Dynamic Ethernet Link Shutdown for Energy Cconservation on Ethernet Links

Maruti Gupta
Department of Computer Science
Portland State University
Portland, OR 97207
Email: mgupta@cs.pdx.edu

Suresh Singh
Department of Computer Science
Portland State University
Portland, OR 97207
Email: singh@cs.pdx.edu

Abstract—**Recent studies of network traffic utilization on campus networks have shown that data networks remain heavily under-utilized. Yet currently there is little attempt to save energy on network interfaces by using the low power modes available in Ethernet transceivers during periods of inactivity or low utilization. In this paper we design and evaluate a Dynamic Ethernet Link Shutdown (DELS) algorithm that utilizes current technology leading to significant benefits in energy savings with little noticeable impact on packet loss or delay. The algorithm uses buffer occupancy, the behavior of previous packet arrival times and a *configurable* maximum bounded delay to make sleeping decisions. The scheme was evaluated using simulations with inputs generated using a synthetic traffic generator for smooth and bursty traffic. The results show that the percentage of total time that a link can be shut down can be anywhere from 80% to 60% for traffic loads up to 5%.**

I. Introduction

In recent years, several studies have shown that wired data networks on campus remain highly under-utilized [1], [2], [3]. Experiments to measure the difference in power consumed during transmission and idle periods [4], [3] have shown that there is little or no difference in the power consumed between idle and transmitting/receive states. However, there is a difference in energy consumed when using different link rates. [5] shows that the power savings when switching from 1Gbps to 100Mbps are approximately 4W (on desktops) and 0.1W when switching from 10Mbps to 100Mbps. Based on this, a proposal was recently made to the IEEE 802.3 committee outlining a scheme called Ethernet Adaptive Link Rate (ALR) to automatically change the link speed from 1Gbps to 100Mbps or from 100Mbps to 10Mbps in order to save energy during low periods of activity [5]. Such schemes are actually in use in laptops where switching to battery power automatically scales down the link speed to enable longer battery usage [6], but not in desktops. Indeed, this technique is especially applicable in campus area networks where the utilization has been shown to be low and gigabit Ethernet is widely deployed.

In this paper we go one step further and show that it is possible to garner even more significant savings by shutting the links down altogether using the dynamic Ethernet link shutdown mechanism proposed in [7] and further extended here. The reasons for doing this are twofold. Apart from greater energy savings in campus area network traffic, it is also better suited towards taking advantage of networks that remain mostly idle for long periods of time such as home networks. For example, approximately 82 million households in US have high-speed broadband connections [8] that remain connected to the Internet 24x7. While a study has not been done to show how many hosts remain on even during periods of inactivity, the network devices (routers and hubs) at either ends remain completely powered on at all times. DELS, with some alterations for the corresponding link layer protocol is a viable option to obtain significant savings in such cases.

This work was funded by the NSF under award number NeTS-NR: 0435328.

DELS reduces energy consumption in Ethernet LAN switches and the hosts attached to them by shutting down the complex transceiver circuitry depending upon traffic arrivals, buffer occupancy and a bounded maximum delay. While there have been no studies that profile the energy consumption of an Ethernet switch, using available data we estimate that Ethernet interfaces alone can consume up to 20% of total switch power budget. [1].

The rest of the paper is organized as follows. In the next section we cover related work and in III we define the problem statement. Section IV covers the details of the mechanism used in DELS and the changes required in the auto-negotiation mechanism for Ethernet. Section V discusses our evaluation methodology and in section VI we study the algorithms in depth with various different traffic distributions as well as using traces generated by a synthetic self-similar traffic generator. The final section presents our conclusions.

