Putting the Cart Before the Horse: Merging Traffic for Energy Conservation

Suresh Singh and Candy Yiu, Portland State University

ABSTRACT

Reducing energy consumption in the Internet has become an increasingly important goal recently. Previous work on reducing energy consumption has primarily looked at either changing link rates or putting interfaces to sleep. Due to the unpredictable nature of traffic, the energy savings achieved have been modest, do not scale, and incur losses and delay. This article proposes a different approach to the problem, which involves aggregating traffic from multiple input links prior to feeding them to the switch interfaces. The main results we obtain are that energy consumption, measured as fraction of interfaces that are awake, scales linearly with load for all loads and the algorithms are fully deterministic yielding zero packet loss.

INTRODUCTION

Concerns over energy consumption in the Internet have prompted significant research activity in the past few years with the goal of achieving *energy proportionality*; that is, energy usage that scales with loading. Previous research work on this topic can be partitioned into two broad categories: work that examines the problem from the network level where energy consumption is part of a route optimization formulation [1, 2], and work that looks at the individual link or switch and uses sleeping or rate adaptation to save energy. The results at the link level demonstrate energy savings for loads less than 30 percent albeit with some packet loss and increased delay. The current article develops a novel solution for *linear scaling* of energy usage with all loads *without any packet loss*. The key idea is *traffic aggregation* via a *merge network* on the input to switches so that we can maximize the number of interfaces put to sleep. This network merges traffic from N input links and feeds them to $K \leq N$ input interfaces on the switch, thus enabling $N - K$ interfaces to be powered off. K scales with load such that packet loss is zero. We evaluate our solution using simulated traffic as well as traffic traces collected at a LAN switch. All cases show linear scaling with zero loss.

The remainder of the article is organized as follows. The next section summarizes related research in energy savings at the switch level. We describe our approach for saving energy, and an algorithm that uses the merge network architecture is presented. The results of our performance study are given, and then we conclude this article.

RELATED WORK

There have been many studies aimed at understanding the energy consumption in various parts of the core Internet and the access networks. Reference [1] provides a study of two Cisco routers, a GSR 12008 and a 7507. In the case of the GSR, the chassis alone consumes 200 W of power. Adding a network processor and switch fabric increases the power draw to 500 W. Adding two four-port Gigabit Ethernet cards, increases the cost to 650 W. Broadly, the energy cost splits as chassis 30 percent, fabric 46 percent, and interfaces 24 percent. In the ideal case, if energy usage of the interfaces and fabric scales with load ρ, the energy consumption of these devices could be written as $200 + \rho \times 450$ W. In this article we show how the energy cost of the interfaces can scale linearly with load. We believe that the switching fabric's cost can also scale linearly with load given that $N - K$ out of N interfaces are off. For instance, [3] provides an algorithm for powering off idle portions of a fabric dynamically.

Approaches for saving energy at the level of routers and switches have focused on exploiting periods of link inactivity to either put interfaces to sleep [4] or use loading models to change the data rate on the interface [5–7]. The results show that for loads up to 30 percent of link capacity, sleeping is possible. However, the energy savings fall rapidly and reach zero beyond these loads. The reasons are twofold: poor traffic prediction pushes us to make very conservative sleep estimates (for fear that packets will be lost at the upstream buffers), and the transition time of the interfaces to go from sleep to wake state has a dramatic impact on sleep times. If the transition time can be made close to zero, we begin to see energy savings even at loads up to 75–80 percent. Similar results hold for the adaptive link rate schemes.

RE-ARCHITECTING THE SWITCH

A fundamental problem we face in putting interfaces to sleep or changing their rates is the unpredictability of traffic on that interface.

