Reducing Network Energy Consumption via Rate-Adaptation and Sleeping

Sergiu Nedevschi
Lucian Popa
Gianluca Iannaccone
Sylvia Ratnasamy
David Wetherall

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2007-128

http://www.eecs.berkeley.edu/Pubs/TechRpts/2007/EECS-2007-128.html

October 29, 2007

Reducing Network Energy Consumption via Rate-Adaptation and Sleeping

Sergiu Nedevschi[*] Lucian Popa [*] [†] Gianluca Iannaccone [†] Sylvia Ratnasamy [†]
David Wetherall[‡][§]

Abstract

We present the design and evaluation of two forms of power management schemes that reduce the energy consumption of networks. The first is based on adapting the rate of network operation to the offered workload, reducing the energy consumed when actively processing packets. The second is based on putting network components to sleep during idle times, reducing energy consumed in the absence of packets.

Using real-world network topologies and traffic workloads, we show that: (1) even simple schemes for sleeping or rate-adaptation can offer substantial savings without significantly degrading network performance and (2) both forms of solutions are valuable depending (primarily) on the power profile of network equipment and the utilization of the network itself.

1 Introduction

In this paper, we consider power management for networks from a perspective that has recently begun to receive attention: the conservation of energy for operating and environmental reasons. Energy consumption in network exchanges is rising as higher capacity network equipment becomes more power-hungry and requires greater amounts of cooling. Combined with rising energy costs, this has made the cost of powering network exchanges a substantial and growing fraction of the total cost of ownership – up to half by some estimates[24]. Various studies now estimate the power usage of the US network infrastructure at between 5 and 24 TWh/year[25, 18, 26], or \$0.5-2.4B/year at a rate of \$0.10/KWh, depending on what is included. Public concern about carbon footprints is also rising, and stands to affect network equipment much as it has computers via standards such as EnergyStar. In fact, the EnergyStar standards for 2009 state that network links shall operate more slowly to conserve energy when idle. A new IEEE working group was launched in early 2007 to focus on this issue for Ethernet [14].

Fortunately, there is an opportunity for substantial reductions in the energy consumption of existing networks due to two factors. First, networks are provisioned for worst-case or busy-hour load, and this load typically exceeds their long-term utilization by a wide margin. For example, measurements reveal backbone utilizations under 30% [15] and up to hour-long idle times at access points in enterprise wireless networks [16]. Second, the energy consumption of network equipment remains substantial even when the network is idle. The implication of these factors is that *most* of the energy consumed in networks is wasted.

Our work is an initial exploration of how overall network energy consumption might be reduced without adversely affecting network performance. This will require two steps. First, network equipment ranging from routers to switches and NICs will need power management primitives at the hardware level. By analogy, power management in computers has evolved around hardware support for *performance* and *sleep* states. The former (*e.g.,*SpeedStep, P-states in Intel processors) tradeoff performance for power (via operating frequency) while the latter (*e.g.,*C-states in Intel processors) reduce idle consumption by powering off sub-components to different extents. Second, network protocols will need to make use of the hardware primitives to best effect. Again, by analogy with computers, preferences for power management control how the system moves between the available states to save energy with minimal impact on users.

Of these two steps, our focus is on the network protocols. Admittedly, these protocols build on hardware support for power management that is in its infancy for networking equipment. Yet the necessary support will readily be deployed in networks where it proves valuable, with forms such as rapid rate selection for Ethernet [14] already under development. For comparison, computer power management compatible with the ACPI standard has gone from scarce to widely deployed over the past five to ten years and is now expanding into the server market. Thus our goal is to learn: what magnitude of energy savings a protocol using feasible hardware primitives might offer; what performance tradeoff comes with these savings; and which of the feasible kinds of hardware primitives would maximize benefits. We hope that our research can positively influence the hardware support offered by industry.

The hardware support we assume from network equip-

[*]University of California, Berkeley
[†]Intel Research, Berkeley
[‡]University of Washingon
[§]Intel Research, Seattle

ment is in the form of performance and sleep states. Performance states help to save power when routers are active, while sleep states help to save power when routers are idle. The performance states we assume dynamically change the rate of links and their associated interfaces. While such variable speed links would have seemed unlikely even in the recent past, efforts such as the recently formed IEEE 802.3 Study Group on rapid rate selection make links with at least some rate-adaptation capabilities (10/100/1000Mbps) a likely outcome. The sleep states we assume in this paper quickly power off network interfaces when they are idle. Given the proliferation of lightweight sleep states in computers (some of which can be reached in a small number of cycles) there appears to be little reason that similar states could not be designed into network hardware.

We develop two approaches to save energy with these primitives. The first adapts the rate of individual links based on the utilization and queuing delay of the link. We go beyond recent work with our attention to rate transitions that minimize added delay and larger sets of rates. The second approach puts network interfaces to sleep during short idle periods. To make this effective we introduce small amounts of buffering, much as 802.11 APs do for sleeping clients; this collects packets into small bursts and thereby creates gaps long enough to profitably sleep. Potential concerns are that buffering will add too much delay across the network and that bursts will exacerbate loss. We coordinate sleep cycles across routers and switches so that the buffering delay penalty is paid only once (not per link) and clear bursts so that we do not amplify loss noticeably. The result is a novel scheme that differs from 802.11 schemes in that all network elements are able to sleep when they are not utilized yet added delay is bounded.

We then evaluate these approaches using real-world network topologies and traffic workloads from Abilene and Intel. We find that: (1) rate-adaptation and sleeping have the potential to deliver substantial energy savings for typical networks; (2) the simple schemes we develop are able to capture most of this energy-saving potential; (3) our schemes do not noticeably degrade network performance; and (4) both rate-adaptation and sleeping are valuable depending primarily on the utilization of the network.

The rest of this paper is organized as follows. We describe our approach in more detail in Section 2, including a power model and our methodology. In Section 3, we develop and evaluate rate adaptation designs. In Section 4, we develop and evaluate sleeping designs. We compare these alternatives in Section 5. We present related work in Section 6 and conclude in Section 7.

2 Approach

This section describes the high-level model for power consumption that motivates our rate adaptation and sleeping solutions, as well as the methodology by which we evaluate these solutions.

2.1 Power Model

Static vs. dynamic power and active vs. idle times At a high level, the power consumption of a system has two components: a *dynamic* component related to the actual processing of work, and a static component that is always present and (mostly) independent of workload. For example, in the context of a single transistor, dynamic consumption is due to the charging/discharging of load capacitance when a circuit is switching, while the static component is caused by leakage currents present even when a circuit is not switching. Thus a processing element sees only the static component of power when idle but both static and dynamic components when active. Hence to a first approximation, we can consider the total energy consumption of a network element as:

$$E = (p_d + p_s)T_{active} + p_s T_{idle} \tag{1}$$

where p_d and p_s are the dynamic and static components of power respectively, and T_{active} and T_{idle} $(= T - T_{active})$ the active and idle times respectively.

Reducing power through sleep and performance states We assume hardware support for sleep and performance states in network equipment, much as is already deployed in computers. Sleep states lower power consumption by putting subcomponents of the overall system to sleep when there is no work to process. Thus sleeping reduces the *static* power when *idle*, *i.e.*,it reduces the $p_s T_{idle}$ term of Equation 1 by reducing the p_s to some sleep-mode power draw p_{sleep} where $p_{sleep} < p_s$.

Performance states lower power consumption by slowing the rate at which work is processed. As we elaborate later in the paper, the dynamic component of power p_d depends on the frequency and voltage at which work is processed. By scaling the frequency and/or voltage, performance states reduce the *dynamic* component of power when *active*.[1] That is, they reduce the $p_d T_{active}$ term in Equation 1.

We also assume a penalty for transitioning between power states. For simplicity, we measure this penalty in time, typically milliseconds, treating it as a period in which the router can do no useful work. Thus there is also a cost for switching between states.

[1]In practice, dynamic voltage scaling can affect the static draw of some components. However, to a first approximation it is reasonable to treat these components as independent.

