A Feasibility Study for Power Management in LAN Switches

Maruti Gupta
Computer Science Department
Portland State University
{mgupta}@cs.pdx.edu

Satyajit Grover
Computer Science Department
Portland State University
{satyajit}@cs.pdx.edu

Suresh Singh
Computer Science Department
Portland State University
{singh}@cs.pdx.edu

***Abstract*— We examine the feasibility of introducing power management schemes in network devices in the LAN. Specifically, we investigate the possibility of putting various components on LAN switches to sleep *during periods of low traffic activity*. Traffic collected in our LAN indicates that there are significant periods of inactivity on specific switch interfaces. Using an abstract sleep model devised for LAN switches, we examine the potential energy savings possible for different times of day and different interfaces (e.g., interfaces connecting to hosts to switches, or interfaces connecting switches, or interfaces connecting switches and routers). Algorithms developed for sleeping, based on periodic protocol behavior as well as traffic estimation are shown to be capable of conserving significant amounts of energy. Our results show that sleeping is indeed feasible in the LAN and in some cases, with very little impact on other protocols. However, we note that in order to maximize energy savings while minimizing sleep-related losses, we need hardware that supports sleeping.**

I. Introduction

Currently, power management schemes exist to minimize the power consumption on devices such as desktops, notebooks and a number of other portable devices. The schemes used for conserving power are generally implemented by the operating system in the device and use various power-saving techniques such as dynamic voltage scaling (DVS), slowing clocks and using lower power-consuming modes as provided by the underlying hardware. However, no such dynamic power management schemes are available for internet devices such as routers and switches at the system level.

In this paper, we look at the feasibility of introducing such schemes in LAN switches. We chose to begin with LAN devices and in particular LAN switches for several reasons: LAN switches comprise the bulk of network devices in the LAN and they also consume the largest percentage of energy (see Table I in [5]). Increasingly, hosts are connected on point-to-point links on switches in a star topology in order to maximize throughput. Thus, a majority of interfaces on LAN switches are directly connected to hosts, that intuitively would have longer periods of low traffic activity than on those interfaces that carry aggregated traffic from many hosts. While the sleep models we propose in this paper can be easily extended to be used for routers as well, a thorough analysis of this approach for routers is beyond the scope of this paper.

Given the intention to save energy on switches, how do we then proceed? In order to save energy in a device, we can either turn it off or put it into deep sleep states where most of the components are powered off or clocked at a lower frequency at lower voltage levels. The one caveat is that these approaches can only be used when the device is idle for some minimal amount of time (very frequent power on/off actually uses more power due to spikes in current draw when a device is powered on and, furthermore, devices take a certain amount of time to transition between sleep and wake states that could result in packet losses). Turning our attention to the LAN switch, we note that saving energy here translates to powering off or putting to sleep LAN switch components, interfaces, or entire switches. However, the side effect of putting ports or switches to sleep is that layer 2 protocols running on the switches may be negatively affected. Thus, implementing power management schemes in a switch presents several challenges: switches do not function in isolation and hence slowing or powering down switches can result in performance penalties in terms of network throughput and end-to-end delay, or worse, packet loss. In addition, it may affect the functioning of various network protocols at layer 2 and above.

In this paper, we examine the different questions arising from the approach of putting switch components to sleep. Our analysis of these questions has been organized into the following sections. In section 3 we use traffic data from our LAN to show that there are significant periods of inactivity in our LAN that can be used for sleeping. In section 4, we develop an abstract sleep model for a generic switch architecture and use it to discuss algorithms for sleeping. Section 5 covers our study of the performance of these algorithms on the traffic data collected at various interfaces at different locations in our LAN. We also use simulations to further study the costs and benefits of sleeping. Section 6 deals with the impact of sleeping on the various protocols running in the LAN and LANs topologies to better facilitate sleeping.

II. Related Work

There has been a great deal of work done on the design of energy efficient protocols in wireless networks. Energy is wasted in wireless networks for two primary reasons – idle energy consumption when the radio is idle and energy consumed when the radio receives a packet that is not meant for it. The former problem occurs in wired networks as well and is the one we study here. The latter problem, on the other hand, is a feature of the broadcast nature of the radio medium. The approaches developed in the wireless world to reduce energy consumption include distributed algorithms for sleeping (e.g., S-MAC [18]), or using TDMA like techniques (as in the Point Coordination Function mode of 802.11b) where a base station or cluster head schedules node transmissions and sleep periods as in [4]. There are a great many techniques that have been developed but we do not provide a summary here because they are not directly relevant to our problem of deciding when to send an interface on a point-to-point link to sleep.

[5] provides motivation for saving energy in the Internet based on energy consumption data gathered by the US Department of Commerce. The authors also suggest that sleeping during periods of low activity is a good way to save energy. However, the paper does not examine this idea in detail nor does it provide any sleeping algorithms.

Previous work in the area of power dissipation in switches has been devoted to the estimation of power consumption in switch fabrics using various methods such as developing statistical traffic models[3] or various analytical models [12], [10], [19], [16], [6]. Their focus has mainly been on the various types of interconnection network fabrics used in switches and an analysis of their power consumption. Power management schemes have been proposed for interconnection network fabrics which are deployed in routers, clusters, server blades etc. These include using DVS with links in response to network traffic [15] and using on/off links for network power optimization [13]. In Peh et al [14], the authors present a mechanism that monitors power utilization of links in an interconnection network and regulates the network power consumption by using power throttling techniques to limit local power dissipation to a certain pre-determined power budget.

The approach in all the work mentioned above, has been focused on the switching fabric component in order to minimize low-power dissipation. In this paper, we present algorithms that are designed to extend power-saving techniques such as dynamic voltage scaling, powering down devices, to all components of the switch hardware by incorporating the algorithms in the device operating system. Thus, our work here examines methods to reduce the power consumption in switches[1] dynamically using low-power states depending upon traffic activity at each interface. We also analyze the impact of sleeping on layer 2 protocols.

III. Feasibility and Models for Sleeping

Examining traffic data from various locations within our LAN, we observe significant periods of inactivity (barring periodic Layer 2 and Layer 3 control packets) that lead us to believe that sleeping is possible. In this section, we begin by describing a small representative subset of this data and then proceed to develop models for sleeping.

To answer the question – are there enough idle periods to justify sleeping? – we collected data at a switch which has 82 ports running at 100Mbps organized in two modules and 30 Gigabit Ethernet ports organized in three modules (we describe the traffic characteristics and the LAN in more detail in section V-A). This switch is connected to 2 routers as well as to other switches and several end hosts. In terms of the spanning tree, it is two hops from the root switch. Packet arrivals in/out of the switch were logged on a workday over a period of two hours – 3am-5am and 8pm-10pm. The former represents a low activity time for the switch and the latter represents a high activity time (being a university campus, the network remains very busy in the evening). For a typical port we compute the inter-activity time which includes packet arrivals to a port from the wire as well as from other ports for transmission. We then calculate the *Percentage of time that the inter-activity times exceed some value, say x seconds*. The left-hand plot in Figure 1 plots this percentage as a function of x for a typical port. As shown in the figure, for the Low activity times, 60% of the 2 hour period saw inter-activity times greater than 20 seconds. For the high activity period this value is about 10%. However, if we look at the percentage of time the inter-activity times are greater than 1 second for the high activity period, we get a very large value of approximately 80%. These numbers suggest that the port can be put to sleep for one or more seconds at a time during low activity periods or for up to one second during high activity periods.