II. Related Work

As mentioned in the introduction, in [3], Christensen at al look at ways of saving energy by changing link speeds in Ethernet links depending on traffic load or by using a low-power proxy device as an alternative to the NIC that saves power by processing layer 2 protocol packets instead of waking up the energy-hungry centrally located high-speed processor to deal

[1]A Gigabit Ethernet transceiver can consume 1-2W depending upon whether it is copper or fiber[6], [9]. As an example, consider a Cisco 3750 Catalyst switch with 24 gigabit copper-based ports and 4 fiber-optic ports. The total maximum operating cost of the switch is 190W[10].

with it. Practically all other energy conservation research has been done in the wireless domain for battery operated devices, particularly for wireless LANs using the 802.11 protocols. There about 90% of the energy is spent during the radio listeningscanning operations [16]. Thus the schemes proposed are mostly centered around minimizing this energy cost. In addition, since 802.11 networks are mostly used in infrastructure mode where the base station is in charge of scheduling and allocating bandwidth, the schemes proposed are typically centrally implemented in order to maximize system capacity and reduce collisions as well as increase energy efficiency. Thus, the work there while useful in understanding demands of network traffic is not directly relevant in the switched Ethernet topology. There has also been considerable amount of work done [17], [18] which deals with conserving energy in servers and server clusters in data centers. These approaches are focused on very specific network traffic loads, i.e web servers whereas our study looks at the traffic in an Ethernet LAN environment.

III. Problem Statement and Sleep mechanism

Currently, there are network interfaces available that can operate in extremely low power modes during which all receiving and transmitting circuitry is shut down and a programmable timer can be used to set the time after which the interface comes back up. There are also low power modes during which no transmission or reception of packets can occur, but the interface is capable of detecting a signal on the wire and responding accordingly. However, this technology is used only in cases where the link is put in such modes when it fails to detect a signal for a given length of time (typically set to 5 seconds) [6] indicating failure. Broadcom and Intel both offer network transceivers that can be used both for hosts and switches that come equipped with such functionality, though not at transition speeds that would actually facilitate a quick transition from a fully operating speed of 1 Gbps to a power down mode. Typically, for gigabit interfaces, the difference in power consumption between full scale operation and these low power modes is 1W . In this paper, we have estimated a transition time of well under 0.5ms, which includes both the transition time as well as the synchronization of the links when the link is re-established. We came up with this number based on information available for wireless radio devices, which are far more complex, yet much more energy efficient due to their use in battery-operated devices as well as the fact that the transition speeds in high-speed fabrics used in Infiniband networks are in the range of nanoseconds as mentioned in [11].

The goal of the dynamic Ethernet link shutdown policy proposed here is thus to intelligently use this link shutdown capability during the idle periods we see in LAN traffic.

Figure 1 shows one half of a full-duplex link. We assume that when the link is off, both the Rx and Tx interfaces are asleep and when the link is on, both the interfaces are powered on. We assume that packets queued for transmission at interface a when the link is off are buffered at A. The key

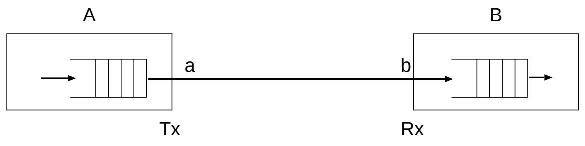

Fig. 1. Simple model for a link.

challenge here is in deciding *when* to turn a link off and for *how long*.

IV. Dynamic Link Shutdown

The DELS algorithm proposes to use the programmable capability of the sleep timer to dynamically change the operating power modes. In this scheme, the interface can either operate at full speed which we refer to as the ‘On‘ state or in the extreme low power mode, referred to as the ‘Off‘ state. The transition between the two states occurs when the transmitting side (upstream interface) nforms the receiving side (downstream interface) using an 802.3 mac frame and receives an acknowledgement (ack) frame in return. Normally, when a link goes down and then comes back up, the auto negotiation protocol is run to, among other things re-synchronize the link. This process takes anywhere from 256ms up to 1s which is much too long, especially for gigabit links so we propose instead a fast exchange of two-way mac frames to re-establish the link in its former operating mode. The upstream interface sends a mac frame with the ”Goto sleep, sleep timer” message and the downstream interface acknowledges this with a ”Ack sleep” message in return. There is no message sent when the interface wakes up, since the wakeup is pre-timed.[2]