Networks with rate adaptation and sleeping support We aim to devise network schemes for rate adaptation and sleeping that use the underlying sleep and performance states to save power without sacrificing performance. To do so, network elements need to coordinate their power management states to some extent. This is because the sleeping and rate adaptation decisions one router makes fundamentally impacts – and is impacted by – the decisions its neighboring routers make. Moreover, as we see later in the paper, the strategies by which each is best exploited are very different.[2] Hence, to avoid complex interactions, we consider that the whole network, or at least well-defined components of it, run in either rate adaptation or sleep mode.

We develop the specifics of our rate adaptation schemes in Section 3, and our sleeping schemes in Section 4. Note that our solutions are deliberately constructed to apply *broadly* to the networking infrastructure – from end-host NICs, to switches, and IP routers, etc. – so that they may be applied wherever they prove to be most valuable. They are not tied to IP-layer protocols.

2.2 Methodology

We evaluate rate-adaptation solutions in terms of their ability to reduce *dynamic* power consumption (Section 3) and sleep solutions in terms of the savings they offer in *idle-time* consumption (Section 4). In this way we assess each solution with an appropriate baseline. We then place each savings in the context of the overall power consumption of a network element to compare the relative merits of sleeping and rate-adaptation (Section 5).

It is difficult to meaningfully evaluate prototype implementations of our solutions in the absence of network equipment with hardware support for power management. Thus we base our evaluations on packet-level simulation with real-world network topologies and traffic workloads. The key factors on which power savings then depend, beyond the details of the solutions themselves, are the technology constants of the sleep and performance states and the characteristics of the network. In particular, the utilization of links determines the relative magnitudes of T_{active} and T_{idle} as well as the opportunities for profitably exploiting sleep and performance states. We give simple models for technology constants in the following sections. To capture the effect of the network on power savings, we drive our simulation with two realistic network topologies and traffic workloads (Abilene and Intel) that are summarized below. We use *ns2* as our packet-level simulator.

[2]Intuitively this is because sleep-mode savings are best exploited by maximizing idle times (which implies processing work as quickly as possible) while performance-scaling is best exploited by processing work as slowly as possible which reduces idle times.

Abilene We use Abilene as a test case because of the ready availability of detailed topology and traffic information. The information from [27] provides us with the link connectivity, weights (to compute routes), latencies and capacities for Abilene's router-level topology. We use measured Abilene traffic matrices (TMs) available in the community [30] to generate realistic workloads over this topology. Unless otherwise stated, we use as our default a traffic matrix whose link utilization levels reflect the average link utilization over the entire day – this corresponds to a 5% link utilization on average with bottleneck links experiencing about 15% utilization.

We also present results from TMs corresponding to the lowest and highest utilization level in a 24-hour period and linearly scale TMs to study performance with increasing utilization. We study performance up to a maximum average network utilization of 31% (as beyond this some links reach very high utilizations). Finally, while the TMs specify the 5-minute average rate observed for each ingress-egress pair, we still require a packet-level traffic generation model that creates this rate. In keeping with previous studies [17, 32] we generate traffic as a mix of Pareto flows, and for some results we use constant bit-rate (CBR) traffic. As per standard practice, we set router queue sizes equal to the bandwidth-delay product in the network; we use the bandwidth of the bottleneck link, and a delay of 100ms.

Intel As an additional real-world dataset, we collected topology and traffic information for the global Intel enterprise network. This network connects Intel sites worldwide, from small remote offices to large multi-building sites with thousands of users. It comprises approximately 300 routers and over 600 links with capacities ranging from 1.5Mbps to 1Gbps.

To obtain the topology, we use a network crawler that periodically queries routers for SNMP data. Using this data, we recreate the topology and the routing tables.

To simulate realistic traffic, we collected unsampled Netflow records [6] from the core routers. The records, exported by each router every minute, contain per flow information including: the communication endpoints, protocol type and port numbers, timestamps of the first and last packet, input and output interfaces, number of packets and total byte size. From these we recreated the traffic at the ingress nodes that we use in our simulations.

3 Rate-Adaptation in Networks

This section explores the use of performance states to reduce network energy consumption.

3.1 Model and Assumptions

Background In general, operating a device at a lower frequency can enable dramatic reductions in dynamic energy consumption for two reasons. First, simply operating more slowly offers some fairly substantial savings.

For example, Ethernet links dissipate between 2-4W when operating between 100Mbps-1Gbps compared to 10-20W between 10-100Gbps[10]. Second, operating at a lower frequency allows the use of dynamic voltage scaling (DVS) that reduces the operating voltage; energy scales quadratically with voltage[33] and hence DVS can offer significant savings. DVS and frequency scaling are already common in microprocessors for these reasons.

We assume the application of these techniques to network links and associated equipment (i.e., linecards, transceivers). While not yet widely available, variable-speed links that adjust the operating voltage to track changes in link frequency have been demonstrated [19, 28] as has the use of DVS in router linecards [22].

Model We assume individual links can switch performance states independently and with independent rates for transmission and reception on interfaces.[3] Hence the savings we obtain apply directly to the consumption at the links and interface cards of a network element, although in practice one could also scale the rate of operation of the switch fabric and/or route processor.

We assume that each network interface supports N performance states corresponding to link rates $r_1, \ldots, r_n$ (with $r_i < r_{i+1}$ and $r_n = r_{max}$, the default maximum link rate) and that the dynamic energy consumption of an interface scales quadratically with the link rate.[4]

In practice, there is a minimum rate threshold below which scaling the link rate offers no further reduction in voltage. We thus define a maximum scale factor λ and set $r_1 = r_n/\lambda$. While current transistor technology allows scaling up to factors as high as 5[33], current processors typically use $\lambda \sim 2$. Thus we use both values as potential rate scaling limits and assume the N available rates to be uniformly distributed in the range $[\frac{r_{max}}{\lambda}, r_{max}]$.

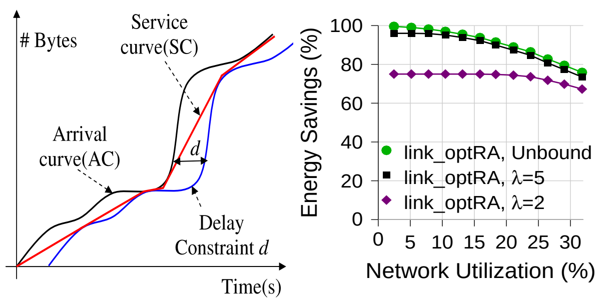
The final defining characteristic of performance states is the transition time, denoted δ, during which packet transmission is stalled as the link transitions between successive rates.[5] We explore performance for a range of transition times (δ) from 0.1 to 10 milliseconds.

Measuring savings and performance In network environments where packet arrival rates can be highly non-uniform, the combination of slower packet processing and transition times can introduce additional packet delay, or even loss, that would not have otherwise occurred.

[3]Already common today – *e.g.,*GPoN access links already operate with different upstream and downstream rates.

[4]In more detail, this is because the processing frequency of components on the link's interface card can be scaled with the link rate and, with the application of DVS, this implies the dynamic energy consumption on the interface scales as r_i^2.

[5]We use this as a simple switching model that lumps all penalties, ignoring other effects that may be associated with switches such as a transient increase in power consumption.



(a) Service Curve Example (b) Dynamic Energy Savings

Figure 1: An illustration of delay-constrained service curves with `link_optRA` and corresponding savings in dynamic energy consumption.

Our goal is explore solutions that usefully navigate the tradeoff between energy savings and performance.

In this section, we measure savings as the percentage reduction in *dynamic* power consumption relative to always processing work at the highest rate r_{max}. We relate these savings to the overall (static and dynamic) power consumption and savings in Section 5. In terms of performance, we measure the average and 98th percentile of the end-to-end packet delay and packet loss.

3.2 Potential Savings

Our initial interest is to understand the extent to which performance states can help if used to best effect. For a DVS processor, it has been shown that the most energy-efficient way to execute C cycles within a given time interval T is to maintain a constant clock speed of C/T [21]. In the context of a network link, this translates into sending packets at a constant rate equal to the average arrival rate. However under non-uniform traffic this can result in arbitrary delays and hence we instead look for an optimal *schedule* of rates (*i.e.,*the set of rates at which the link should operate at different points in time) that minimize energy while respecting a specified constraint on the additional delay incurred at the link.