The plot on the right shows the same data for the entire switch. Unlike the case with an individual port, however, we see that all inter-activity times are significantly smaller than one second long and thus putting the entire switch to sleep may not be feasible.

A. Models for Sleeping

Currently there exist no sleep models for switches. Thus, for the purposes of this paper, we assume a typical switch architecture to be modular with each module or line card comprising of several interfaces and containing per-interface processing elements and other relevant circuitry for each interface. Considering the fact that current LAN switches support enormous port densities (about 24-48 100Mbps interfaces per module), each interface in a line card is therefore provided with significant processing power as well as memory buffers to enable high-speed packet forwarding. In line with current trends in hardware design for high-speed routers and switches, where the packet processing functionality is being pushed out to the line cards [23], [21], for this paper we assume that line cards contain most of complexity of the device and this is where we look to put components to sleep.

A typical line card contains several components, each of which can be put into one or more power saving modes. Thus, the CPU can be clocked at lower rates, the memory (at each buffer or the forwarding cache, etc.) can each enter one or more sleep states, if the line card has a switching fabric, either portions of it can be put to a sleep state or the whole fabric can be clocked slower [6], and so on. Let us represent the sleep states of each device on the line card as a linear sequence [Wake $, s_1, s_2, \ldots, s_k,$ Off] where s_i is a deeper sleep state than s_{i-1} (e.g., a device may operate at different clock frequencies, each then denotes a sleep state in

[1] We use the term switches from here onwards to mean layer 2 switches used in LANs

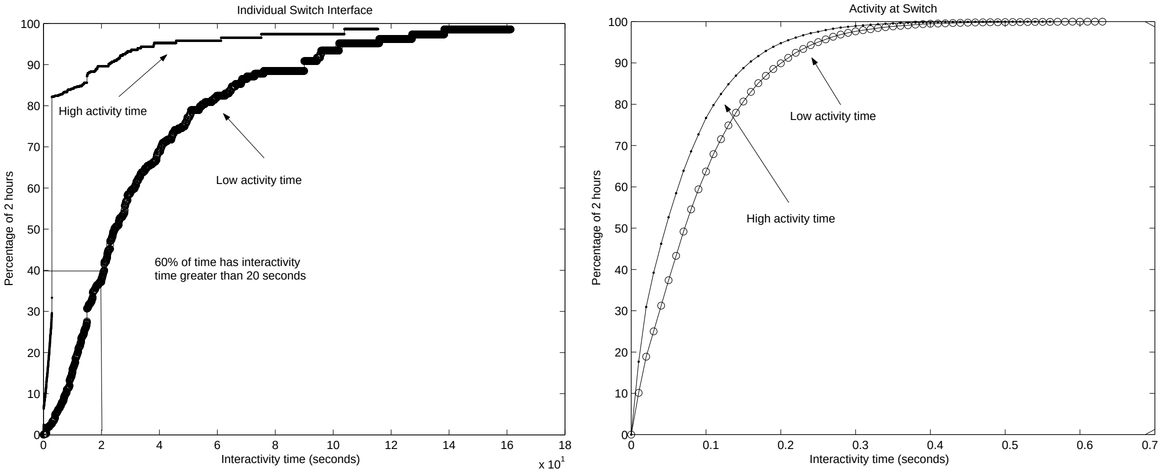

Fig. 1. Interactivity times during high activity and low activity times.

our usage). Associated with each state s_i is a tuple $\{$power draw, δt_i, e_i $\}$, where δt_i denotes the transition time from state s_i to s_{i-1} (we assume that going to deeper sleep states is instantaneous), and e_i is the spike in power draw when this transition occurs ([11]). Given this representation for each device on the line card, we can construct the set of sleep states for the line card as a whole as a *cross product* of the sleep states of each of the individual devices. While complete, this representation is unnecessarily complex because most of the sleep states will be meaningless from a *functional* standpoint and some combinations will not be supported by the hardware. For instance, the cross product gives us a state where the CPU is in Wake while all other devices are in their deepest sleep state, and so on. Therefore, we need to significantly pare down this representation to obtain more realistic sleep models for the line card.

As noted above, the sleep model for a line card is obtained from the sleep models of its constituent parts. While it is a research task to develop appropriate sleep models for line cards, in this section we develop a simple sleep model of an interface for illustrative purposes. The model we develop is based on the *functionality* of the interface in each of its sleep states. In other words, the sleep model is based on varying degrees of services offered, which is similar to what is used in the ACPI standard [7].

Figure 2 shows the three sleep states of our first model (sandwiched between Off and Wake states) where the energy draw in the states decreases as we go from Wake to Off. One property common to all three sleep states is that *interface state* is preserved, i.e., the station cache, learning bridge table, etc. are all stored even during sleep.

- *Simple Sleep:* In Simple Sleep, the interface sets a sleep timer[2], and only wakes up when the timer expires. All packets arriving during the sleep period are lost. Barring the energy required to maintain interface state information in memory, all other components of the interface are powered off or put into very low energy states.

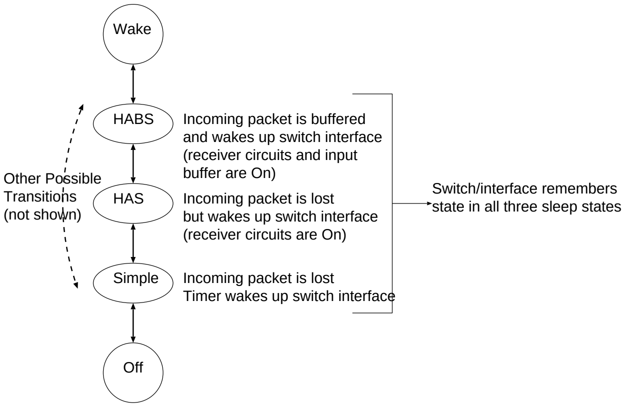
Wake
HABS
HAS
Simple
Off
Incoming packet is buffered
and wakes up switch interface
(receiver circuits and input
buffer are On)
Incoming packet is lost
but wakes up switch interface
(receiver circuits are On)
Incoming packet is lost
Timer wakes up switch interface
Other Possible
Transitions
(not shown)
Switch/interface remembers
state in all three sleep states

Fig. 2. Function-based Sleep Model States.

- *HAS:* (Hardware Assisted Sleep) is a step up in functionality. Here, an incoming packet wakes up the interface but is then lost. Thus, the interface can wake up either when the sleep timer expires or when a packet arrives.
- *HABS:* (Hardware Assisted Buffered Sleep) is the most sophisticated sleep state. Here, an incoming packet wakes up the interface and is buffered. Comparing HAS and HABS, we note that HABS is an extension of HAS where we require the input buffer to remain powered on so as to buffer incoming packets. Thus, in HABS, the input circuits for receiving, demodulating, and other receiver-based functions are powered on whereas in HAS they are not. In HAS, we only require that the interface be able to detect activity on the wire/fiber. This then causes the remainder of the interface to be woken up.

