Accelerating TCP/IP Processing," *Proceeding of the International Conference on High Performance Computing (HiPC)*, pp. 289–300, December 2006.

[74] M. Mostowfi, "Improving the Energy Efficiency of IEEE 802.3az EEE and Periodically Paused Switched Ethernet," Master's Thesis, Department of Computer Science and Engineering, University of South Florida, July 2010.

[75] M. Mostowfi and K. Christensen, "An Energy-Delay Model for a Packet Coalescer," *Proceedings of the IEEE SoutheastCon*, pp. 1–6, March 2012.

[76] M. Mostowfi and K. Christensen, "Saving Energy in LAN Switches: New Methods of Packet Coalescing for Energy Efficient Ethernet," *Proceedings of the Second International Green Computing Conference (IGCC)*, pp. 1–8, July 2011.

[77] M. Mostowfi, K. Christensen, S. Lee, and J. Yun, "HTTP Timed Redirection to Reduce Energy Use of Servers," under review for *IEEE Internet Computing*, submitted on January 7, 2013.

[78] M. Mostowfi, K. Christensen, S. Lee, and J. Yun, "SME Web Energy Efficient Platform (SWEEP): A New Architecture for a Hybrid Web Server," to appear in the *Journal of Sustainable Computing – Informatics and Systems* (available online in April 2013).

[79] M. Mostowfi, K. Christensen, S. Lee, and J. Yun, "Timed Redirection: HTTP Request Coalescing to Reduce Energy Use of Hybrid Web Servers," *Proceedings of the IEEE Conference on Local Computer Networks (LCN)*, pp. 168–171, October 2012.

[80] R. Myslewski, "Intel dubs 2009 'The year of 10Gb Ethernet'," *The Register*, March 2009.

[81] S. Nedevschi, J. Chandrashenkar, B. Nordman, S. Ratnasamy, and N. Taft, "Skilled in the Art of Being Idle: Reducing Energy Waste in Networked Systems," *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation*, pp. 381–394, April 2009.

[82] S. Nedevschi, L. Popa, G. Iannaccone, S. Ratnasamy, and D. Wetherall, "Reducing Network Energy Consumption via Rate-Adaptation and Sleeping," *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pp. 323–336, April 2008.

[83] G. Neiger, A. Santoni, F. Leung, D. Rodgers, and R. Uhlig, "Intel Virtualization Technology: Hardware Support for Efficient Processor Virtualization," *Intel Technology Journal*, August 2006. URL: http://www.intel.com/technology/itj/2006/v10i3/.

[84] The Network Simulator – ns-2, 2008. URL: http://www.isi.edu/nsnam/ns/.

[85] H. Newman, "Falling 10GbE prices spell doom for fibre channel," EnterpriseStorageForum.com, June 2009.

[86] B. Nordman, Talk in *Energy Efficient Digital Networks PAC Meeting*, California Energy Commission, September 2009.

[87] B. Nordman, "Networks, Energy and Energy Efficiency," Presentation at *Cisco Green Research Symposium*, March 2008.

[88] B. Nordman and K. Christensen, "Greener PCs for the Enterprise," *IEEE IT Professional*, vol. 11, no. 4, pp. 28–37, July/August 2009.

[89] B. Nordman and K. Christensen, "Reducing the Energy Consumption of Network Devices," tutorial, *IEEE 802 LAN/MAN Standards Committee Plenary Session*, July 2005.

[90] A. Odlyzko, "Data Networks are Lightly Utilized, and Will Stay That Way," *Review of Network Economics*, vol. 2, no. 3, pp. 210–237, September 2003.

[91] M. Olson, K. Christensen, S. H. Lee, and J. Yun, "Hybrid Web Server: Traffic Analysis and Prototype," *Proceedings of the 36th IEEE Conference on Local Computer Networks (LCN)*, pp. 131–134, October 2011.

[92] R. Pang, M. Allman, M. Bennett, J. Lee, V. Paxson, and B. Tierney, "A First Look at Modern Enterprise Traffic," *Proceedings of the 5th ACM SIGCOMM Conference on Internet Measurement*, pp. 15–28, October 2005.

[93] V. Paxson, "Some Perspectives on the Performance Impact of Link-speed Switching Outages," Presentation at IEEE 802.3 meeting, July 2007.

[94] "Plug Computer Community," 2011. URL: http://www.plugcomputer.org/.