More precisely, given a packet arrival curve AC, we look for a service curve SC that minimizes energy consumption, while respecting a given upper bound d on the per-packet queuing delay. The delay parameter d thus serves to tradeoff savings for increased delay. Fig.1(a) shows an example arrival curve and the associated latest-departure curve (AC+d), which is simply the arrival curve shifted in time by the delay bound d. To meet the delay constraint, the service curve SC must lie within the area between the arrival and latest-departure curves.

In the context of wireless links, [20] proves that if the energy can be expressed as a convex, monotonically increasing function of the transmission rate, then the minimal energy service curve is the *shortest Euclidean distance* in the arrival space (bytes $\times$ time) between the arrival and shifted arrival curves. Since the energy function for a DVS link is also a convex, monotonically increasing function of link rate, this property holds in our context as well. Fig. 1(a) illustrates an example of such a minimal energy service curve. Intuitively, this is the set of lowest constant rates obeying the delay constraint.

Note that this per-link optimal strategy is *not* suited to practical implementation since it assumes perfect knowledge of the future arrival curve, link rates of infinite granularity and ignores switching overheads. Nonetheless, it is useful as an estimate of the potential savings enabled by performance states and hence to calibrate practical protocols.

We evaluate the savings achieved by applying the above per-link solution at all links in the network and call this approach `link_optRA`. One issue in doing so is that the service curves at the different links are inter-dependent – *i.e.*,the service curve for a link l depends in turn on the service curves at other links (since the latter in turn determine the arrival curve at l). We address this by applying the per-link optimal algorithm iteratively across all links until the service and arrival curves at the different links converge.

We estimate the energy savings with the resulting link rates using a numerical simulator and the Abilene network scenario (Section 2.2). We evaluate the potential savings in three situations:

- an ideal scenario in which a link can operate at any rate up to r_{max}
- an optimistic scenario where $\lambda = 5$ and $N = 10$
- a realistic scenario where $\lambda = 2$ and $N = 10$

Figure 1(b) plots the corresponding savings for increasing network utilization. We see that even with realistic traffic workloads rate adaptation can save a very significant fraction of the network's dynamic energy consumption – typically well above 70% of the total dynamic consumption. Savings are affected by the maximum scale factor λ (*e.g.*,savings saturate at 75% for $\lambda = 2$) and decrease with network utilizations although savings remain high ($> 70\%$) for all λ and practical utilization levels. Finally, we note that even for limited rate-scaling support (λ =2), significant savings are possible.

3.3 A practical algorithm

Building on the insight offered by the per-link optimal algorithm, we develop a simple approach, called `practRA` (practical rate adaptation), that seeks to navigate the tradeoff between savings and delay constraints. A practical approach differs from the optimum in that a) it does not have knowledge of future packet arrivals, b) it can only choose among a fixed set of available rates $r_1, \ldots, r_n$, and c) at every rate switch, it incurs a switching penalty δ, during which it cannot send packets.

While knowledge of the future arrival rate is unavailable, we can use the recent history of packet arrivals to predict the rate of future arrivals. We denote this predicted arrival rate as $\hat{r}_f$ and estimate it with an exponentially weighted moving average (EWMA) of the measured history of past arrivals. Similarly, we can use the current link buffer size q and rate r_i to estimate the potential queuing delay so as to avoid violating the delay constraint.

With these substitutes, we define a technique inspired by the per-link optimal algorithm. In `practRA`, packets are serviced at a constant rate until we intersect one of the two bounding curves presented earlier (Figure 1(a)): the arrival curve (AC), and the latest-departure curve (AC+d). Thus, we avoid increasing the operating rate r_i unless not doing so would violate the delay constraint. This leads to the following condition for rate increases:

A link operating at rate r_i with current queue size q increases its rate to r_{i+1} iff ($\frac{q}{r_i} > d$ OR $\frac{\delta \hat{r}_f + q}{r_{i+1}} > d - \delta$)

The first term checks whether the delay bound d would be violated were we to maintain the current link rate. The second constraint ensures that the service curve does not get too close to the delay-constrained curve which would prevent us from attempting a rate increase in the future without violating the delay bound. That is, we need to allow enough time for a link that increases its rate to subsequently process packets that arrived during the transition time (estimated by $\delta \cdot \hat{r}_f$) and its already-accumulated queue. Note that we cannot use delay constraints d smaller than the transition time δ. Similarly, the condition under which we allow a rate decrease is as follows:

A link operating at rate r_i with current queue size q decreases its rate to r_{i-1} iff $q = 0$ AND $\hat{r}_f < r_{i-1}$

First, we only attempt to switch to a lower rate when the queue at the link is empty ($q = 0$). Intuitively, this corresponds to an intersection between the arrival curve and the service curve. However, we don't always switch to a lower rate r_{i-1} when a queue empties as doing so would prevent the algorithm from operating in a (desired) steady state mode with zero queuing delay. Instead, the desirable steady state is one where $r_i > r_f > r_{i+1}$ and we want to avoid oscillating between rates (which would lead to larger average delay). For example, a link that sees a constant arrival rate of 3.5Mbps might oscillate between 3 and 4Mbps (incurring queuing delays at 3Mbps), instead of remaining at 4Mbps (with low average queuing delay). We thus use the additional condition: $\hat{r}_f < r_{i-1}$ to steer our algorithm toward the desired steady state and ensure that switching to a lower rate does not immediately lead to larger queues.

In addition to the above conditions, we further discourage oscillations by enforcing a minimum time $K\delta$

between consecutive switches. Intuitively, this is because rate switching should not occur on timescales smaller than the transition time δ. In our experiments, we found $K = 4$ to be a reasonable value for this parameter.

Note that unlike the `link_optRA` algorithm, the above decision process does not guarantee that the delay constraints will be met since it is based on estimated rather than true arrival rates. Similarly, `practRA` cannot guarantee that the rates used by the links match those used by the link-optimal algorithm. In what follows, we use simulation to compare our approximate algorithm to the optimal under realistic network conditions. We leave it to future work to analytically bound the inaccuracy due to our approximations.

Finally, we observe that the above rate-adaptation is simple in that it requires no coordination across different nodes, and amenable to implementation even in high-speed equipment. This is because the above computations and decision making need not be performed on a per-packet basis.

3.4 Evaluation

We evaluate the savings *vs.* performance tradeoff achieved by our practical rate-adaptation algorithm and the impact of equipment and network parameters on the same.

Savings *vs.* performance. We first evaluate `practRA` running on the Abilene scenario. We measure savings in terms of the percentage reduction in *dynamic* power consumption of `practRA`, and compare this to the savings achieved by the per-link optimal algorithm `link_optRA`. To measure performance, we use the average and 98th percentile of the end-to-end packet delay and packet loss. We compare performance using `practRA` to that in a network with no rate adaptation, as this shows the overall performance penalty due to our rate-adaptation protocol.

For the Abilene setup, Figure 2 plots the savings under increasing utilization for different values of λ and a delay constraint $d = \delta$ + 2ms. Here, `practRA` uses a transition time δ =1ms. We see that our simple algorithm approaches the savings achievable by `link_optRA`. At $\lambda = 2$, the algorithms are effectively equivalent for most utilizations. This is because the lowest rate available to a link is $\frac{r_{max}}{2}$ and as the utilization on links typically remains well below 50%, both algorithms converge to running most links at a constant rate of $\frac{r_{max}}{2}$.