We assume that transitioning from a deeper sleep state to a lighter sleep state (HABS to HAS, for example) takes a

[2] The value of this timer can be set so as to ensure that the interface wakes up for periodic traffic (e.g., the Hello packet of STP). However, non-periodic traffic (data packets or ARP, etc.) cannot be predicted and will be lost if they arrive during a sleep.

non-zero transition time δ and results in a spike in energy consumption (see [11] for an example from the wireless domain). On the other hand, transitioning to deeper sleep states is assumed to be instantaneous and does not use energy.

B. Implications of Sleeping

Sleeping has implications for Layer 2 and above protocol functioning as well as for traffic behavior. We defer a discussion of the protocol-related issues to later sections (section VI) and focus on the traffic issues here.

- *Simple Sleep:* Since all packets are lost, we can envision disastrous performance for application and higher-layer protocols. For instance, a TCP connection will generate retransmissions which has two effects – first, the end application will see poor throughput and second, the energy savings we get by sleeping will probably be offset by the additional cost of retransmissions. This limits our choices for using this state to the following:
  - *Interface connected to end host:* ACPI – Advanced Configuration and Power Interface – for personal computers enables the operating system to run sophisticated power management algorithms. Imagine extending this functionality (call it Extended ACPI) where the host informs the switch interface it is connected to when it is going to sleep. If this technology existed, then the switch interface could very easily enter Simple Sleep. The reason for using Simple Sleep rather than Off is that interface state is preserved in Simple Sleep which ensures correct protocol behavior. In addition, we also need to ensure that connectivity is not lost when the host comes back on.
  - *Interfaces connecting switches:* Interfaces connecting two switches or switches to servers can be put to Simple Sleep only if we can guarantee that no packets will be sent to a sleeping interface. This can be accomplished by having some form of LAN-wide protocol that coordinates sleeping and ensures that all data paths traversing a sleeping interface have buffering available. For example, an upstream switch could buffer packets for its downstream neighbor for the sleep duration.
- *HAS:* This state is more amenable to being used on interfaces with less predictable traffic than Simple Sleep because an incoming packet wakes up the sleeping interface and thus subsequent packets will not be lost. While better than Simple Sleep, however, HAS still loses some number of packets which will negatively impact protocol performance. One way around this is for the upstream interface (of a sleeping interface) to introduce a dummy packet ahead of the packet to be sent to its sleeping neighbor. By setting an inter-packet delay equal to the time to transition to the Wake state, the dummy packet can wake up the neighbor in time to receive the data packet. The obvious drawbacks include added traffic and increased processing.
- *HABS:* Unlike HAS, the HABS state does not require dummy packet transmissions on a link or the exchange of sleep timers between neighboring interfaces because the packet which wakes up the interface is not lost. This benefit, however, comes at the cost of lower energy savings as compared with Simple Sleep and HAS and the added delay (equal to the HABS – Wake transition time) for each packet that wakes up an interface (HAS also introduces a delay while Simple Sleep discards the packet).

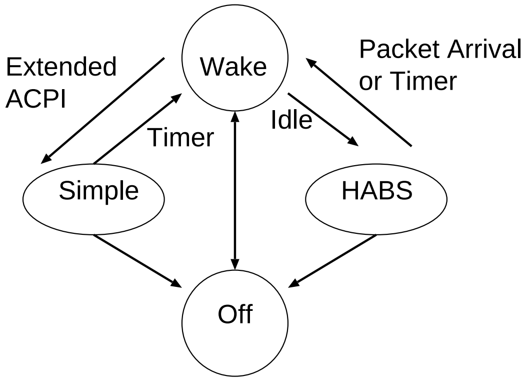

Fig. 3. Simplified Sleep Model.

Based on the above discussion, it appears that a simpler sleeping model that eliminates HAS and restricts state transitions, as shown in Figure 3, is more appropriate.

1) For switch interfaces connected to end hosts (user machines), we can use Simple Sleep, particularly in combination with some form of Extended ACPI where the host informs the switch that it is going to sleep.
2) For switch to switch or switch to router interfaces, we use HABS because it simplifies the implementation of sleep algorithms by reducing coordination between switches. We use HABS for switch interfaces connected to hosts if the hosts do not use Extended ACPI or similar power management approaches.

In the next section we develop sleep algorithms using this simplified sleep model.

IV. Algorithms for Sleeping

Three questions any sleeping algorithm needs to answer are:

- *When* can an interface go to sleep
- *Length* of the sleep interval, t_S (either in Simple Sleep or HABS)
- *Length* of wake interval between consecutive sleeps, t_I

A. Wake – Simple Sleep

As shown in Figure 3, we propose that the Simple Sleep state only be used on interfaces connected to end hosts which use technology such as Extended ACPI. Thus, the switch interface goes to sleep when the end host goes to sleep and it wakes up periodically (every t_S) to check to see if the host has woken up. If it has, then the interface remains awake until the host falls back to sleep again. Thus, the answer to the three questions posed above for the *Wake – Simple Sleep* transitions is simple and the only issue is the selection of t_I and t_S.

How does the switch interface know that the host has woken up? We propose the following simple mechanism. After the end host wakes up, the network interface on the host begins sending packets to the switch interface with some period τ. Then, if $\tau < t_I$, the switch will receive at least one such packet during its wake interval and can then remain awake until the host's network interface informs it that the host is going back to sleep. Given this mechanism, we can see that the selection of t_S is more of a policy question than an optimization one. For instance, if t_S is very long, then the end host will remain disconnected from the LAN (because the switch interface is asleep) for long enough that the user will notice. This is clearly not desirable and we can envision the system administrator setting an appropriate t_S value.

B. Wake – HABS

We next look at the *Wake – HABS* transitions which can be used for all kinds of devices. The following *Optimal Algorithm* illustrates the main ideas behind the sleeping algorithm.

- The decision on *whether* to sleep is made immediately after processing the last packet in the buffer.
- Say x is the time to the next packet arrival. e_S and e_I denote the energy consumed by the switch interface in Sleep state and in the Wake state. e_W is the energy consumed when the interface transitions from the sleep state to the wake state (δ is the time taken by this transition). e_T is the energy consumed by the end host's network interface to transmit the wakeup packets to the switch interface. Then,
  - if $(x-\delta)e_S+\delta e_W < xe_I$ the interface goes to sleep for length of time $t_S = x - \delta$. t_I is the time to process the incoming packet at time x
  - Otherwise, the interface stays awake.