[95] B. Raghavan and J. Ma, "The Energy and Emergy of the Internet," *Proceedings of the 10th ACM Workshop on Hot Topics in Networks (HotNets)*, November 2011.

[96] P. Reviriego, J. Hernandez, D. Larrabeiti, and J. Maestro, "Performance Evaluation of Energy Efficient Ethernet," *IEEE Communication Letters*, vol. 13, no. 9, pp. 697–699, September 2009.

[97] M. Rodriguez-Perez, S. Herreria-Alonso, M. Fernandez-Veiga, and C. Lopez-Garcia, "Improved Opportunistic Sleeping Algorithms for LAN Switches," *Proceedings of the 28th IEEE Conference on Global Telecommunications*, pp. 35–40, November/December 2009.

[98] H. Ross and D. Murray, "E. H. Weber: The Sense of Touch," New York: Academic Press, 1978.

[99] K. Roth, F. Goldstein, and J. Kleinman, "Energy Consumption by Office and Telecommunications Equipment in Commercial Buildings Volume I: Energy Consumption Baseline," National Technical Information Service (NTIS), US Department of Commerce, 2002.

[100] C. Schurgers, V. Tsiatsis, and M. B. Srivastava, "STEM: Topology Management for Energy Efficient Sensor Networks," *Proceedings of the IEEE Aerospace Conference*, pp. 1099–1108, March 2002.

[101] A. Shah, C. Bash, R. Sharma, T. Christian, B.J. Watson, and C. Patel, "The Environmental Impact of Data Centers," *Proceedings of the ASME International Technical Conference and Exhibition on Packaging and Integration of Electronic and Photonic Microsystems (InterPACK)*, July 2009.

[102] T. Simunic, L. Benini, P. Glynn, and G. De Micheli, "Event-Driven Power Management," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 20, no. 7, pp. 840–857, July 2001.

[103] S. Singh and C. Raghavendra, "PAMAS: Power Aware Multi-Access Protocol with Signaling for Ad Hoc Networks," *Computer Communications Review*, vol. 28, no. 3, pp. 5–26, July 1998.

[104] V. Soteriou, and P. Li-Shiuan, "Dynamic Power Management for Power Optimization of Interconnection Networks Using On/Off Links," *Proceedings of the 11th Symposium on High Performance Interconnects*, pp. 15–20, August 2003.

[105] M. B. Srivastava, A. P. Chandrakasan, and R. W. Brodersen, "Predictive System Shutdown and Other Architectural Techniques for Energy Efficient Programmable Computation," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 4, no. 1, pp. 42–55, March 1996.

[106] M. Stemm and R. Katz, "Measuring and Reducing the Energy Consumption on Network Interfaces in Handheld Devices," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Science*, August 1997.

[107] J. Sugerman, G. Venkitachalam, and B.-H. Lim, "Virtualizing I/O Devices on VMware Workstation's Hosted Virtual Machine Monitor," *Proceedings of the USENIX Annual Technical Conference*, pp. 1–14, June 2001.

[108] H. Tamura, Y. Yahiro, Y. Fukuda, K. Kawahara, and Y. Oie, "Performance Analysis of Energy Saving Scheme with Extra Active Period for LAN Switches," *Proceedings of the IEEE Global Telecommunications Conference (GLOBECOM)*, pp. 198–203, November 2007.

[109] N. Tolia, Z. Wang, M. Marwah, C. Bash, P. Ranganathan, and X. Zhu, "Delivering Energy Proportionality with Non Energy-Proportional Systems – Optimizing the Ensemble," *Proceedings of the Conference on Power Aware Computing and Systems (HotPower)*, December 2008.

[110] R. Tucker, J. Baliga, R. Ayre, K. Hinton, and W. Sorin, "Energy Consumption in IP Networks," *Proceedings of the European Conference and Exhibition on Optical Communication*, September 2008.

[111] M. Tzannes, "ADSL2 Helps Slash Power in Broadband Designs," *EETimes*, January 2003. URL: http://www.eetimes.com/electronics-news/4136967/ADSL2-Helps-Slash-Power-in-Broadband-Designs.

[112] "USA QuickFacts," U.S. Census Bureau. URL: http://quickfacts.census.gov/qfd/states/00000.html.

[113] "Verdiem – The Leader in IT Energy Management and Efficiency," 2011. URL: http://www.verdiem.com.