[1] *Not all inputs may have packets at the same time, and multiple packets may be destined for the same output but the routing is still deterministic.*

0163-6804/11/$25.00 © 2011 IEEE

Indeed, as we noted, these mechanisms show no energy savings even for loads as low as 30 percent because of the unpredictability of traffic arrivals. Our approach to this problem is to use traffic aggregation on the input to a switch in order to better use the resources. Table 1 summarizes the loading statistics for eight switch interfaces in our network. We see that the loading of individual interfaces can be very small. Indeed, the aggregate load of all these interfaces combined is 1 Gb/s, which means that, in the ideal case, all the load could be handled by one interface. The traffic pattern of P1 is very bursty with short intense periods of over 250 packets/s followed by virtually no traffic for extended periods. The servers show more steady load, and in some instances (e.g., P5) we can fit a piecewise Poisson model.

Figure 1a shows a typical switch architecture, while Fig. 1b illustrates the idea of merging traffic. In a N-port switch, each interface is connected to one end-host or the interface of some other switch. We propose the idea of merging data streams *before* feeding them to the switch. In Fig. 1b, we place an $N \times K$ merge network before the switch input interfaces. K is the number of switch input interfaces that are powered on. Thus, the traffic from the N input links is aggregated together and then fed into the K interfaces, allowing $N - K$ interfaces to sleep. It is clear that K should be a function of load and will change dynamically.

Interface	Type	Raw	Utilization
P1	Workstation	0.7 Mb/s	0.07%
P2	DMZ	304 Mb/s	30%
P3	DNS	4.96 Mb/s	0.496%
P4	Server	112 Mb/s	11.2%
P5	Server	352 Mb/s	35.2%
P6	Server	96 Mb/s	9.6%
P7	LACP	22.4 Mb/s	2.2%
P8	Server	110 Mb/s	11%

Table 1. *Captured traces (11 am, Monday).*

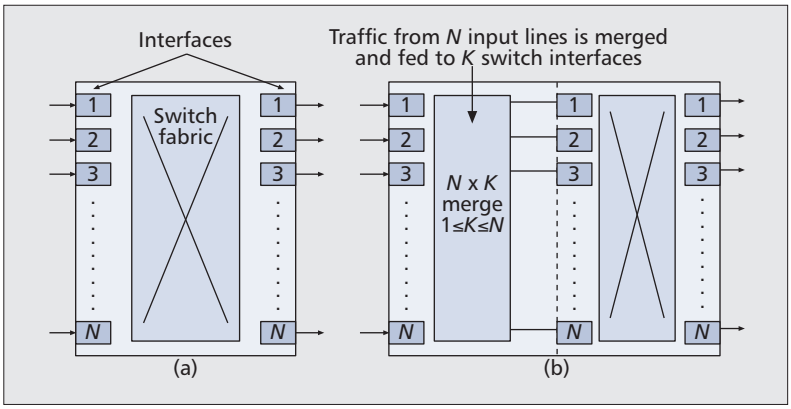

Figure 1. *Modified switch architecture: a) typical; b) with aggregation.*

Architecture of the Merge Network

The function of a merge is to send N input links to some $K \le N$ interfaces on the switch. The challenge in designing such a merge is the need for flexibility of assignment, and for routing packets between the incoming lines and switch interfaces at low energy cost. We note that in the past a great deal of research has been done in the general area of interconnection networks. However, the model for routing there is quite different from the one we have here. In the typical interconnection network model, N inputs are to be routed to N outputs, but the output for each packet is fully deterministic. This is unlike our model where up to N input links need to be routed to a subset of switch interfaces. The routing is non-deterministic in the sense that if the number of packets arriving at N input links is less than or equal to K, no packet is lost regardless of which input link the packets come on, and the packets can be sent to any of the awake interfaces. Another important difference is that in the merge network, the routing is done entirely in analog; that is, the packets are not received and then forwarded. The reason is to keep the complexity and hence the energy cost down.

Consider, for example, $N = 4$ and $K = 2$ — for ease of explanation we label the input links using letters and the input interfaces to the switch using numbers. So input lines (a, b, c, d) are routed to input switch interfaces (1, 2). In this case, packets arriving on any of a or b or c or d can be routed to *either* 1 or 2, and this choice is dynamic. If one or two packets arrive, none of them is lost; but if three or four packets arrive, only two get sent to switch interfaces with the others lost. The key idea behind traffic aggregation is that the number of interfaces K is changed based on load. Therefore, in this example, the merge network needs to support all of 4 × 1, 4 × 2, 4 × 3, and 4 × 4 merges.