Deriving from this optimal algorithm, we can define two simple practical algorithms as follows:

1) *Estimated Algorithm:* We use an estimator for $\bar{x}$ (mean time to next packet) and sleep if,

$$(\bar{x}-\delta)e_S+\delta e_W < \bar{x}e_I \tag{1}$$

In this paper we use the EWMA[24] (exponentially weighted moving average) filter,

$$\bar{x}_t = \alpha\bar{x}_{t-1} + (1-\alpha)x_t \tag{2}$$

where x_t is the observed value and $\alpha < 1$, α is a filter specific constant. Other estimators can also be used but as our results show, this filter works quite well.
Unlike the Optimal Algorithm, the *interface sleeps until woken up by an arriving packet* and it then stays awake until there are no more packets to process, whereupon it makes the sleep decision again using $\bar{x}$. This implies that each packet arriving to a sleeping interface will incur an *additional delay* of δ.

2) *Estimated & Periodic Algorithm:* Since layer 2 and 3 protocols generate periodic traffic (e.g. configuration BPDUs in STP), we exploit this periodicity to make more informed sleep decisions.

- We maintain timers for all periodic traffic. After processing the last packet in the buffer, we obtain y, the time to the next periodic packet, by using the periodicity of the various protocols and knowledge of when we last saw a periodic packet from those protocols.
- As in the *Estimated* case, we determine $\bar{x}$ for the non-periodic traffic.

The interface sleeps if,

$$(\min(\bar{x},y)-\delta)e_S+\delta e_W < \min(\bar{x},y)e_I$$

V. Estimated Energy Savings in our LAN

In this section we study the energy savings obtainable if interfaces use HABS. We do not consider the Simple Sleep state here because none of the end hosts ever went to sleep and therefore Simple Sleep would show large number of packet losses.

In order to determine the energy savings possible by sleeping, we used the following metric:

$$\gamma = \frac{E}{E_S}$$

if $\gamma > 1$ then we get energy savings using sleep otherwise, sleeping results in energy loss. Here E_S denotes the energy consumed if we use sleep algorithms and E denotes the energy consumed without sleeping.

To get expressions for E and E_S, assume that over some length of time T, N packets are processed. Then,

$$E = e_I(T - Nt_P) + Ne_Pt_P$$

(t_P – time to process a packet and e_P – energy to process packets) where the first term indicates the energy during idle periods when the interface is awake but not doing anything and the second term denotes the processing cost for the N packets. To compute E_S, assume that the total time spent sleeping is T_S. Then,

$$E_S = e_ST_S + E_I(T - Nt_P - N\delta - T_S) + Ne_W\delta + Ne_Pt_P$$

where the terms are, energy spent sleeping, energy spent in wake state (but idle), energy spent to wake up N times, and energy spent to process N packets. Note that no packets are lost in HABS which is why the Ne_Pt_P term appears in E_S.

It is easy to see that γ is affected by the values for e_I, e_S, e_W, e_P and δ. Unfortunately, there are no values available for these constants for present-day switches (with the exception of e_I, the other values are not available because switches do not support sleep states as we have described them). Given this constraint, we decided to use ranges of values using the wireless domain and network adapters implementing ACPI standards as our guide. 802.11b radio cards using the Prism chipset [1] take 5ms to transition up. from sleep to wake state while the radio described in [11] takes 0.5ms. The RFM 1000 radio used in the Motes sensor nodes [2] takes between 11 and 20ms to wake. For network adapter cards, the power consumed during idle state is given as 0.5W and the power consumption in the deepest sleep state is 0.17W [22]. In terms of energy draw e_W during the wake up process, [11] reports a value equivalent to the energy used for transmitting a packet (i.e., $e_P = e_W$). Finally, the RFM 1000 radio reportedly uses 1/1000th the energy in sleep as compared with idle whereas, the 802.11b Orinico card uses 1/15th the value, [8].

Given this, we propose using the range $0.001 \leq \delta \leq 0.1$ sec for the transition from HABS to Wake time to study the impact of δ on energy savings. For the power draw, we use $e_I = 1$ Watt, $e_S = 0.1$W and $e_W = e_P = 2$W for most studies and $e_S = 0.5$W for one study just to show the dependence of sleeping power draw on overall energy consumption (note that the ratio e_W/e_S can be varied in addition to varying δ but we saw little that could not be observed using the smaller set of values we have presented above).

To illustrate the impact of different e_S values (for a fixed e_W) we return to the example discussed in Figure 1. In Figure 4 we plot $\gamma = \frac{E}{E_S}$ for the Optimal Algorithm as a function of δ for the High activity and Low activity times from Figure 1. We make the following observations:

- In general, we get lower energy consumption if we use sleeping. However, the amount of energy savings depend on the load with low activity periods giving larger energy savings. This is not surprising because the interface is off for longer periods.
- The difference between $e_S = 0.1$ and $e_S = 0.5$ is dramatic but also makes sense. If e_S is 0.5, sleeping is only beneficial if the interactivity times are large enough to justify sleeping. Otherwise, the interface stays awake. However, even here we notice a 2x improvement over not sleeping.

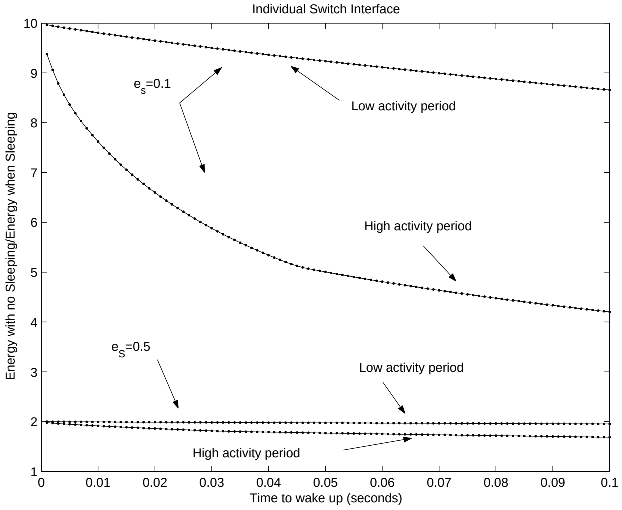

Fig. 4. Ratio of energy without sleep to optimal sleep from Figure 1.

A. Results

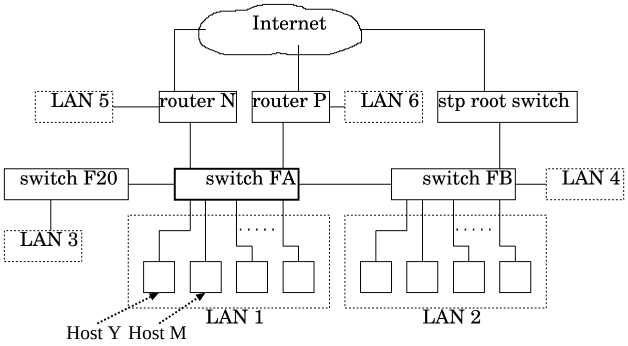

Fig. 5. Schematic of our LAN.

In order to evaluate the two algorithms based on estimating $\bar{x}$, we collected traffic data in our network (see Figure 5) at the interfaces indicated. The traffic was collected at two switch – host interfaces where we looked at the switch interface (Y–FA and M–FA), one switch – switch interface (FA–FB), and one switch – router (FA–P) interface. As we can see, switch FA is 2 hops away from the root switch. This switch is the one that is described in section III and Figure 1. Data was collected during light activity times (3am-5am) as well as during heavy activity times (8pm-10pm) for two hours on a regular workday using tcpdump for gathering traces and then using the ethereal tool for analysis of the traces. A summary of the traffic collected at each of these interfaces is detailed in Table I.