A. Mechanism

The DELS algorithm has two parts, namely checking to see if it is feasible to power down the link and if feasible, then computing the length of the sleep period. If the link is already down, then it buffers the packet for later transmission. The feasibility of turning the link off is determined by the number of packets in the buffer and the mean inter-arrival time of w packets, where w is the size of a sliding window of most recent inter-arrival times. If the current buffer size is low enough and the mean inter-arrival time greater than the transition time, the algorithm decides it is feasbile to sleep. The algorithm then computes the length of the maximum sleep interval given a maximum bounded delay and the maximum buffer size as described in the following section. The algorithm overhead is composed of checking the feasibility of turning the link on/off and then computing the length of sleep. The feasibility check is made upon the arrival of a packet during the On state. However, the computation of sleep is done only if the check succeeds.

Figure 2 shows the finite state machine diagram for the upstream interface. The following lists the various parameters,

[2]It must be noted that the transition time must accomodate the time required for re-synchronization of clocks before any frames can be sent on the link again.

timers and variables used. This algorithm runs at the upstream interface of a link. It is invoked whenever the buffer occupancy falls below a threshold (we use a value of 10%).

- B is the output buffer size at the upstream interface.
- w is the number of the most recent inter-arrival times. When a new packet arrives, the oldest inter-arrival time is discarded and the new inter-arrival time is added to the list
- λ is the mean inter-arrival time
- $\tau = \alpha B$ is the buffer occupancy threshold and $\alpha < 1$ (we use $\alpha = 0.1$ in our experiments)
- m is the number of packets in the buffer
- δ is the transition time from off to on state
- $stimer$ is the sleep time programmed in the timer
- t^{max} is the maximum sleep time configured

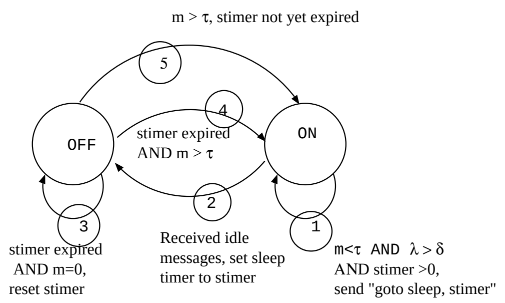
m > τ, stimer not yet expired
5
4
OFF
stimer expired AND m > τ
ON
2
3
1
stimer expired AND m=0, reset stimer
Received idle messages, set sleep timer to stimer
m<τ AND λ > δ AND stimer >0, send "goto sleep, stimer"

Fig. 2. State machine for the upstream link

In figure 2, transition 1 takes place when the interface detects conditions suitable for transitioning to Off state, based upon the buffer occupancy B, the mean λ and t^{max} and sends a frame containing the "Goto sleep, $stimer$ message to the downstream link. On receiving a frame containing the ack message, it transitions into the off state, $stimer$ is set with the computed sleep value. We explain how this value is derived later in this section. In transition 3, the interface may remain in the Off state, when the sleep timer expires and the buffer is still empty. The downstream link on the other hand, will come back up on expiration of its $stimer$, but if it does not detect any signal on the wire for time δ, it will reset its sleep timer to $(stimer - \delta)$ and go back to Off state. The transition from Off state to On state can occur for two different conditions. In transition 4, the condition is an expiration of the sleep timer. In transition 5, the condition is the exceeding of the buffer threshold that causes the upstream link to make an unscheduled wake-up and start transmitting the normal active signal on the wire causing the downstream link to wake up as well. Note the transition to On state does not use an explicit message from the sender.