In order to enable this form of non-deterministic routing in the $N \times K$ mesh, we require a special hardware element we call the *selector* whose functional behavior is shown in Fig. 2a. There are two incoming lines and two outgoing ones. When there is just one packet arrival on either incoming line, the packet is sent out on the solid outgoing line as shown in Fig. 2a. However, if two packets arrive at the two inputs, the earlier one is sent along the solid outgoing line while the later one is sent out along the dotted line. If the links have slotted transmissions and packets arrive simultaneously, we assume some static ordering (say based on incoming line numbers) to determine which packet gets sent out over which outgoing line.

We developed a prototype of a selector for Cat 5e Ethernet wiring using four broadband absorptive single throw switches with appropriate transistor logic. The design of the selector is discussed in [8], but we note that the design we use is far from optimal and was done primarily to demonstrate feasibility.

To illustrate how the selector can help us build a merge network, consider the 4 × K exam-

ple above. As before, label the interfaces 1, 2, 3, and 4, and the corresponding incoming lines *a*, *b*, *c*, and *d*. Figure 2b shows a merge network that supports all possible merge combinations. The solid line emanating from each selector is the default output, while the dotted line is the deflection route. To configure the network to perform a 4 × 1 merge, we simply shut off interfaces 2–4. Thus, packets arriving at those interfaces are dropped and only one packet, the one that is routed to interface 1, makes it through. Similarly, to implement a 4 × 3 merge, we simply shut off interface 4; thus, three packets make it to interfaces 1–3, while the fourth, if there is a fourth, will be sent to interface 4 and dropped. An important observation about the merge network is that it is *passive* and does not require any logic elements. Therefore, its energy consumption is much lower than the interconnection networks that make up the internal switching fabrics of high-speed switches.

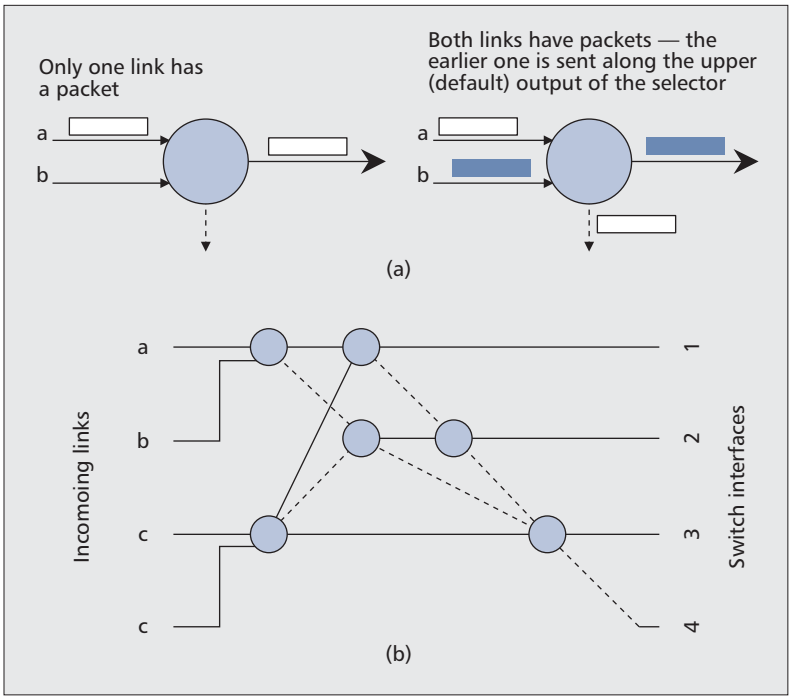

Figure 2. *Example of a merge network: a) logical operation of a selector; b) example of a merge network.*