From this table we can make the following observations:

- There are three protocols that generate periodic traffic – STP Hello, OSPF Hello, and CDP (Cisco Discovery Protocol). In addition, there are various application layer protocols that also generate periodic traffic (e.g., IPX/SDP, Microsoft clients, etc.).
- We selected two hosts to reflect different types of behavior. Host M is a Unix server that sees a great deal of RPC traffic (primarily because of NFS) and hence the percentage of periodic traffic is very small. Furthermore, there is little difference between light and heavy periods. Host Y, on the other hand, is a Windows machine used by an individual user and hence we note the big difference between light and heavy periods. Neither machine sleeps and furthermore, host Y shows traffic even without a user present because of periodic Windows functions like the Outlook mail client, etc.
- For host Y, periodic traffic dominates during light periods – STP traffic accounts for 74.1% traffic during light periods as opposed to 4.76% during heavy loads. Thus, we would expect the *Estimated & Periodic* algorithm to perform better than the *Estimated* algorithm.
- The switch–router interface and switch–switch interface show little traffic variation between different times of day. In addition, for these two interfaces, the percentage of periodic traffic is negligible.

	Host Y		Host M		Switch FB		Router N	
Link Speed	100 Mbps		100 Mbps		1 Gbps		2 Gbps	
Traffic Type	Light	Heavy	Light	Heavy	Light	Heavy	Light	Heavy
Total Packets	7636	77164	122622	107819	521611	617698	188803	634020
Ethernet (%)	95.14	99.69	99.8	99.65	100	100	100	100
ARP (%)	0.83	0.06	0.14	0.13	0.07	0.79	37.25	18.37
STP (%)	74.10	4.76	3	5.25	negligible		negligible	
CDP (%)	2.44	0.16	0.1	0.17	negligible		negligible	
IP (%)	12.93	94.4	96.5	93.7	90.9	91.4	47.4	81.63
Breakdown of IP traffic by higher layer protocols								
UDP (%)	1.15	0.25	90.97	85.7	87.96	89.7	29.1	11.4
RPC (%)	0	0	90.5	84.67	87.2	89.3	0	0.03
TCP (%)	7.56	93.68	5.24	7.84	2.52	1.46	13.72	58.67
ICMP (%)	4.22	0.46	0.26	0.24	0	0	1.39	1.51
OSPF (%)	0	0	0	0	0.26	0.14	1.35	0.02
Others (%)	4.86	0.31	0.2	0.35	0	0	0	0

TABLE I

TRAFFIC CHARACTERISTICS.

Let us study the performance of the *Optimal, Estimated, and Estimated & Periodic algorithms* for the data collected. The parameter settings we use are:

$$\alpha = 1/8, e_I = 1, e_S = 0.1, e_W = 2, e_P = 2, 0.01 \leq \delta \leq 0.1$$

Recall that α is the weight used in equation 2. The value of t_P is derived based on the speed of the interface and packet size (e.g., a 512 byte packet will take 512x8/100 = 40μs on a 100Mbps interface). Note that we only used the *Wake – HABS* transitions from Figure 3 because none of the hosts sleeps and we therefore cannot use Simple Sleep. Figure 6 plots γ vs δ for the four interfaces (top left is the Y – switch, top right M–switch, bottom left is switch – switch, and bottom right is switch – router). Each plot shows the data for the three algorithms for light and heavy load periods (a total of six curves per plot).

1) We observe that the two algorithms that estimate $\bar{x}$ perform almost as well as the Optimal algorithm in all cases. Furthermore, the difference between the Estimated & Periodic and Estimated algorithms is almost non-existent implying that we can use the simpler Estimated algorithm in practice.
2) Sleeping appears to be beneficial in all cases, i.e., $\gamma > 1$. While we expected this for the Optimal algorithm, this result is somewhat surprising for the two algorithms based on estimating $\bar{x}$ since this is the easiest statistic to obtain. One implication of this is that the additions required to switch hardware are simple enough (i.e., determine $\bar{x}$ and implement HABS) to be justified by the large energy savings possible.
3) For smaller δ the gains are much greater. This is expected because the wakeup cost δe_W is correspondingly smaller allowing interfaces to sleep more often (and for shorter periods).
4) We examine the impact of selecting different values for α (equation 2) on these results in Figure 7. We plot the impact of selecting $\alpha = 7/8, 5/8, 3/8, 1/8$ on the Estimated algorithm for the switch – switch and switch – router interfaces during heavy loads. It is interesting to observe that an appropriate selection for α makes a significant difference in performance of the Estimated algorithm (as compared to Optimal).

Our results indicate a clear potential for saving energy in the LAN by adopting simple energy saving algorithms. However, the actual extent of these savings will be affected by the hardware which determines the values of the various constants $e_I, e_S, e_P, e_W, \delta$.

B. Simulation Results

While interesting, the results presented above have two gaps:

1) The results rely on interface-specific traffic traces and do not look at sleeping behavior on interfaces in the network as a whole. For instance, if a port is a forwarding STP port versus a blocked one, the results can vary. Also, the results post-processed the data using the algorithms presented.
2) Thus far, we have only considered Wake – HABS transitions. What happens if the switches do not have the capability for HABS but do have the ability to go into Simple Sleep?

To address these two issues, we incorporate the *Estimated & Periodic* algorithm into Opnet and study the energy savings obtainable using HABS. For Simple Sleep, on the other hand, we used the *Estimated* algorithm with a change described later. In addition, for Simple Sleep, we also calculate the percentage of packets lost.

We use the topology shown in Figure 8 in which there are six switches (sw0 is the root switch). The figure shows the location of the various servers and hosts in the LAN as well. We run the STP protocol in addition to different data streams.

Data for the simulations is generated using Markov Modulated Poisson Process (MMPP) sources at each end-host. For this, we implement a 2-state MMPP model in Opnet The parameters for the MMPP models are selected based on an analysis of traffic traces collected in our own LAN (and used