B. Computing the Sleep timer setting

If $m \leq \tau$ then the upstream interface examines the possibility of putting the downstream interface to sleep for time $stimer$. This is done by estimating the number of packets n that will arrive in time $stimer$ into the upstream interfaces buffer. The goal is to ensure with a high probability that the total number of packets $n+m < \alpha B$. In words, the probability that the total number of packets in the buffer will exceed some threshold of $\tau = \alpha B$ is small.

This computation is done by assuming that packet inter-arrival times are iid Exponential Random variables within a small window([12] shows that inter-packet rates are piecewise Poisson). Thus, if $X_1, X_2, \ldots X_n$ are Random variables for consecutive inter-packet times then $X = \sum X_i$ has a Gamma distribution. We find maximum t such that,

$$P[X \geq t] \geq 0.9$$

In other words, we find the maximum t such that the probability of more than n arrivals is less than 10%.

If $t > \delta$ then the upstream interface puts the downstream interface to sleep for time $\min\{t - \delta, t^{max}\}$ where t^{max} is the maximum amount of time that any interface is allowed to sleep.

We use a value of $w = 5$ (number of most recent inter-arrivals considered) in the experiments due to a very high variability in the experiments. We expect that this algorithm will result in output traffic (i.e., on the output of the upstream interface) that is more bursty than the input traffic. This is because the upstream interface buffers packets and then emits them at the maximum rate when the downstream interface is awake.[3]

V. Evaluation Methodology

We evaluated the algorithms by implementing the link shutdown policy in Matlab for the upstream interface. The input to the interface is in the form of a text file containing inter-packet arrival times and packet size in bytes. The inter-packet arrival times were generated using a synthetic self-similar traffic generator as well as arrivals from different distributions for different loads. The packet sizes were based upon the packet size distribution of our own traces collected on campus and follow the standard bimodal distribution.

For comparison, we also implemented the Delay-Optimal algorithm. The Delay-Optimal algorithm basically always returns the optimal value of sleep timer since it knows the arrival times of all future packets and is thus able to go to sleep without resulting in any delay or packet loss. We used the following *metrics* and variable *parameters* in our evaluations. The metrics used are:

1) Total amount of time the link is off
2) Average response time or delay
3) Number of packets dropped
4) Additional buffer required to match the packet loss behavior when the interfaces do not sleep

[3] We also note that the behavior of this algorithm can be modeled as a single server queue where the server goes on vacation for t when the probability that the buffer will overflow is less than 10%. In this paper we do not develop a formalism for this model due to space limitations, but is currently part of our future work.

A. DELS Parameter values

- Buffer threshold, α: The amount of time spent sleeping is computed such that the number of packets received during the time interval would not exceed a certain buffer threshold. This threshold was set to 10% since it got the best trade-off between sleep times, delay and packet loss, based on our studies done with traces earlier in [7]. If the threshold is set too high, then the sleep time t would be higher since more packets can be accommodated in the buffer, but would also result in possible packet loss or large delays. On the other hand, if it is set too low, then the number of decisions to sleep would be lowered, thus decreasing the chance of packet loss, but also minimizing the sleep time.
- Size of window for last w packet arrivals: We chose a window size of 5 in order to decrease the overhead of storing large number of values and also due to the fact that Ethernet traffic normally shows a great amount of variability.
- Buffer Size, B: The buffer size B was set to 256KB to provide adequate buffering in case of a very large burst and to take into account that while the link speed is 1Gbps, it is possible for packets to arrive at higher speeds especially at the switch port if multiple hosts talk to the same host in a short amount of time. Indeed, observe that even a 0.5ms burst of maximum sized packets at 200% capacity will easily overwhelm a smaller buffer of size 32KB. Such bursts have been observed at the switch port in our campus traffic. Moreover, 256KB is supported by gigabit NIC cards today.
- Delay, t^{max}: The maximum delay we tested our scripts for was set to 2.5ms. This allows for reasonable sleep times while providing a maximum bounded delay to the user-level applications.