Figure 3 plots the corresponding average and 98th percentile of the end-to-end delay. We see that the increase in average delay due to `practRA` is very small: $\sim$ 3ms in the worst case. The increase in maximum delay is also reasonable: 82ms with `practRA` relative to 75ms with no adaptation. As utilization increases, the growth in both average and maximum delays tracks that

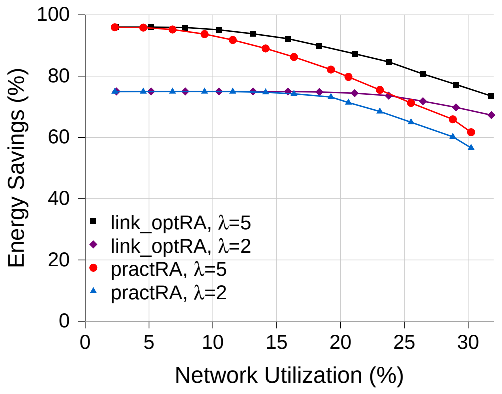
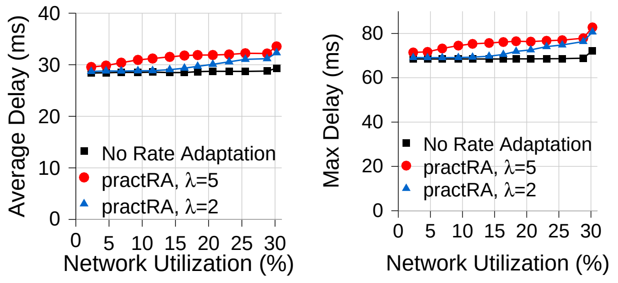


Figure 2: Savings in dynamic power consumption using `link_optRA` *vs.* `practiRA` algorithms.



(a) Average Delay (b) Max Delay (98th percentile)

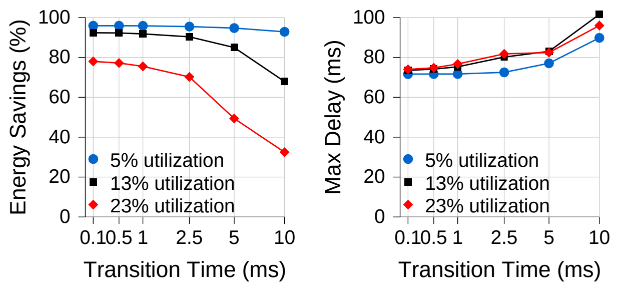
Figure 3: The impact on delay of `link_optRA`.

without rate adaptation. Again, the increase in maximum delay for λ =2 is lower at low utilizations as most links operate at the relatively high rate of $\frac{r_{max}}{2}$. Finally, we found that `practRA` introduced no additional packet loss for the range of utilizations considered.

In summary, these results suggest that rate adaptation as implemented by `practRA` offers significant energy savings of close to `link_optRA` with little impact on packet loss or delay. In all our tests, we found that `practRA` to have minimal effects on the average delay and loss and hence, from here on, we measure the performance impact only in terms of the 98th percentile in packet delay.

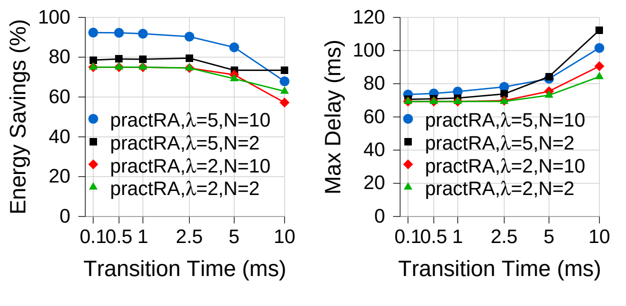
Impact of hardware characteristics We now evaluate how the technology constants of performance states affect `practRA`. The three hardware parameters for performance states are (1) λ, the maximum scale factor;(2) δ, the time to transition between successive rates and (3) N, the granularity of available rates.

The impact of λ was seen in Figure 2 and 3. We look next at the impact of transition times δ. Figures 4 (a) and (b) plot the energy savings and 98th percentile of delay under increasing δ for different network utilizations. For each test, we set the delay constraint as $d = \delta + 2$ms. As would be expected, we see that larger δ lead to reduced



(a) Dynamic Energy Savings (b) Max Delay (98th percentile)

Figure 4: Switch Time Impact with Utilization for `practRA`



(a) Dynamic Energy Savings (b) Max Delay (98th percentile)

Figure 5: Switch Time Impact with λ and the number of rates for `practRA`

savings and higher delay. Moreover, this decay is more rapid at higher utilizations because the absolute difference between savings at adjacent rates is larger at high rates, as a consequence of the quadratic energy function.

On the whole we see that, in this scenario, both savings and performance remain attractive for transition times as high as $\sim$ 2ms.

Figure 5 (a) and (b) present the dynamic energy savings (a) and maximum delay (b) for different N (the number of rates) and scale factor λ at a network utilization of 12%. Comparing the improvement with higher N for $\lambda = 5$ to $\lambda = 2$, we see that a larger number of rates is mostly useful for higher λ as this offers a wide operating range of link rates. Interesting, we see that even with higher λ (=5) as the transition time δ grows, the benefit due to larger N drops and at high values of δ =10ms, having fewer rates is actually preferable. This is because at higher δ rate switches are less frequent and with a larger number of rates it can take a longer time to ramp up/down to the desired operating rate because `practRA` only steps through rates in sequence. This suggests that depending on δ and N, it might be useful to allow out-of-sequence rate transitions.

Impact of network topology and traffic We now evaluate `practRA` applied to the Intel enterprise network. The routing structure of the Intel network is strictly hierarchical with a relatively small number of nodes that

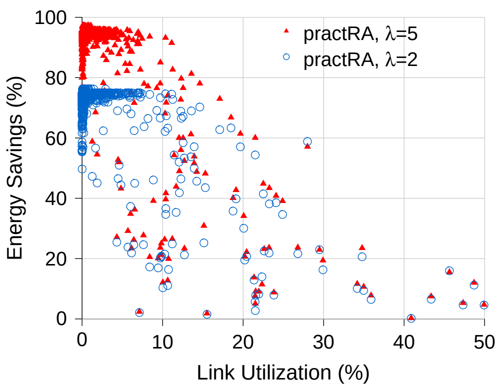


Figure 6: Dynamic Energy Savings per link

connect to the wide-area. Because of this we find a wide variation in link utilization – far more than on the Abilene network. Over 77% of links have utilizations below 1% while a small number of links (2.5%) can see significantly higher utilizations of between 20-75%. The overall network utilization varies between 1.5% and 3.5%.

The resultant energy savings also varies greatly across links. This is shown in Figure 6 that plots the savings per link – each data point in the scatter-plot shows the savings and utilization for a single link, the network wide utilization being on average 2.5%. We see that the dominant trend in savings *vs.* utilization remains similar to that seen with the Abilene network. The majority of links see significant savings (over 90% for $\lambda = 5$). The network-wide savings in dynamic power range between 85-92% (for λ =5) and 65-72% (for λ =2) for the different time periods we tested. The average delay was increased by $\sim$10% for λ =5 and by $\sim$6% for λ =2.

Figure 7 shows the impact of traffic burstiness on the Abilene network. We compare the savings `practRA` achieves for CBR traffic to that with Pareto traffic (as before). As expected, a smoother arrival rate increases savings and brings the performance of `practRA` close to that of `link_optRA`.

4 Putting Network Elements to Sleep

We now focus on power management algorithms that exploit sleep states to reduce power consumption during idle times.

4.1 Model and Assumptions

Background A well established technique, as used by microprocessors and mobiles, is to reduce idle power by putting hardware subcomponents to sleep. For example, modern Intel processors such as the Core Duo [1] have a succession of sleep states (called C-states) that offer increasingly reduced power at the cost of increasingly high latencies to enter and exit these states. We assume similar sleep states made available for network equip-

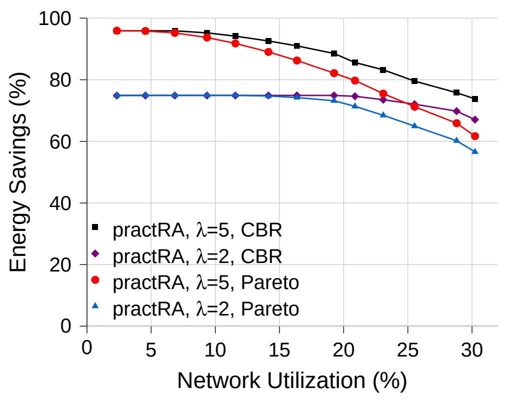


Figure 7: Impact of traffic burstiness

ment. For the purpose of this study we ignore the options afforded by multiple sleep states and assume as an initial simplification that we have a single sleep state.

Model We model a network sleep state as characterized by three features or parameters. The first is the power draw in sleep mode p_{sleep} which we assume to be a small fraction β of the static mode power draw p_s. For example, this factor is about 0.001 for RFM radios [11], 0.3 for PC cards [11], less than 0.1 for DRAM memory [7], and so on.