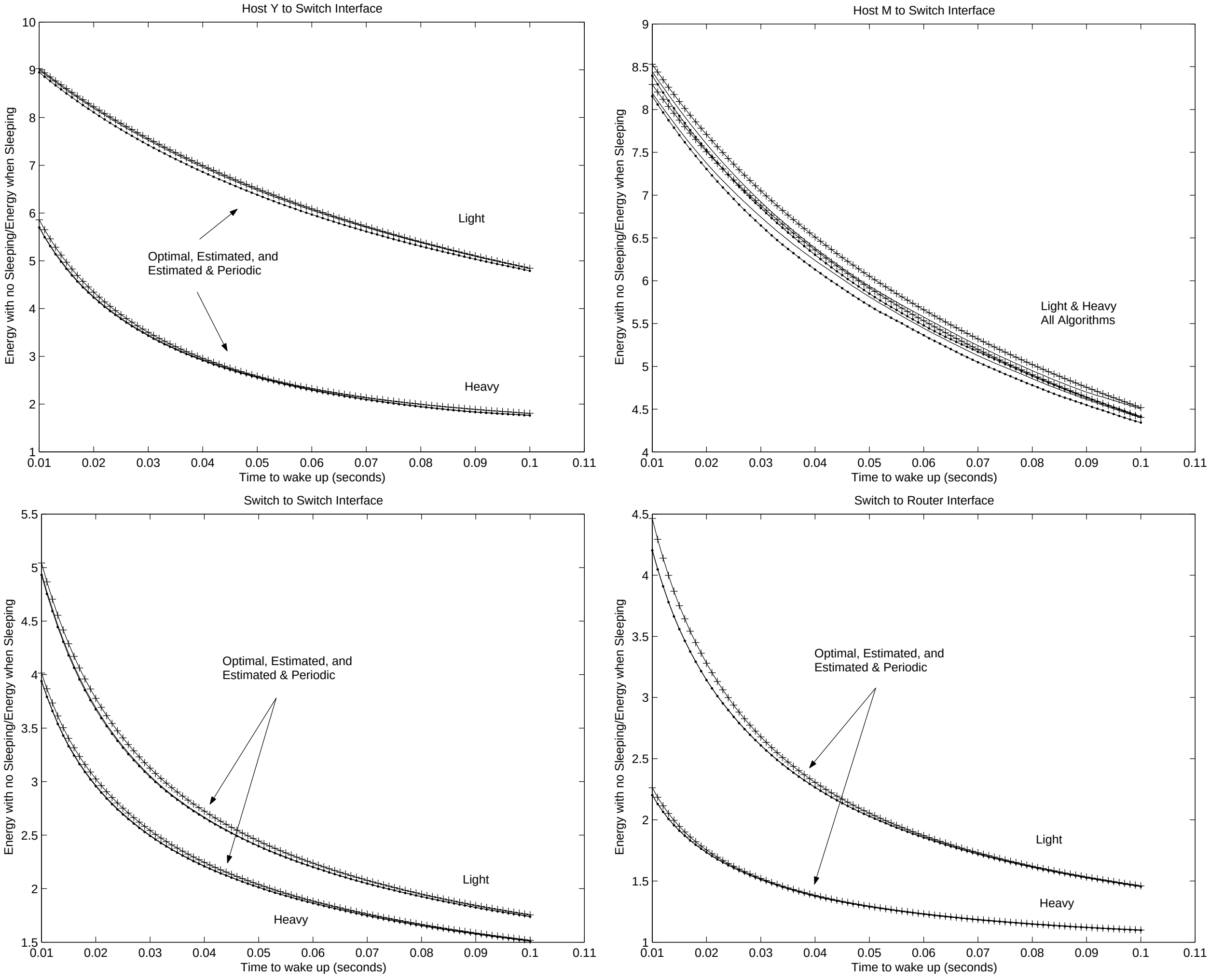

Fig. 6. Energy savings for various interfaces at light and heavy loads.

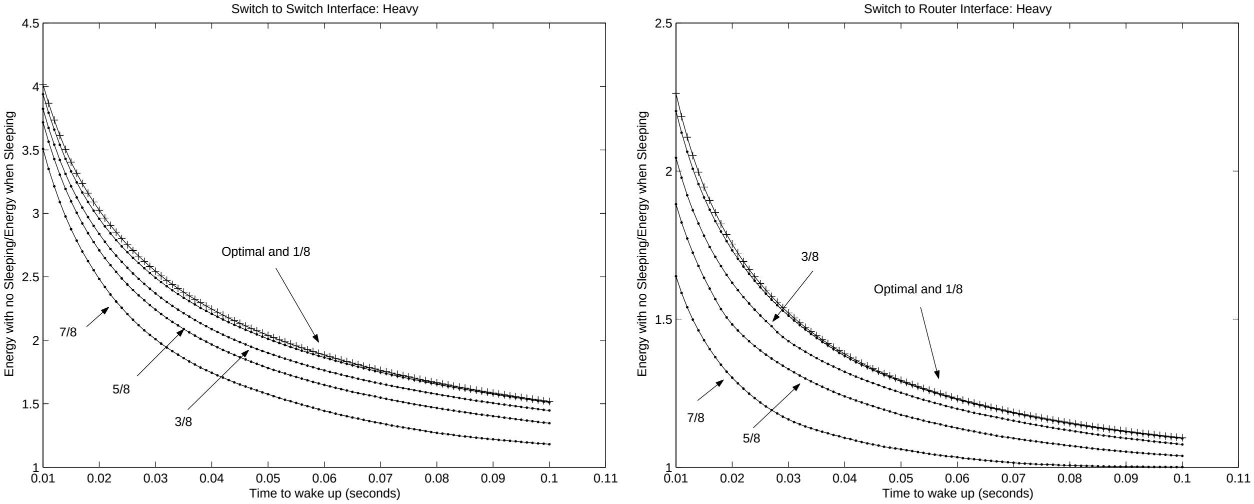

Fig. 7. Effect of varying α in equation 2.

in previous discussions). The *means* are given in Table II. Each end-host generates packets at random intervals which are uniformly randomly sent to all the servers. The simulation time is 2 hours with the initial 30s used for configuring STP.

Data traffic starts after 40s. Packet size used is 512 bytes and all links are 100Mbps.

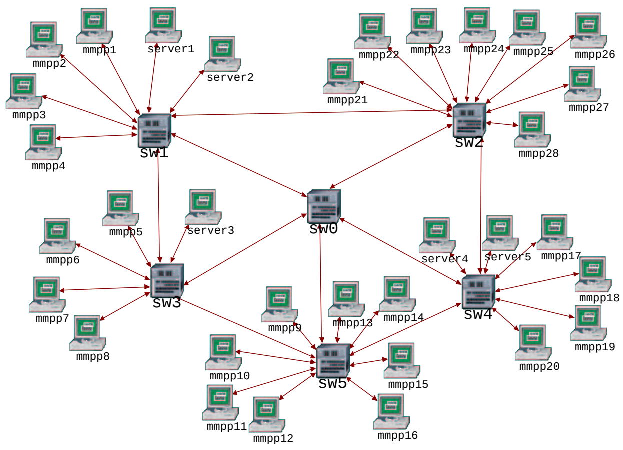

Fig. 8. Simulation topology.

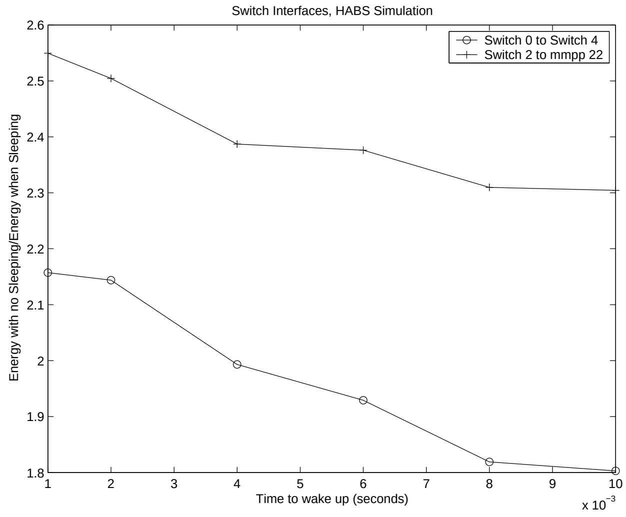

Fig. 9. Energy savings in Simulation using HABS.