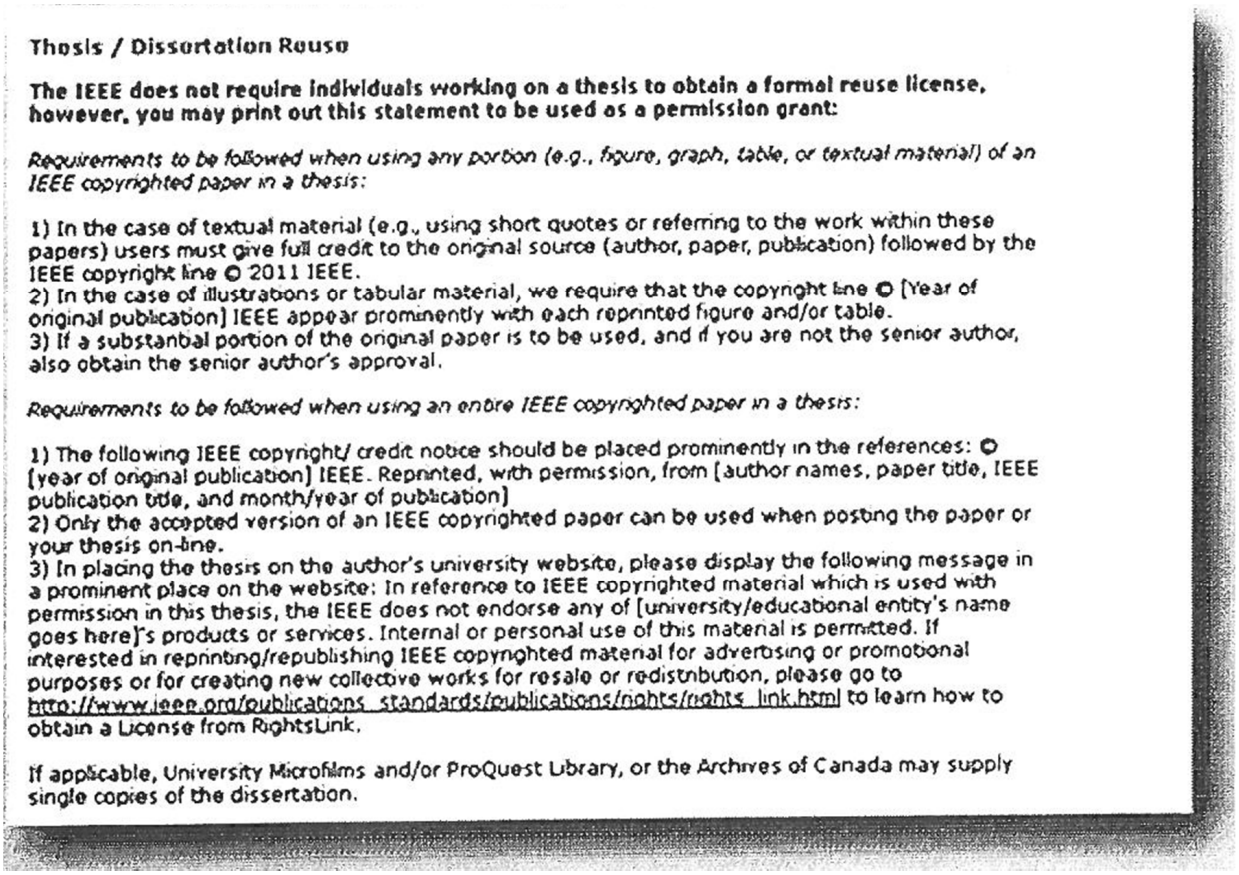
[114] Vitesse Semiconductor Corporation, "SparX-G8™-8-Port Gigabit Ethernet Integrated PHY Switch," VSC7388 Datasheet, March 2006.

[115] E. Williams, "Revisiting Energy Used to Manufacture a Desktop Computer: Hybrid Analysis Combing Process and Economic Input-Output Methods," *Proceedings of IEEE International Symposium on Electronics and the Environment*, pp. 80–85, May 2004.

## Appendices

## Appendix A: Permission for Use of Figures 2.3 and 2.5, and [24], [75], [76], [77], and [79]

Figures 2.3, and 2.5, have appeared in papers or standards published by the Institute of Electrical and Electronics Engineers (IEEE). The contents of Chapter 3 have appeared in [24], [75], and [76], all of which are IEEE publications. The contents of Chapter 5 have appeared in [77] (currently under review) and [79] which are also IEEE publications. The IEEE does not require individuals working on a thesis to obtain a formal reuse license. However, authors are asked to print out the statement in Figure A.1 to be used as a permission grant.