The parameters above are selected for gigabit links and would be different in case of a 100Mbps link, but can be easily derived, once the transition time and the maximum delay are known.

The variable parameters we use in our study include the following:

1) Network traffic loads: It is interesting to find at what loads can we expect to gain significant energy savings. We test the DELS algorithm for loads varying from 1% to 41% in the case of self-similar traffic and from 1% to 25% in the case of traffic following general distributions.
2) Distributions: The traffic was chosen from the following four different distributions, namely Uniform Random, Exponential, Pareto and Weibull. While Ethernet traffic has never been characterized as uniform Random, it is interesting to know how the algorithm would perform given this kind of traffic. The means for each distribution were varied in order to generate the appropriate load in Matlab, and the shape parameter for Pareto distribution was selected as 0.8.

We used a fixed value of $\delta = 500\mu s$ for the sleep to wake transition time for the interfaces for all experiments. This value is based on proprietary transition time information for electrical circuits of the type used in Gbps NIC cards.

VI. Experimental results

A. Stationary Distributions

We begin by evaluating DELS with smooth stationary distributions to determine how the algorithm responds under different network load conditions. We tested it with 4 different distributions, namely uniform Random, Exponential, Weibull and Pareto, which were generated in Matlab. We also ran the Delay-Optimal algorithm for the same traces for comparison. The uniform Random distribution numbers were generated from an interval ranging from 10 to 800 microseconds for the 1% load and then lowering the upper bound to 26 for the 25%load. For the Pareto distribution, the shape parameter was set to 0.8 and the location θ and scale σ parameters were varied from 75 to 3.5 microseconds to generate loads from 1% to 25% respectively. For the Exponential distribution, the parameter μ ranged from 420 to 17 microseconds and for the Weibull distribution, we used similar values of location as Exponential and the scale parameter was set to 1.

Figure 3 shows the sleep times obtained using the Delay-Optimal algorithm for the different distributions for loads 1%, 5%, 10% and 25%. We observe that the sleep times decrease rapidly for all the loads beyond 1% utilization for all distributions except for the Pareto distribution. Since in the Pareto distribution, a significant number of inter-arrival times will be centered around the largest inter-arrival times, it makes sense that the Delay-Optimal would perform well for Pareto distribution. In comparison, the uniform Random algorithm is spread around a small inter-packet arrival time, thus leading to shorter sleep times. The Exponential and Weibull distributions are very similar in nature and are somewhere between Random and Pareto distributions.

Figure 4 shows the sleep times obtained using the DELS algorithm. In contrast to the Delay-Optimal, we find the largest sleep times for uniform Random due to the clusters of packets around a small mean and the least for Pareto, with it's widely varying inter-packet times. However, in all case the DELS performs better than the Delay-Optimal. This is because the optimal algorithm only allows sleeping when the buffer is empty and when there is no increase in delay or probability of packet drops. On the other hand, On/Off sleeps even when there are packets in the buffer thus resulting in higher delay than the optimal. Note that optimal has a non-zero delay because when there are bursty arrivals, packets do suffer buffering delay. Further, in current switches, packets suffer delays due to QoS guarantees for some flows.

The delay graphs for Delay-Optimal Figure 5 and DELS Figure 6 show the tradeoff between sleep times and delay for Delay-Optimal and DELS algorithm. The packet loss in all cases during this run were zero.