Figure 9 plots γ versus δ (using HABS) for interfaces connecting the root switch sw0 to sw4 and the interface connecting sw2 to the end host denoted as mmpp22. Both these ports are forwarding ports in STP. We used a value of $\alpha = 7/8$ for these runs. The values of the e_I, e_P, e_W energy parameters were kept the same for both types, but the transition times and the sleep-to-wake transition energy were different. Since Simple Sleep is a deeper sleep state than HABS, it consumes less power than HABS, but takes longer to transition from sleep to wake state. Thus, we used $e_S = 0.1W$ for Simple Sleep and $e_S = 0.3W$ for HABS. For HABS, transition time values were $\delta = 0.001, 0.002, ...0.01 seconds$ and for Simple Sleep, $\delta = 0.01, 0.02, ...0.1 seconds$.

The energy savings for the sw0–sw4 interface vary from over 2x to 1.8x while the savings range from 2.5x to 2x for the other interface. This difference is explained by the fact the sw0–sw4 sees far more traffic than the other interface since it is connected to the root switch sw0.

Figure 10 plots the energy savings when we use Simple Sleep instead of HABS. The algorithm used here is a variation of the *Estimated Algorithm*. As in that algorithm, the decision to sleep is made based on equation 1. The length of the sleep interval $t_S = \bar{x} - \delta$. We do not set a value for t_I rather, after waking up, the interface remains awake until the next packet is processed after which it uses equation 1 to make a sleep decision again. We see that the interface connecting sw2 to mmpp22 shows between 4.5x and 2x improvement in energy whereas the sw0 to sw4 interface shows 2.2x to 1x improvement. Suprisingly, Simple Sleep seems to perform better than HABS initially, though it degenerates to worse performance than HABS as the time to wake up increases. One of the reasons to explain the lower energy savings for HABS may be that HABS consumes more power during its sleep state, thus leading to lower energy savings overall.The low energy benefits may also lead to fewer sleep cycles since the estimated interval must be large enough to make it worthwhile. Interestingly, the packet loss rates observed were less than 7.5% for both interfaces[3]. The reason the percentage of lost packets decreases with increase in δ is that the interface stays awake more frequently (as per equation 1) and only sleeps when $\bar{x}$ is large.

In summary, we note that using either sleep state (HABS or Simple Sleep) yields large reduction in energy consumption. Simple Sleep, however, does drop packets with the expected consequences for higher layer protocol behavior.

VI. Impact of Sleeping on Protocols and Topology Design

By and large, we can claim that using *HABS does not affect the higher layer protocols in any way*. However, using HABS will result in a certain amount of delay that depends upon the transition time from sleep to wake. It will also add a certain amount of queuing delay due to the accumulation of packets in the buffer while the interface is still waking up. This delay may affect the performance of the network, and must be further examined. Now, if Simple Sleep is the only available sleep option, then what impact will using this sleep state have on these protocols? This is the question we examine next in this section.

For protocols that emit *periodic messages*, e.g.,in STP (Spanning Tree Protocol) configuration BPDUs are typically sent every 2 seconds, we can adjust the sleep timers appropriately and thus, minimize the impact of packet losses due to sleeping. This adjustment must be finely tuned else it can result in significant penalty in terms of energy losses as well as network performance. For example, frequent loss of configuration BPDUs can trigger a recomputation of the spanning tree which would increase energy consumption and result in loss of connectivity during that period. For asynchronous protocols,

[3] Note that we did not use TCP traffic and thus there were no retransmissions due to the lost packets.

Model used at each host				
	State 1 (ON)		State 2 (OFF)	
	Inter-activity time (Poisson)	Duration (Exponential)	Inter-activity time (Poisson)	Duration (Exponential)
High Load	0.1s	0.5s	14.67s	15.0s

TABLE II
MMPP PARAMETERS.

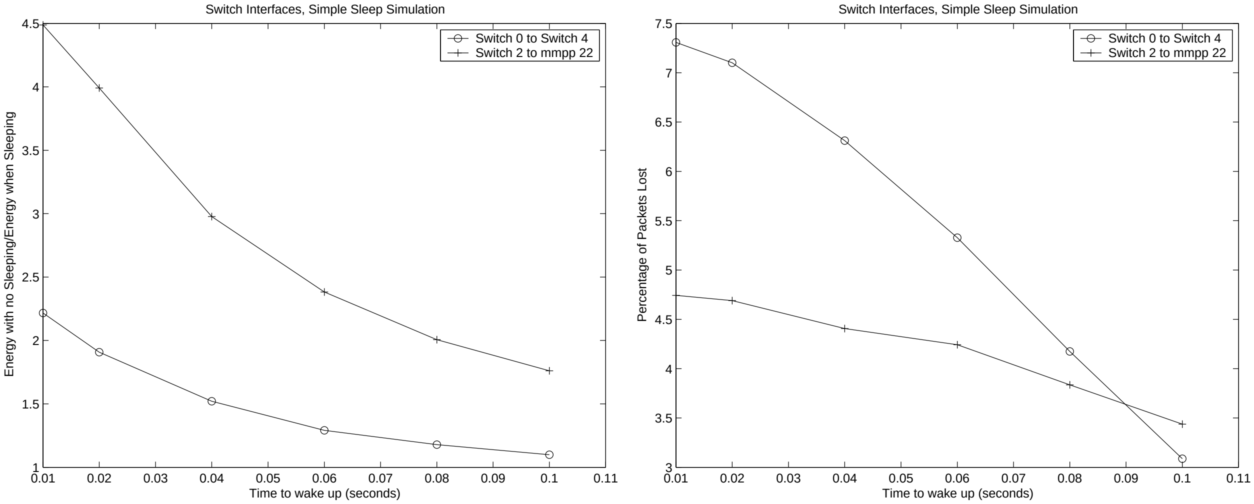

Fig. 10. Energy savings and packet loss using Simple Sleep.

we do have a problem since the sleep timers cannot be adjusted as predictably, thus leading to a possibly greater number of packet losses.

Consider, for example, ARP requests. If the switch port that connects to the destination host is in sleep state then the sleeping port can be woken up by the switch when it detects a broadcast packet directed to the LAN on which the sleeping port resides. This will also mean waking up all other sleeping ports on the same LAN. If, on the other hand, the sleeping switch port is on the forwarding path to other hosts on a different segment of the same LAN then using Simple Sleep will result in the ARP packet being lost. A similar discussion holds for other asynchronous protocols running on top of IP, which includes the impact of lost packets on performance of the TCP protocol.