**Thesis / Dissertation Reuse**

**The IEEE does not require individuals working on a thesis to obtain a formal reuse license, however, you may print out this statement to be used as a permission grant:**

*Requirements to be followed when using any portion (e.g., figure, graph, table, or textual material) of an IEEE copyrighted paper in a thesis:*

1) In the case of textual material (e.g., using short quotes or referring to the work within these papers) users must give full credit to the original source (author, paper, publication) followed by the IEEE copyright line © 2011 IEEE.
2) In the case of illustrations or tabular material, we require that the copyright line © [Year of original publication] IEEE appear prominently with each reprinted figure and/or table.
3) If a substantial portion of the original paper is to be used, and if you are not the senior author, also obtain the senior author's approval.

*Requirements to be followed when using an entire IEEE copyrighted paper in a thesis:*

1) The following IEEE copyright/ credit notice should be placed prominently in the references: © [year of original publication] IEEE. Reprinted, with permission, from [author names, paper title, IEEE publication title, and month/year of publication]
2) Only the accepted version of an IEEE copyrighted paper can be used when posting the paper or your thesis on-line.
3) In placing the thesis on the author's university website, please display the following message in a prominent place on the website: In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of [university/educational entity's name goes here]'s products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.

If applicable, University Microfilms and/or ProQuest Library, or the Archives of Canada may supply single copies of the dissertation.

Figure A.1: Permission to Reuse Content from an IEEE Publication in a Dissertation

## Appendix B: Permission for Use of Figure 2.8

Figure 2.8 has appeared in a paper published by Wiley. The use of this figure in this dissertation constitutes fair use since it is a minimal and commercially insignificant portion of an existing work for scholarship purposes. Therefore, it does not require permission as per the publisher's website notice, which is shown in Figure B.1 (available at http://www.wiley.com/legacy/authors/guidelines/stmguides/3frames.htm).