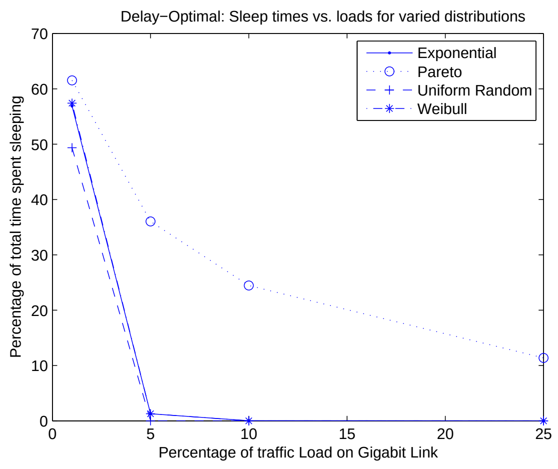

Fig. 3. Delay-Optimal:Variation of sleep times with load and traffic characteristics

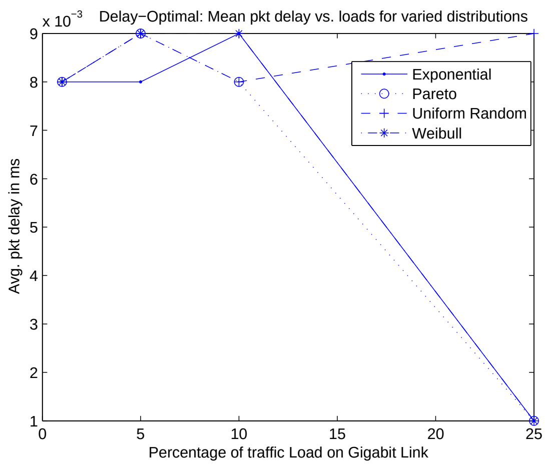

Fig. 5. Delay-Optimal:Variation of delay with load and traffic characteristics

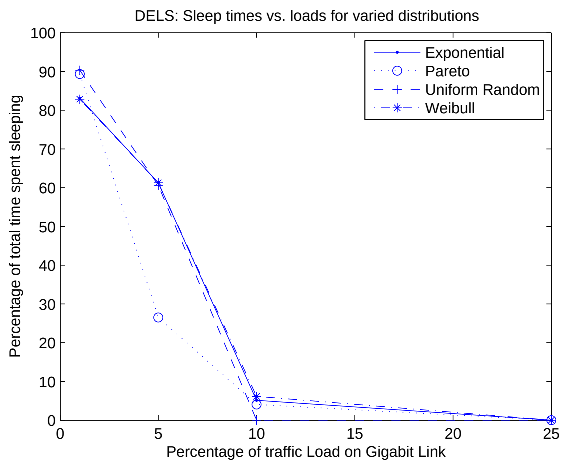

Fig. 4. DELS:Variation of sleep times with load and distributions

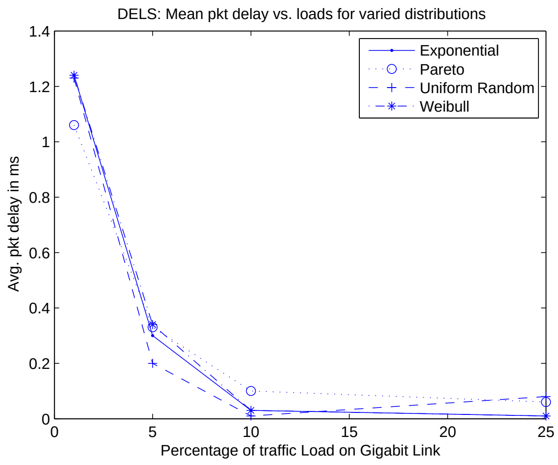

Fig. 6. DELS:Variation of delay with load and traffic characteristics

B. *Self-similar traffic*

We generated synthetic traffic using a synthetic self-similar traffic generator from [13]. Self-similar traffic is characterized by the Hurst parameter, referred to here as the H value, which is used to indicate the degree of autocorrelation in a dataset over a given time scale. If there is a high degree of autocorrelation between the elements of a dataset over all timescales, the process is said to be self-similar. This generator uses the technique described by [14] to generate self-similar traffic, namely the generator multiplexes large number of Pareto distributed streams of on and off periods. The traffic we use for our experiments was generated by multiplexing 256 streams with a mean burst size of 4KB each. The technique is described in greater detail at [15]. We used the tool to generate half a million packets each for H values ranging from 0.5 to 0.9 and varied the load level from 1% to 40%.