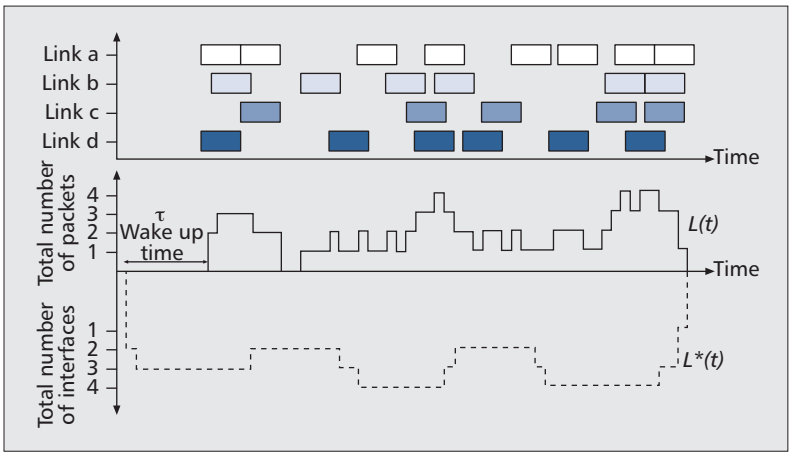

Figure 3. *Optimal aggregation of packets.*

The general structure of an $N \times K$ merge network is a simple generalization of the one shown in Fig. 2b. A log N depth binary tree made up of selectors gives us an $N \times 1$ merge. Next, take all the $N - 1$ deflected outputs (dotted line in Fig. 2a) of all the selectors and form a binary tree with those to get an $(N - 1) \times 1$ merge. This process iterates to create a complete merge network.

An important question that arises is that of the optimality of the merge network as well as measuring its complexity. The complexity of the merge network can be specified by two numbers: the *depth* of the network and the total *number* of selectors used. For the network shown in Fig. 2b, the depth is 4 while the number of selectors used is 6. Indeed, we can prove that the minimum depth of an $N \times N$ merge network is $\log 2 N + N - 2$, and the number of selectors needed is $N(N - 1)/2$.

Deterministic Traffic Aggregation Algorithm

In this section we begin by defining "optimality" of merging and then describe a zero loss aggregation algorithm that scales energy usage (as measured by the number of on interfaces) linearly with load.

A key parameter that affects the amount of sleeping possible is τ, the time to wake up a sleeping interface. If τ is very small, interfaces can be woken up just in time to process a packet (for the optimal case), thus resulting in perfect linear scaling of energy cost with load. The essential idea behind our definition of optimality is, given perfect traffic prediction, interfaces are woken up τ time before they need to process a packet. The energy cost therefore includes the τ wakeup time. Consider the example in Fig. 3 where we have four incoming links with packet arrivals as shown. At the bottom we show the load $L(t)$ as a function of time (solid line plot). Given a nontrivial value of τ as indicated in the figure, we can plot the number of interfaces that need to be awake or be woken up as a function of time $L^*(t)$ (dotted line plot). Note that initially we need two interfaces to process the two packets coming along links *a* and *d*; therefore, two interfaces need to be woken up τ time previously. This is why the bottom dotted line plot starts with a value of 2 for the number of interfaces. However, a short time later we need to initiate wakeup for a third interface to process the packet arriving along link *b*. After these three packets have been processed, we only need two open interfaces; therefore, the idle interface can be put back to sleep (this operation is assumed instantaneous). We denote the instantaneous load by $L(t)$ (the solid line curve in the figure) and the number of interfaces either awake or in the process of waking up as $L^*(t)$ (the dotted line curve). The amount of energy consumed is the area under $L^*(t)$.

Deterministic Aggregation Algorithm

The key idea we use here is to perform traffic shaping at the upstream interfaces by buffering packets for a short amount of time prior