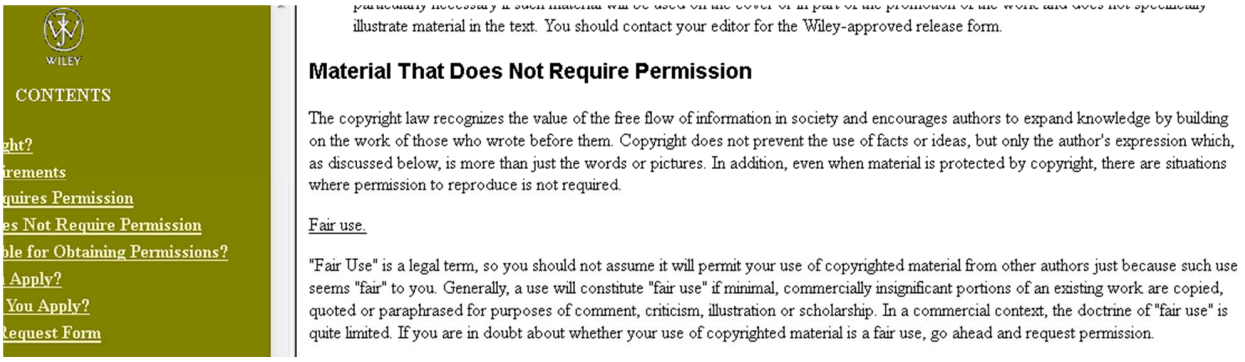
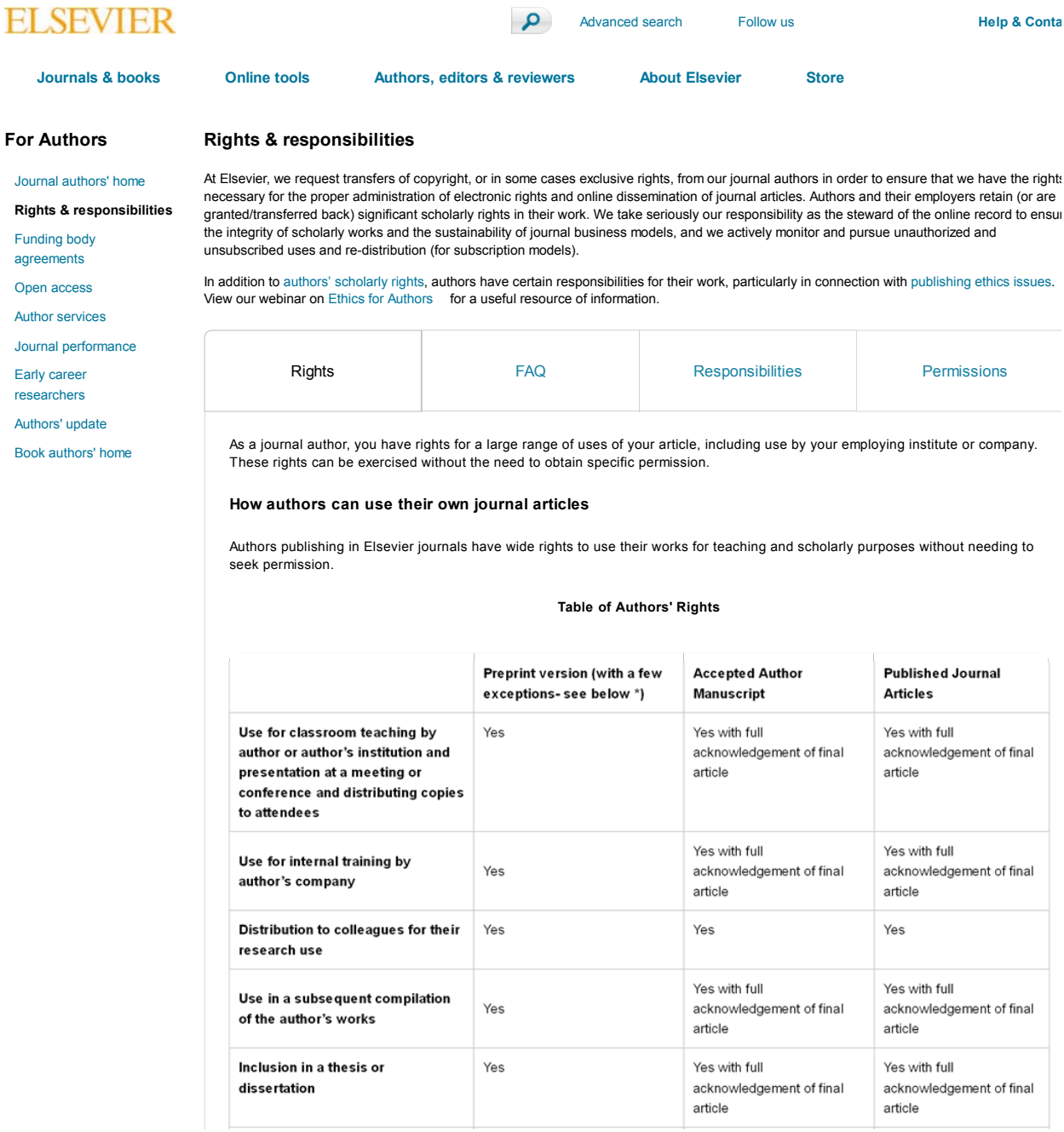



Figure B.1: Permission to Reuse Content from a Wiley Publication in a Dissertation

## Appendix C: Permission for Use of [78]

The contents of Chapter 4 have appeared in [78] which is published by Elsevier. As per the publisher's website notice which is shown in Figure C.1 (available at http://www.elsevier.com/journal-authors/author-rights-and-responsibilities), authors are allowed to use their own publications in their dissertations.



ELSEVIER

Advanced search Follow us Help & Conta

Journals & books Online tools Authors, editors & reviewers About Elsevier Store

**For Authors**

- Journal authors' home
- **Rights & responsibilities**
- Funding body agreements
- Open access
- Author services
- Journal performance
- Early career researchers
- Authors' update
- Book authors' home

**Rights & responsibilities**

At Elsevier, we request transfers of copyright, or in some cases exclusive rights, from our journal authors in order to ensure that we have the rights necessary for the proper administration of electronic rights and online dissemination of journal articles. Authors and their employers retain (or are granted/transferred back) significant scholarly rights in their work. We take seriously our responsibility as the steward of the online record to ensu the integrity of scholarly works and the sustainability of journal business models, and we actively monitor and pursue unauthorized and unsubscribed uses and re-distribution (for subscription models).