Figure 7 plots the variation of observed sleep times with varying loads and H values. Here, we see a clear relationship between the sleep times and load, namely, the sleep times decrease with increasing loads. Comparing the sleep times observed using DELS with the Delay-Optimal case Figure 8, we find that the sleep times for DELS are consistently greater. The explanation for this is similar to that for the stationary distributions.

We also find that corresponding to the sleep times, the packet delay is much higher than the optimal for lower loads. Table I lists the mean packet delay for a specific value of H for different loads for both DELS and Delay-Optimal. Note that the maximum packet delay seen for DELS, i.e. 1.02ms, is within bounds of acceptability. As the load increases, the delay values decrease and become closer to the optimal values.

However there seems to be no clear relationship between H

	Load 1%	**Load 5%**	**Load 10%**	**Load 25%**
DELS	1.02ms	0.5ms	0.3ms	0.03ms
Delay-Optimal	0.02ms	0.01ms	0.01ms	0.01ms

TABLE I
MEAN PACKET DELAY FOR DIFFERENT LOADS, $H = 0.8$.

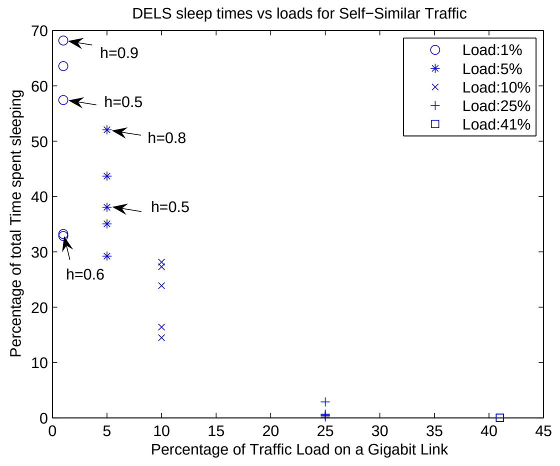

Fig. 7. DELS:Variation of sleep times with load and H on self-similar traffic

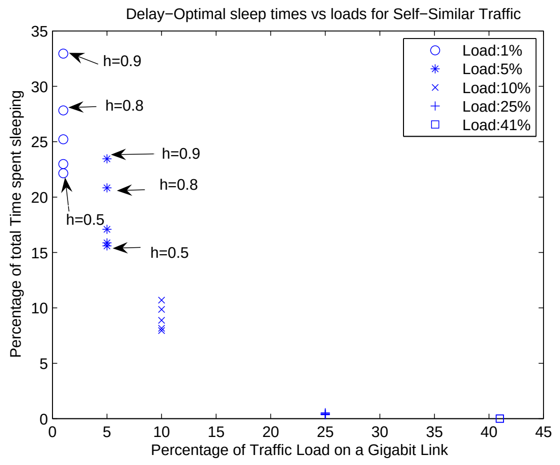

Fig. 8. Delay-Optimal :Variation of sleep times with load and H on self-similar traffic

values and sleep times observed for the DELS case, though in both cases we find that the sleep times are highest for the larger H values of 0.8 and 0.9 as compared to 0.6. It's not entirely clear why this may be so. Interestingly, the optimal case shows a much clearer relationship where the sleep times increase with increase in H. One reason may be that for larger H, the traffic is composed of longer Off periods scattered among more frequent but shorter bursts, thus giving more opportunities for sleep.

We also observe that loads beyond 10% do not yield significant sleep times. There were no packet drops for all the traces.

VII. Conclusions

Based on the analysis presented here, we see that energy savings due to DELS are significant for loads up to 5% in most cases, even extending to 30% in some cases for 10% utilization. However, beyond 5% traffic loads, the percentage of time spent sleeping gradually diminishes enough to be of little significance. We also found that the mean packet delay stays within reasonable bounds of up to 1ms which is not very noticeable at higher layer applications. Also, this delay is not propagated at every hop of the packet's path, since we expect DELS to be deployed between hosts and first-level LAN swithces alone.