The second characterizing parameter of a sleep state is the time δ it takes to transition in and out of sleep states. High values of δ raise the bar on when the network element can profitably enter sleep mode and hence δ critically affects potential savings. Because few network equipment vendors support sleep modes, it is hard to estimate realistic values of δ, and estimates in the literature have ranged from as low as 10μsecs(in [12] for router linecards) to as high as 1ms and 10ms, these latter derived by analogy to wireless interfaces [11]. We thus evaluate our solutions over a wide range values of transition times.

Finally, network equipment must support a mechanism for invoking and exiting sleep states. The simplest option for this is *timer-driven sleeping*, in which the network element enters and exits sleep at well-defined times. Prior to entering sleep the network element specifies the time in the future at which it will exit sleep and all packets that arrive at a sleeping interface are lost. Support for timer-driven sleep appears likely as there are already commercial network interfaces from Broadcom and Intel with the ability to shut down transceivers for a period controlled by a programmable timer. The second possibility, described in [12], is for routers to wake up automatically on sensing incoming traffic on their input ports. To achieve this *”wake-on-arrival”* (WoA), the circuitry that senses packets on a line is left powered on even in sleep mode. While support for WoA is not common in either computers or interfaces today, this is a special-purpose form of hardware support that might prove desirable for network equipment. Note that even with wake-on-arrival, bits arriving during the transition period δ are effectively lost. To handle this, the authors in [6] propose the use of “dummy” packets to rouse a sleeping neighbor. A node A that wishes to wake B first sends B a dummy packet, and then waits for time δ before transmitting the actual data traffic.

Measuring savings and performance As mentioned earlier, the power consumption of a network when idle is p_sT_{idle} and a network element that sleeps for a portion T_{sleep} of the idle time will see a reduced idle-time consumption given by:

$$p_{sleep}T_{sleep} + p_s(T_{idle} - T_{sleep})$$

where $p_{sleep} = \beta p_s$. As a simplifying assumption, we first evaluate savings with $\beta = p_{sleep} = 0$ in which case savings are effectively the percentage of idle time spent in sleep and then look at savings for different β.

Finally, we assume individual line cards in a network element can be independently put to sleep. This allows for more opportunities to sleep than if one were to require that a router sleep in its entirety (as the latter is only possible when there is no incoming traffic at any of the incoming interfaces). Correspondingly our energy savings are with respect to interface cards which typically represent a major portion of the overall consumption of a network device. That said, one could in addition put the route processor and switch fabric to sleep at times when all line cards are asleep.

4.2 Approaches and Potential savings

For interfaces that support wake-on-arrival, one approach to exploiting sleep states is that of *opportunistic sleeping* in which link interfaces sleep when idle – *i.e.*,a router is awakened by an incoming (dummy) packet and, after forwarding it on, returns to sleep if no subsequent packet arrives for some time. While very simple, there are two problems with this approach. The first is that it requires the more sophisticated hardware support of wake-on-arrival. The second problem is this approach results in frequent transitions which limits savings for higher transition times. Thus while opportunistic sleeping might be effective in LANs [11, 23] with high idle times, for fast links this potential appears small. For example, with a 10Gbps link, even under low utilization (5%) and packet sizes of 1KB, the average packet inter-arrival time is very small – 15μs.

Instead we consider a novel approach that allows us to explicitly control the tradeoff between network performance and energy savings. Our approach is to shape traffic into small bursts at the edges of the network – edge devices then transmit packets in bunches and routers

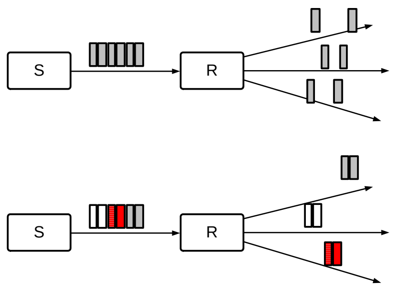


Figure 8: Packets within a burst are organized by destination egress.

within the network wake up to process a burst of packets, and then sleep until the next burst arrives. The intent is to provide sufficient bunching to create opportunities for sleep if the load is low, yet not add excessive delay.[6] More precisely, we introduce a buffer interval "B" that controls the tradeoff between savings and performance. An ingress router buffers incoming traffic for up to B ms and, once every B ms, forwards buffered traffic in a burst.

To ensure that bursts created at the ingress are retained as they traverse through the network, an ingress router arranges packets within the burst such that all packets destined for the same egress router are contiguous within the burst (see figure 8).[7]

The above "buffer-and-burst" approach (B&B) creates alternating periods of contiguous activity and sleep leading to fewer transitions and amortizing the transition penalty δ over multiple packets. This improvement comes at the cost of an added end-to-end delay of up to B ms. Note that because only ingress routers buffer traffic, the additional delay due to buffering is only incurred once along the entire ingress-to-egress path. As importantly, this approach – unlike opportunistic sleep – can be used by interfaces that support only timer-driven sleep. (A router R1 that receives a burst from upstream router R2 at time t1 knows that the next start-of-burst will arrive at time t1$+$B and can hence sleep between bursts.)

The question then is how significant are the savings this approach enables for reasonable additional delay?

We note that the best possible savings would occur if a router received the incoming bursts from *all* ingress routers close in time such that it processes all incoming bursts and returns to sleep thus incurring exactly one sleep/wake transition per B ms. This might appear possible by having ingress routers *coordinate* the times at which they transmit bursts such that bursts from different ingresses arrive close in time at intermediate routers. For example, consider the scenario in Figure 9(a) where ingress routers R0 and R1 are scheduled to transmit traffic at times 2 and 1 respectively. If instead R1 were to schedule its burst for time 7 instead, then bursts from R0 and R2 would align in time at R2 thus reducing the number of distinct burst times - and sleep-to-wake transitions - at downstream routers R3 and R4.

Unfortunately, the example in Figure 9(b) suggests this is unachievable for general topologies. Here, S_i and S_j represent the arrival times of incoming bursts to nodes R3 and R1 respectively and we see that the topology makes it impossible to find times S_i and S_j that could simultaneously align the bursts downstream from R2 and R4.

We thus use a brute-force strategy to evaluate the maximum achievable coordination. For a given topology and traffic workload, we consider the start-of-burst time for traffic from each ingress I to egress J (denoted S_{ij}) and perform an exhaustive search of all S_{ij} to find a set of start times that minimizes the number of transitions across all the interfaces in the network. We call this scheme `optB&B`. Clearly, such an algorithm is not obviously practical and we use it merely as an optimistic bound on what might be achievable were nodes to coordinate in shaping traffic under a buffer-and-burst approach.

We compare the savings from `optB&B` to the upper bound on router sleep time as given by $1-\mu$, where μ is the network utilization. This upper bound is not achievable by any algorithm since (unlike `optB&B`) it does not take into account the overhead δ due to sleep/wake transitions. Nonetheless it serves to capture the loss in savings due to δ and the inability to achieve perfect coordination.

Any traffic shaping incurs some additional complexity and hence a valid question is whether we need *any* traffic shaping, or whether *opportunistic sleeping* that does not require shaping is enough? We therefore also compare `optB&B` to opportunistic sleeping based on wake-on-arrival(W0A). For this naive WoA, we assume optimistically that an interface knows the precise arrival time of the subsequent packet and returns to sleep only for inter-packet arrival periods greater than δ. Because the performance of opportunistic WoA depends greatly on the inter-arrival times of packets we evaluate WoA for two types of traffic: constant bit rate (CBR) and Pareto.

For each of the above bounds, Figure 10 plots the percentage savings under increasing utilization in Abilene. We use a buffer period of B =10ms and assume a (conservative) transition time δ of 1ms. Comparing the savings from `optB&B` to the utilization bound, we see that a traffic shaping approach based on buffer-and-burst

[6]It is a radical approach in the sense that much other work seeks to avoid bursts rather than create them (e.g., token buckets for QOS, congestion avoidance, buffering at routers). As our measurements of loss and delay show, our schemes avoid the pitfalls associated with bursts because we introduce only a bounded and small amount of burstiness and a router never enters sleep until it has cleared all bursts it has built up.