In addition to authors' scholarly rights, authors have certain responsibilities for their work, particularly in connection with publishing ethics issues. View our webinar on Ethics for Authors for a useful resource of information.

Rights | FAQ | Responsibilities | Permissions

As a journal author, you have rights for a large range of uses of your article, including use by your employing institute or company. These rights can be exercised without the need to obtain specific permission.

**How authors can use their own journal articles**

Authors publishing in Elsevier journals have wide rights to use their works for teaching and scholarly purposes without needing to seek permission.

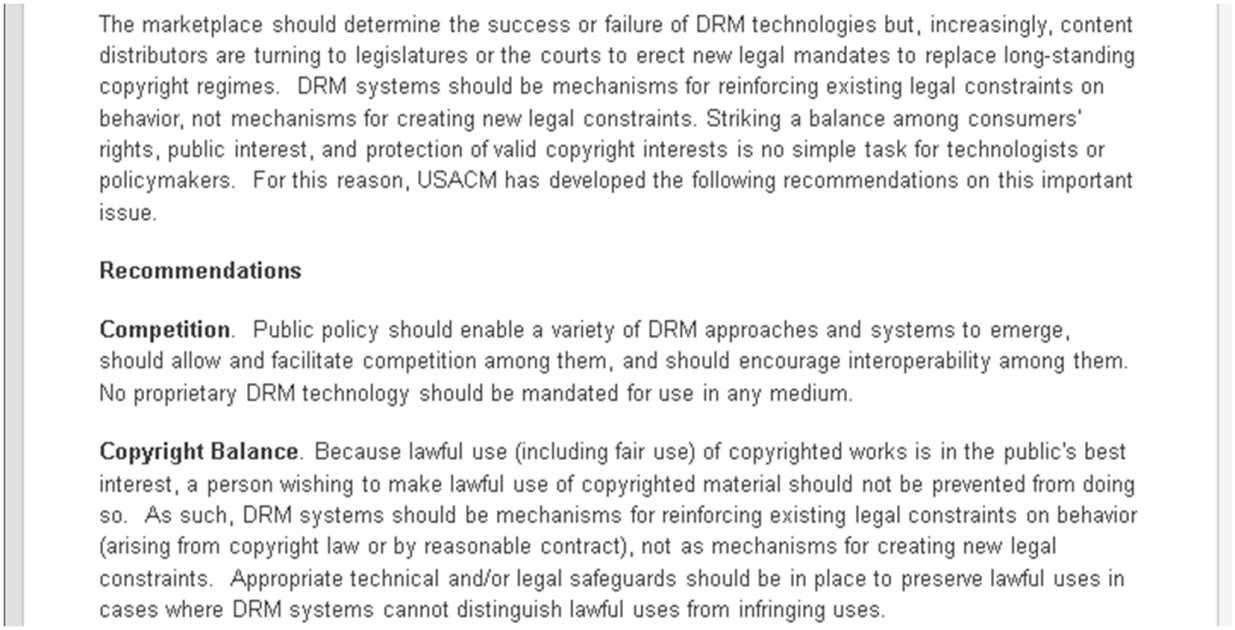
**Table of Authors' Rights**

| | Preprint version (with a few exceptions- see below *) | Accepted Author Manuscript | Published Journal Articles |
|---|---|---|---|
| **Use for classroom teaching by author or author's institution and presentation at a meeting or conference and distributing copies to attendees** | Yes | Yes with full acknowledgement of final article | Yes with full acknowledgement of final article |
| **Use for internal training by author's company** | Yes | Yes with full acknowledgement of final article | Yes with full acknowledgement of final article |
| **Distribution to colleagues for their research use** | Yes | Yes | Yes |
| **Use in a subsequent compilation of the author's works** | Yes | Yes with full acknowledgement of final article | Yes with full acknowledgement of final article |
| **Inclusion in a thesis or dissertation** | Yes | Yes with full acknowledgement of final article | Yes with full acknowledgement of final article |

Figure C.1: Permission to Reuse Content from an Elsevier Publication in a Dissertation

## Appendix D: Permission for Use of Figures 2.9, 2.10, 2.11, and 2.12

Figures 2.9, 2.10, 2.11, and 2.12 have appeared in papers published by the Association of Computing Machinery (ACM). The use of these figures in this dissertation constitutes fair use since minimal and commercially insignificant portions of these papers are used for scholarship purposes. Therefore, it does not require permission as per the publisher's website notice, which is shown in Figure D.1 (available at http://usacm.acm.org/ip/category.cfm?cat=26&Intellectual%20Property).