A. Impact of Network Topology and VLANs on Sleeping

In order to maximize energy savings (using either or both Simple Sleep and HABS), we need to pay attention to the network topology. It is reasonable to assume that the network has several redundant paths (for fault tolerance and load balancing), then it is possible to identify the individual loads during off-peak hours and aggregate the traffic load so as to use only some of the paths and put the rest of them to sleep. However, this can be hard to do if the topology does not allow for aggregation of loads along a certain path due to constraints such as the existence of multiple instances of the spanning tree protocol for each configured VLAN in the network.

For example, in most cases, switches do not have multiple links between them and if they do, only one link would be set to forwarding state by the Spanning Tree Protocol in order to avoid loops. However in certain cases, it is possible to have them both in forwarding state. For example, consider the following simple network shown in Figure 11, with two switches and two links between them. Switch A is connected to hosts 1-10 and switch B is connected to hosts 11-20. Assume, hosts 1,3,5,15,17,20 belong to VLAN 100 and hosts 2,4,6,12,14,16 belong to VLAN 200. Each VLAN runs its own instance of STP. In this case, it is possible that the STP for VLAN 100 may select link 1 to be in forwarding state and the STP for VLAN 200 would select link 2 for the same purpose.

Thus, even if a single link can accommodate the traffic activity for both VLANs, it may not be possible to aggregate traffic on one link due to the way the port states are set up by each STP.

While it is reasonable to use both links when traffic loads justify it, in periods of low traffic activity we could configure the path costs of each link such that only one of them would be selected by all instances of the STP running for each VLAN. This would allow the other link to be put to sleep and thus decrease the overall power consumption of the switch.

VII. CONCLUSIONS

Our study clearly shows that sleeping in order to save energy is indeed a feasible option in the LAN. Our initial idea of using Simple Sleep for host-to-switch interfaces along with Extended ACPI would result in very good energy savings

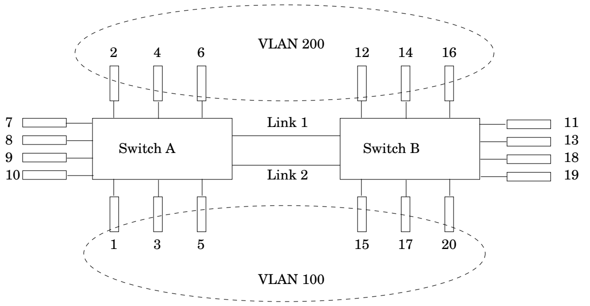

Fig. 11. Impact of VLANs on sleeping.

as well as minimum or even, no packet loss. If appropriate hardware support is provided (in the form of implementing HABS) then the extent of energy savings for other interfaces can be quite significant. Considering the interaction between sleeping and protocols, we note that there is a need to utilize topologies that allow more ports to sleep. Furthermore, since different VLANs can potentially have different spanning trees, the likelihood of a port showing low activity is reduced. We suggest that during periods of low load, the VLANs recompute their spanning trees so that the use the same ports and links as far as possible. This strategy makes sense because, during low load periods, there is no need to do load balancing across links (as is done in most topologies today).

In our future work, we will examine the problem of developing better sleeping algorithms to maximize sleeping. We also plan to study the impact of sleeping on routing protocols as well as higher-layer protocols such as TCP.

References

[1] Prism Power Management Modes, http:// www.intersil.com.

[2] http:// www.xbow.com/ Products/ Wireless_Sensor_Networks.htm

[3] Wassal, A.G.;Hasan, M.A. ”Low-power sytem-level design of VLSI packet switching fabrics” CAD of Integrated Circuits and Systems, IEEE Transactions, on June 2001

[4] B. Chen, K. Jamieson, H. Balakrishnan, and R. Morris, “SPAN: An energy-efficient coordination algorithm for topology management in ad hoc wireless networks”, *ACM MOBICOM 2001*, July 26 – 21, 2001, Rome, Italy.

[5] M. Gupta and S. Singh, “Greening of the Internet”, *ACM SIGCOMM’03*, Karlsruhe, Germany, August 2003.

[6] G. Essakimuthu et al. “An analytical power estimation model for crossbar interconnects”, Technical Report CSE-02-009, Penn state University, Department of Computer Science and Engineering, 2002.

[7] http://www.acpi.info/index.html

[8] L. Feeney and M. Nilsson, “Investigating the Energy Consumption of a Wireless Network Interface in an Ad Hoc Networking Environment”, *Proc. IEEE INFOCOM 2001*, April 22–26, 2001, Anchorage, AK.

[9] M. Kim and B. Noble, “Mobile Network Estimation”, *Proc. ACM MOBICOM 2001*, Rome, Itlay.

[10] D. Langen, A. Brinkman, and U. Ruckert, ”High level estimation of the area and power consumption of on-chip interconnects”, *IEEE Int’l ASIC/SOC Conference*, 2000.

[11] R. Min et al, “Energy-centric enabling technologies for wireless sensor networks”, *IEEE Wireless Communications*, August 2002, pp. 28 – 39.

[12] C. Patel, S.Chai, S. Yalamanchili, D. Shimmel, ”Power constrained design of multiprocessor interconnection networks”, *Int’l Conference of Computer Design*, 1997.

[13] L. Peh and V. Sorteiou,”Dynamic Power Management for Power optimization of Interconnection Networks Using On/Off Links”, *11th Symposium of High Power Interconnects*, 2003.

[14] L. Shang, L. Peh, N. Jha.”PowerHerd: Dynamic Satisfaction of Peak Power Constraints in Interconnection Networks”, *ICS 2003*.

[15] Li Shang, Li-Shuan Peh, and Niraj K. Jha, ”Dynamic Voltage Scaling with Links for Power Optimization of Interconnection Networks”, In *Proceedings of the 9th Symposium on High-Performance Computer Architecture*, pages79-90, Feb. 2003.

[16] Hang-Sheng Wang, Li-Shiuan Peh, Sharad Malik, “A Power Model for Routers: Modeling Alpha 21364 and InfiniBand Routers” (2002).

[17] A. G. Wassal and M. A. Hasan, “Low-power system-level design of VLSI packet switching fabrics”, *IEEE Transactions on CAD of Integrated Circuits and Systems*, June 2001.

[18] W. Ye, J. Heideman, and D. Estrin, “An energy-efficient MAC protocol for wireless sensor networks”, *IEEE INFOCOM 2002*, New York, NY, June 23 – 27, 2002.

[19] T. T. Ye, L. Benini, and G. D. Micheli, “Analysis of power consumption on switch fabrics in network routers”, *Proc. DAC 2002*, pp. 524–529, 2002.

[20] F. Zane, G. Narlikar, and A. Basu, “CoolCAMs: Power-efficient TCAMs for forwarding engines”, *Proc. IEEE INFOCOM 2003*.

[21] James Aweya, IP Router Architectures:An Overview, Journal of Systems Architecture 46 (2000) pp.483-511, 1999.

[22] Intel 21143 PCI/CardBus 10/100 Ethernet LAN Controller datasheet. ftp://download.intel.com/design/network/datashts/27807301.pdf

[23] http://www.cisco.com/en/US/products/hw/routers/ps167/ products_white_paper09186a0080091fdf.shtml

[24] Minkyong Kim, Brian Noble, Mobile Network Estimation, *Proc. 7th annual International Conference on Mobile computing and networking*.