[7]BGP information or MPLS tunnel identifiers offer two possible solutions by which ingress routers can map incoming packets to their egresses. Alternately, an ingress router can arrange packets by destination prefix.

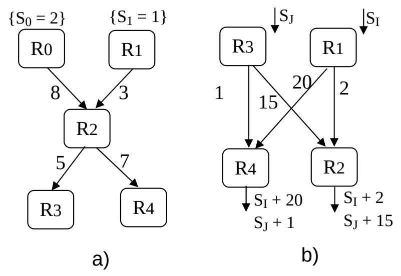


Figure 9: Examples of burst synchronization.

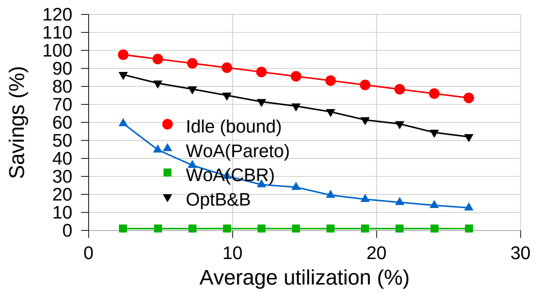


Figure 10: Savings in idle-time consumption using `optB&B` and opportunistic sleeping and compared to the upper bound $(1-\mu)$.

can achieve much of the potential for exploiting sleep. As expected, even at very low utilization, WoA with CBR traffic can rarely sleep; perhaps more surprising is that even with bursty traffic WoA performs relatively poorly. These results suggest that – even assuming hardware WoA – traffic shaping offers a significant improvement over opportunistic sleep.

4.3 A Practical Algorithm

We now look for practical solutions capable of exploiting the potential for savings suggested by the results above.

We consider a very simple buffer-and-burst scheme, called `practB&B`, in which each ingress router sends its bursts destined for the various egresses one after the other in a single "train of bursts". At routers close to the ingress this appears as a single burst which then disperses as it traverses through the network.

`practB&B` bounds the number of bursts (and correspondingly the number of transitions) seen by any router R in an interval of B ms to at most I_R, the total number of ingress routers that send traffic through R. In practice however, our results show that the number of bursts seen by R in time Bms is significantly smaller than this bound.

`practB&B` is simple – it requires no inter-router coordination as the time at which bursts are transmitted is decided independently by each ingress router. For networks supporting wake-on-arrival the implementation is trivial – the only additional feature is the implementation of ingress buffering. For network that employ timer-driven sleeping, packet bursts would need to include a marker denoting the end of burst and notifying the router of when it should expect the next burst on that interface.

4.4 Evaluation

We evaluate the savings *vs.* performance tradeoff achieved by `practB&B` algorithm and the impact of equipment and network parameters on the same.

Savings *vs.* performance using `practB&B` We measure savings in terms of the percentage reduction in idle-time power consumption and compare the savings from `practB&B` to those achievable by `optB&B`. In terms of performance, we measure the average and 98th percentile of the end-to-end packet delay as well as packet loss. We compare performance using `practB&B` to that of a network that never sleeps (as today).

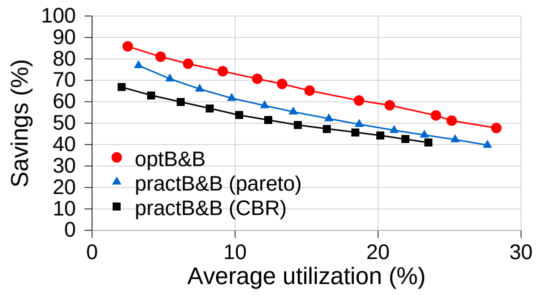
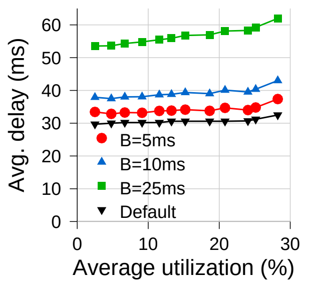


Figure 11: Idle-time energy savings with `practB&B` using CBR and Pareto traffic.

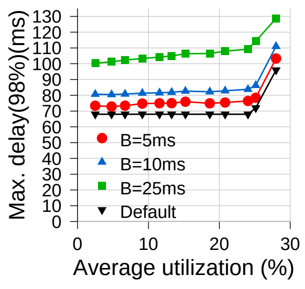
Figure 11 plots the savings with increasing utilization on the Abilene network using a buffering interval B =10ms and $\beta = 0$ (and hence $p_{sleep} = 0$). We plot the savings for both CBR and Pareto traffic workloads workloads. We see that even a scheme as simple as `practB&B` can create and exploit significant opportunities for sleep and approaches the savings achieved by the significantly more complex `optB&B`. As with opportunistic sleeping, we see that `practB&B`'s savings with CBR traffic are lower than for the more bursty Pareto workloads, but that this reduction is significantly smaller in the case of `practB&B` than with opportunistic sleeping (recall Figure 10). That Pareto traffic improves savings is to be expected as burstier traffic only enhances our bunching strategy.

Figure 12(a) and (b) plot the corresponding average and 98th percentile of the end-to-end delay. As expected, we see that the additional delay in both cases is proportional to the buffering interval B. Note that this is the end-to-end delay, reinforcing that the buffering delay B is incurred once for the entire end-to-end path.

We see that for higher B, the delay grows slightly faster with utilization (*e.g.,*compare the absolute increase

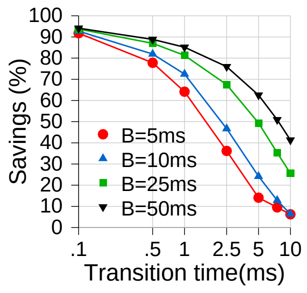


(a) Average delay (optB&B)

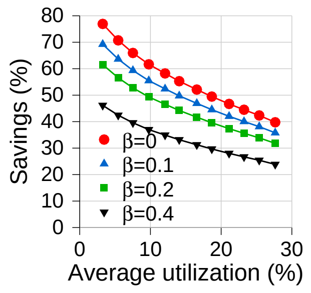


(b) Maximum delay

Figure 12: The impact on delay of practB&B.



(a) Impact of δ



(b) Impact of β

Figure 13: The impact of hardware constants on idle-time savings in practB&B.

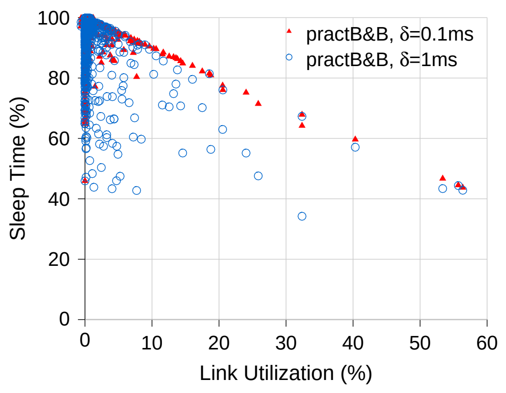


Figure 14: Idle-time energy savings per link

in delay for B =5ms to 25ms) because this situation is more prone to larger bursts overlapping at intermediate routers. However this effect is relatively small even in a situation combining larger B (25ms) and larger utilizations (20-30%) and is negligible for smaller B and/or more typical utilizations.

We see that both average and maximum delays increase abruptly beyond network utilizations exceeding 25%. This occurs when certain links approach full utilization and queuing delays increase (recall that the utilization on the horizontal axis is the average network-wide utilization). However this increase occurs even in the default network scenario and is thus not caused by practB&B's traffic shaping.

Finally, our measurements revealed that practB&B introduced no additional packet loss (relative to the default network scenario) until we approach utilizations that come close to saturating some links. For example, in a network scenario losses greater than 0.1% occur at 41% utilization without any buffering, they occur at 38% utilization with $B = 10ms$, and at 36% utilization with $B = 25$. AS networks do not typically operate with links close to saturation point, we do not expect this additional loss to be a problem in practice.

In summary, the above results suggest that practB&B can yield significant savings with a very small (and controllable) impact on network delay and loss. For example, at a utilization of 10%, a buffer interval of just $B = 5ms$ can save 50% of idle-time consumption while adding an average and maximum delay of 5ms and no loss.