The marketplace should determine the success or failure of DRM technologies but, increasingly, content distributors are turning to legislatures or the courts to erect new legal mandates to replace long-standing copyright regimes. DRM systems should be mechanisms for reinforcing existing legal constraints on behavior, not mechanisms for creating new legal constraints. Striking a balance among consumers' rights, public interest, and protection of valid copyright interests is no simple task for technologists or policymakers. For this reason, USACM has developed the following recommendations on this important issue.

**Recommendations**

**Competition**. Public policy should enable a variety of DRM approaches and systems to emerge, should allow and facilitate competition among them, and should encourage interoperability among them. No proprietary DRM technology should be mandated for use in any medium.

**Copyright Balance**. Because lawful use (including fair use) of copyrighted works is in the public's best interest, a person wishing to make lawful use of copyrighted material should not be prevented from doing so. As such, DRM systems should be mechanisms for reinforcing existing legal constraints on behavior (arising from copyright law or by reasonable contract), not as mechanisms for creating new legal constraints. Appropriate technical and/or legal safeguards should be in place to preserve lawful uses in cases where DRM systems cannot distinguish lawful uses from infringing uses.

Figure D.1: Permission to Reuse Content from an ACM Publication in a Dissertation

## Appendix E: Glossary of Symbols

(In order of appearance)

| | |
|---|---|
| $E_P$ | Energy Proportionality Index, utilization divided by power |
| $U$ | Utilization of a system as a ratio of the full capacity |
| $P$ | Power consumption of a system as a ratio of the peak power consumption |
| $T_s$ | Transition time from active to sleep mode |
| $T_w$ | Transition time from sleep to active mode |
| $t_{gap}$ | Length of an idle period |
| $t_{sleep}$ | Time in an idle period where a system can effectively sleep |
| $P_a$ | Power consumption of a system in wake (active) state |
| $P_s$ | Power consumption of a system in sleep state |
| $P_{ws}$ | Power consumption of a system in transition from wake to sleep state |
| $P_{sw}$ | Power consumption of a system in transition from sleep to wake state |
| $G$ | Energy savings gained in a system by sleeping in an idle period |
| $\lambda$ | Arrival rate (packets to a coalescer, requests to a hybrid server, etc.) |
| $\mu_c$ | Service rate of a coalescer |
| $\mu_d$ | Service rate of a downstream queue |
| $N$ | Packet count to open coalescer gate |
| $M$ | Total number of packets departing a coalescer in a coalescing cycle |
| $S_G$ | Fraction of total sleep time to total idle time |
| $D_c$ | Mean packet delay for a coalescer |
| $D_d$ | Mean packet delay for a downstream queue |
| $D$ | Mean end-to-end packet delay for a tandem coalescer-downstream queue system |
| $L_c$ | Mean queue length for a coalescer |
| $L_d$ | Mean queue length for a downstream queue |
| $t_{off}$ | time to coalesce $N$ packets in a coalescer |
| $t_{on}$ | In a coalescer, time between beginning of transmission until the end of transmission plus the time until the arrival of the first packet of the next cycle |
| $T_{off}$ | In synchronized coalescing, time for which the switch is off |

## Appendix E: (continued)

| | |
|---|---|
| $T_{on}$ | In synchronized coalescing, time for which the switch is on |
| $C$ | In synchronized coalescing, the fraction of time the switch is on |
| $P_{link}$ | In EEE, power consumption of a link as a percentage of the link's peak power consumption |
| $t_{LPI}$ | In EEE, percentage of times a link spends in LPI mode |
| $t_a$ | In EEE, percentage of times a link spends in Active mode |
| $t_{ws}$ | In EEE, percentage of times a link spends in sleep transition |
| $t_{sw}$ | In EEE, percentage of times a link spends in wakeup transition |
| $T_{coalesce}$ | In EEE with packet coalescing, the maximum time for which packets are collected before sending |
| $T_{epoch}$ | In timed redirection, duration of an epoch consisting of a not serving period and a serving period |
| $T_{notServe}$ | In timed redirection, duration of a not serving period which is predetermined and fixed |
| $T_{serve}$ | In timed redirection, duration of a serving period which is variable |
| $T_{delay}$ | In timed redirection, delay with which a request should be retried, included in Retry-Delay |