For self-similar traffic, there seemed to be no relationship between the sleep times obtained and H, but the sleep times were very closely related to the load. As the load increased, the sleep times decreased. Thus, the DELS algorithm is capable of significant power savings with no discernible impact on packet loss or delay.

References

[1] A. Odlyzko, "Data networks are lightly utilized, and will stay that way," *Review of Network Economics*, vol. 2, Issue 3, no. Issue 3, pp. 210–237, 2003.

[2] M. Gupta and S. Singh, "Greening of the internet," in *ACM SIGCOMM*, Karlsruhe, Germany, August 25 - 29 2003.

[3] C. Gunaratne, K. Christensen, and B. Nordman, "Managing energy consumption costs in desktop pcs and lan switches with proxying, split tcp connections, and scaling of link speed," *International Journal of Network Management*, vol. 15, no. 5, pp. 297–310, Sept/Oct 2005.

[4] M. Gupta, S. Grover, and S. Singh, "A feasibility study for power management in lan switches," in *IEEE ICNP*, Berlin, Germany, October 5 - 8 2004.

[5] C. Gunaratne, K. Christensen, and S. Suen, "Ethernet adaptive link rate (alr): Analysis of a buffer threshold policy," in *IEEE Globecom*, 2006.

[6] Intel, "Intel website." [Online]. Available: http://www.intel.com/

[7] M. Gupta and S. Singh, "Using low power modes for energy savings in ethernet lans," in *IEEE Infocom*, 2007.

[8] J. B. Horrigan. (2006, May) Broadband adoption in the united states. [Online]. Available: http://www.pewinternet.org/pdfs/PIP_Broadband_trends2006.pdf

[9] Broadcom, "Broadcom website." [Online]. Available: http://www.broadcom.com/

[10] Cisco, "Cisco website." [Online]. Available: http://www.cisco.com/

[11] E.-J. Kim, G. Link, K. H. Yum, N. Vijaykrishnan, M. Kandemir, M. J. Irwin, and C. R. Das, "A holistic approach to designing energy-efficient cluster interconnects," *IEEE Trans. on Computers*, vol. 54, no. 6, pp. 660–671, June 2005.

[12] T. Karagiannis, M. Molle, and M. Faloutsos, "A nonstationary poisson view of internet traffic," in *IEEE INFOCOM 2004*, 2004.

[13] G. Kramer, "Generator of self-similar traffic, version 3." [Online]. Available: http://wwwcsif.cs.ucdavis.edu/ kramer/code/trf_gen3.html

[14] W. Willinger, M. S. Taqqu, R. Sherman, and D. V. Wilson, "Self-similarity through high-variability: statistical analysis of ethernet lan traffic at the source level," in *SIGCOMM '95: Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication*. New York, NY, USA: ACM Press, 1995, pp. 100–113.

[15] G. Kramer, "On generating self-similar traffic using pseudo-pareto distribution, technical brief." [Online]. Available: http://wwwcsif.cs.ucdavis.edu/ kramer/papers/self_sim.pdf

[16] C. E. Jones, K. M. Sivalingam, P. Agrawal, and J. C. Chen, "A survey of energy efficient network protocols for wireless networks," *Wireless Networks*, vol. 7, pp. 343 – 358, 2001.

[17] R. Sharma, C. Bash, C. Patel, R. Friedrich, and J. Chase, "Balance of power: Dynamic thermal management for internet data centers," Hewlett Packard, Tech. Rep. HPL-2003-5, Feb 2003.

[18] C. Lefurgy, K. Rajamani, F. Rawson, M. Kistler, and R. Keller, "Energy management for commercial servers," *IEEE Computer*, vol. 36, no. 12, pp. 39 – 48, Dec 2003.