Impact of hardware characteristics We now evaluate how the technology constants of sleep states affect the savings *vs.* performance tradeoff in practB&B. The relevant parameters characterizing sleep states are the transition time δ and β ($= p_{sleep}/p_s$).

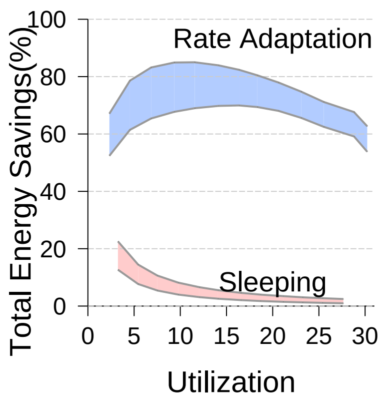
Assuming a value of $\beta = 0.1$, we investigate the effect of δ. Figure 13(a) and (b) plots the savings and performance respectively for increasing transition time and different choices of buffering intervals B. As expected, the magnitude of δ drastically influences the ability to sleep. Figure 13(b) captures the impact of increasing β on the savings for practB&B using a buffering interval B of 10ms. As expected, we see that energy savings are directly proportional to β.

These findings reinforce our intuition that hardware support featuring low-power sleep states and quick transitions (preferably $<$ 1ms) between these states are essential to effectively save energy.

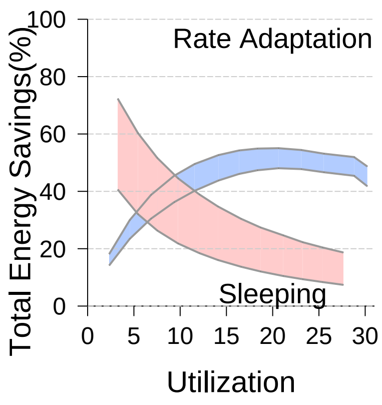
4.5 Impact of network topology and traffic

We evaluate practB&B over the Intel enterprise network traces. Figure 14 shows the savings in idle-time consumption across links – each point in the scatter-plot corresponds to a single link and we look at savings for two values of transition times: $\delta = 0.1ms$ and $\delta = 1ms$. We see that the dominant trends in savings *vs.* utilization remains and that higher δ yields lower savings. For example, on average links spend 85% of idle times in sleep mode for $\delta = 0.1ms$ compared to 80% for $\delta = 1ms$.

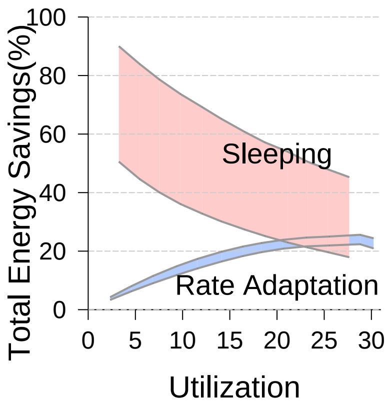
The effect of burstiness in traffic was shown in Figure 11 comparing the performance of practB&B using CBR and Pareto workloads.



(a) $\alpha = 0.01$



(b) $\alpha = 0.1$



(c) $\alpha = 0.5$

Figure 15: Total Energy Saving of Sleeping vs. Rate Adaptation

5 Overall Energy Savings

In the previous sections we evaluated solutions based on their ability to reduce dynamic power consumption (by rate adaptation, Section 3) or idle-time consumption (by sleeping, Section 4). In this section, we place these savings in the context of the *overall* energy consumption of a network element and hence compare the relative merits of rate adaptation *vs.* sleeping.

To compare the solutions, we introduce a proportionality factor α that represents the relative magnitudes of static *vs.* default dynamic power consumption. More precisely, recall that the total energy consumption of a network element can be approximated as:

$$E = (p_d(r_{max}) + p_s)T_{active} + p_s T_{idle}$$

Then, we define the proportionality factor as:

$$\alpha = p_s / p_d(r_{max}) \qquad (2)$$

where $p_d(r_{max})$ is the dynamic power consumption at the default maximum rate and p_s is the static component of power. The precise value of α will depend on the system-level architecture as α varies significantly across components. However, static consumption is typically lower than dynamic consumption and hence we evaluate $0 < \alpha < 1.0$.

By using α, we can compute the overall energy savings using either sleeping and rate-adaptation. Figures 15 plots the savings for `practRA` and `practB&B` for three different values of α, ranging from very low (0.01), to very high (0.5). In each case, we run both `practRA`and `practB&B` using hardware parameters that reflect the best and worst case savings for the algorithm in question. For `practRA`, these parameters are λ (the max rate scale factor) and δ (transition time) which we choose as (λ =5, $\delta = 0.1ms$) and ($\lambda = 2$, $\delta = 1ms$). For `practB&B`, these parameters are β (ratio of sleep to static power draw) and δ (transition time) which we set as ($\delta = 0.1ms$, $\beta = 0.0$) for the best case and ($\delta = 1ms$, $\beta = 0.3$) for the worst.

The conclusion we draw from Figure 15 is that, for each value of α, there is a “boundary” utilization below which sleeping offers greater savings and above which rate adaptation is preferable. Comparing across graphs, we see that the precise value of this boundary utilization depends primarily on the value of α and only secondarily on the hardware parameters for the algorithm. For example, the boundary utilization at $\alpha = 0.1$ varies between approximately 5-12% while that at $\alpha = 0.5$, the boundary utilization lies about 22% utilization. Similarly, we also looked at overall savings for varying traffic parameters (CBR, Pareto) and found that, similar to hardware parameters, the burstiness of traffic has a secondary effect on the boundary utilization. And while the choice between sleeping *vs.* rate adaptation is clear for very low or very high values of α, for more practical values of α, the method of choice will depend on utilization and hardware parameters.

The graphs also reveal a trend in the overall savings with rate adaptation: savings first increase with increasing utilization and then decrease. In other words, there is a “sweet spot” utilization value that maximizes the overall savings due to rate adaptation. This is because at very low utilization, the dynamic power is a small part of the total power, and so even if we save most of it, the overall savings are small. As the utilization increases, the rate adaptation saves more of the total energy. However, at large utilizations, the potential for rate adaptation becomes increasingly small, and consequently the overall savings decrease again. In addition, we see that this sweet-spot utilization grows with α This is because the utilization at which rate adaptation yields maximum savings depends on the energy fraction represented by

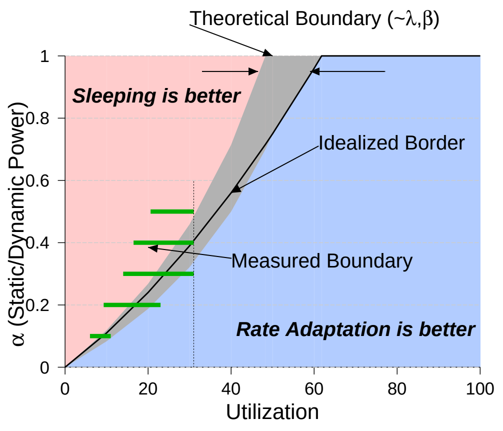


Figure 16: When is it better to sleep/rate adapt

the dynamic component.

Figure 16 summarizes the tradeoff between sleeping and rate-adaptation. It plots the boundary utilization for different values of α and network utilizations for the best and worst case combinations of hardware parameters given before. The length of the horizontal lines captures the effect of varying hardware constants on the boundary utilization. These lines stop at a utilization of 31% because that the maximum we see in our traffic traces. As a point of comparison, we also plot the boundary utilization for a single idealized link. We compute the energy consumption of an idealized link that sleeps as:

$$E_{sleep} = p_d(r_{max})\mu T + p_{sleep}(1-\mu)T$$

Similarly, the idealized link with rate adaptation is one that runs with an average rate of μr_{max} for an energy consumption of:

$$E_{rate} = p_d(\mu r_{max})T$$

The gray zone in Figure 16 represents the spread in boundary utilizations for this idealized link as the hardware constants β and λ are varied.

In summary, we find that both sleeping and rate-adaptation are useful for moderate values of α, with the tradeoff between them depending primarily on utilization. Results such as this can guide operators in deciding how to best run their networks. For example, an operator might choose to run the network with rate adaptation during the day and sleeping at night based on where the boundary utilization intersects diurnal behavior, or identify components of the network with consistently low (or high) utilization to be run with sleeping (or rate-adaptation). Similarly, these results can inform hardware system architects in their selection of components for desirable α.

6 Related Work

There is a large body of work on power management in contexts complementary to ours. This includes power provisioning and load balancing in data centers[4, 8], and OS techniques to extend battery lifetimes in mobiles[9, 31].

Perhaps the first to draw attention to the problem of saving overall energy in the network was an early position paper by Gupta *et al.* [12]. They use data from the US Department of Commerce to detail the growth in network energy consumption and argue the case for energy-saving network protocols, including the possibility of wake-on-arrival in wired routers. In follow-on work they evaluate the application of opportunistic sleeping in a campus LAN environment [23, 11]. As discussed in Section 4, opportunistic sleep is not well suited to more utilized networks and cannot be applied to hardware that supports only timer-driven sleep.

Other recent work looks at powering-down redundant access points (APs) in enterprise wireless networks [16]. The authors propose that a central server collect AP connectivity and utilization information to determine which APs can be safely powered down. This approach is less applicable to wired networks that exhibit much less redundancy; while such networks may tolerate the failure of one or more links, it is harder to bring down an entire path.

Sleeping has also been explored in the context of 802.11 to save client power, e.g., see [2]. The 802.11 standard itself includes two schemes (Power-Save Poll and Automatic Power Save Delivery) by which access points may buffer packets so that clients may sleep for short intervals. In some sense, our proposal for bunching traffic to improve sleep opportunities can be viewed as extending this idea deep into the network. The challenge in this extension lies in achieving this coordination across multiple hops in general network topologies and doing so with a bounded increase in end-to-end delay. Finally, the IEEE Energy Efficient Ethernet study group has recently started to explore rate adaptation for energy savings. Initial studies consider individual links and are based on synthetic traffic and infinite buffers [3].

In the domain of sensor networks, there have been numerous efforts to design energy efficient protocols, as the energy budget is one of the most stringent constraints. Approaches investigated include putting nodes to sleep using TDMA-like techniques to coordinate transmission and idle times (*e.g.*,SPAN [5] for 802.11 networks, FPS [13] for sensor networks), and distributed algorithms for sleeping (e.g. S-MAC [29]). The sensor network context differs from ours in many ways. Solutions typically assume a broadcast medium and dense deployments where entire nodes can be put to sleep (and the data routed via alternate paths as opposed to rousing the node to forward data). Moreover, packet rates are typically lower and longer delays more acceptable.

7 Conclusion

We have argued that power management states that slow down links and put components to sleep stand to save much of the present energy expenditure of networks. At a high-level, this is apparent from the facts that while network energy consumption is growing networks continue to operate at low average utilizations. We present the design and evaluation of simple power management algorithms that exploit these states for energy conservation and show that – with the right hardware support – there is the potential for saving much energy with a small and bounded impact on performance, *e.g.,* a few milliseconds of delay. We hope these preliminary results will encourage the development of hardware support for power saving as well as algorithms that use them more effectively to realize greater savings.

References

[1] Power and Thermal Management in the Intel Core Duo Processor. In *Intel Technology Review, Volume 10, Issue 2, Section 4*. 2006.

[2] Y. Agarwal, R. Chandra, et al. Wireless wakeups revisited: energy management for voip over wi-fi smartphones. In *ACM MobiSys*. 2007.

[3] C. Gunaratne and K. Christensen and B. Nordman and S. Suen. Reducing the energy consumption of Ethernet with Adaptive Link Rate (ALR). In *Submission to IEEE Transactions on Computers*. 2007.

[4] J. S. Chase, D. C. Anderson, et al. Managing energy and server resources in hosting centers. In *ACM SOSP*. 2001.

[5] B. Chen, K. Jamieson, et al. Span: An energy-efficient coordination algorithm for topology maintenance in ad hoc wireless networks. In *ACM MOBICOM*, pp. 85–96. 2001.

[6] Cisco Systems. NetFlow services and applications. White Paper, 2000.

[7] X. Fan, C. S. Ellis, et al. Memory Controller Policies for DRAM Power Management. In *International Symposium on Low Power Electronics and Design*. 2003.

[8] X. Fan, W.-D. Weber, et al. Power provisioning for a warehouse-sized computer. In *ACM ISCA*. 2007.

[9] J. Flinn and M. Satyanarayanan. Energy-aware adaptation for mobile applications. In *ACM SOSP*. 1999.

[10] C. Gunaratne, K. Christensen, et al. Managing Energy Consumption Costs in Desktop PCs and LAN Switches with Proxying, Split TCP Connections, and Scaling of Link Speed. October 2005.

[11] M. Gupta, S. Grover, et al. A feasibility study for power management in lan switches. In *ICNP*. 2004.

[12] M. Gupta and S. Singh. Greening of the internet. In *ACM SIGCOMM, Karlsruhe, Germany*. August 2003.

[13] B. Hohlt, L. Doherty, et al. Flexible power scheduling for sensor networks. In *IEEE and ACM Third International Symposium on Information Processing in Sensor Networks (IPSN)*. April 2004.

[14] IEEE 802.3 Energy Efficient Ethernet Study Group. http://grouper.ieee.org/groups/802/3/eee_study/.

[15] Ipmon Sprint. The Applied Research Group. http://ipmon.sprint.com/.

[16] A. Jardosh, G. Iannaccone, et al. Towards an Energy-Star WLAN Infrastructure. In *HOTMOBILE*. 2007.

[17] S. Kandula, D. Katabi, et al. Walking the tightrope: Responsive yet stable traffic engineering. In *ACM SIGCOMM 2005*.

[18] K. Kawamoto, J. G. Koomey, et al. Electricity used by office equipment and network equipment in the US. 2002.

[19] J. Kim and M. A. Horowitz. Adaptive supply serial links with sub-1v operation and per-pin clock recovery. In *International Solid-State Circuits Conference*. 2002.

[20] M. Lin and Y. Ganjali. Power-efficient rate scheduling in wireless links using computational geometric algorithms. In *IWCMC*. 2006.

[21] J. Lorch. Operating systems techniques for reducing processor energy consumption. In *Ph.D. Thesis, University of California, Berkeley*. 1994.

[22] M. Mandviwalla and N.-F. Tzeng. Energy-efficient scheme for multiprocessor-based router linecards. In *IEEE SAINT*. 2006.

[23] Maruti Gupta and Suresh Singh. Dynamic Ethernet Link Shutdown for Energy Conservation on Ethernet Links. In *IEEE ICC*. 2007.

[24] E. Miranda and L. McGarry. Power/thermal impact of networking computing. In *Cisco System Research Symposium, August, 2006*.

[25] B. Normand. Energy Efficient Ethernet, Outstanding Questions. 2007.

[26] K. W. Roth, F. Goldstein, et al. Energy Consumption by Office and Telecommunications Equipment in Commercial Buildings - Volume I: Energy Consumption Baseline. Tech. Rep. 72895-00, Arthur D. Little, Inc, Jan. 2002.

[27] The Abilene Observatory. http://abilene.internet2.edu/. observatory.

[28] G.-Y. Wei. Energy-efficient I/O interface design with adaptive power-supply regulation. In *Journal of Solid-State Circuits*. 2001.

[29] W. Ye, J. Heidemann, et al. An energy-efficient mac protocol for wireless sensor networks. In *ACM INFOCOM*. 2002.

[30] Yin Zhang's AbileneTM,http://www.cs.utexas.edu/ yzhang. /research/AbileneTM.

[31] W. Yuan and K. Nahrstedt. Energy-efficient soft real-time cpu scheduling for mobile multimedia systems. In *ACM SOSP*. 2003.

[32] M. Yuksel, B. Sikdar, et al. Workload generation for ns simulations of wide area networks and the internet. In *Communication Networks and Distributed Systems Modeling and Simulation Conference*. 2000.

[33] B. Zhai, D. Blaauw, et al. Theoretical and practical limits of dynamic voltage scaling. In *DAC*. 2004.