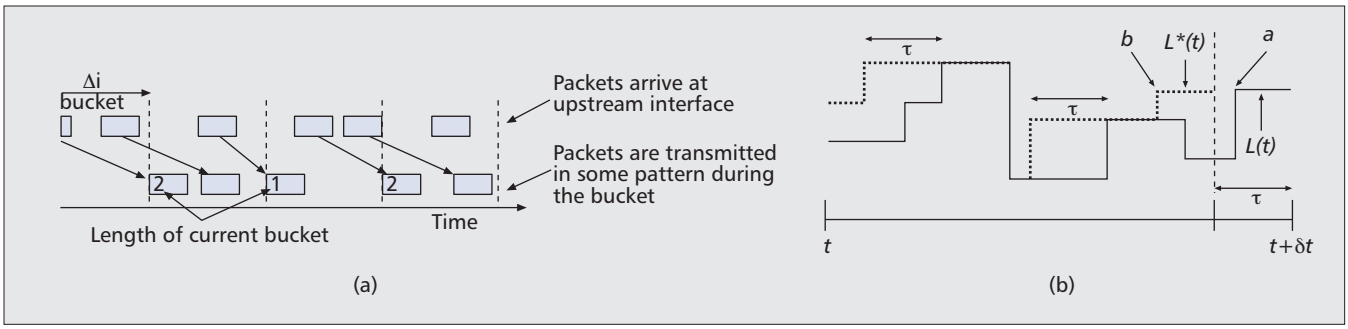

Figure 4. *Traffic aggregation algorithm: a) Δ aggregation of packets at upsteam interfaces for link* i*; b) computing* L* *given* L.

to transmitting them in an appropriately shaped burst. If the downstream merge network knows the expected load for the next burst for each link, it can intelligently open an appropriate number of interfaces and guarantee zero packet loss. We call the buffering interval a *bucket*. The size of the bucket will depend on the buffering capability of the upstream interfaces as well as on the maximum acceptable delay.

Algorithm (Δ *Aggregation*): Let us denote the bucket size of link i by Δ_i and let τ denote the time to wake up an interface. Figure 4a illustrates the bucketization process performed by the upstream interface for link i. As illustrated, packets arriving during the current bucket are buffered and then transmitted in the next bucket.

- Each bucket has one packet right at the start of the bucket, and this packet contains the load information of the bucket (i.e., number of packets) in a special field in the header — this is the only way for the upstream interface to communicate with the downstream interface regarding loading for the current bucket.
- The *pattern* of packets within a bucket is known to the downstream switch. For example, packets can be back to back or spread out within the bucket. This pattern is also communicated to the downstream interface in the header of the first packet within the bucket.

The buffer space required at the upstream interface is Δr bits when the transmission speed is r b/s and the maximum delay experienced by a packet is Δ. Note that this delay may affect the performance of protocols like TCP if it is significant as compared to the round-trip time of a given connection. However, in our study this value was less than 0.5 ms; thus, we believe that the impact on TCP will be minimal.

Let us now consider the algorithm for determining K in the $N \times K$ merge. At any time t, assuming we start t from zero and count up, the downstream switch will know the load along link i until time

$$\left\lfloor \frac{t}{\Delta_i} \right\rfloor \Delta_i + \Delta_i .$$

Given N incoming links, the time until which the downstream nodes know the *total load* is

$$t + \delta_t = \min_{i=1}^{N} \left\lfloor \frac{t}{\Delta_i} \right\rfloor \Delta_i + \Delta_i .$$

As previously, let us denote the total load as a function of time, $L(t)$. We can now state a general *lossless* algorithm for turning interfaces on and off as follows.

Algorithm $K(t)$: At any time t determine load L for interval $[t, t + \delta_t]$. Let $L_{\max}(t_1, t_2)$ be defined as the maximum load in the interval $[t_1, t_2]$. In other words,

$$L_{\max}(t_1, t_2) = \max_{t'=t_2}^{t_2} \{L(t')\}.$$

Let $L^*(t)$ denote the number of interfaces at time t that are either awake or in the process of waking up. We compute L^* as

$$L^*(t - \tau) = L_{\max}(t - \tau, t).$$

The reasoning behind this expression is illustrated in Fig. 4b. First, note that L^* is only defined till time $t + \delta_t - \tau$ because it takes time τ to wake up an interface, and at time $t + \delta_t - \tau$ we only know the load until time $t + \delta_t$. At the time indicated by point a more interfaces need to be awake due to an increase in load. However, these interfaces need to have been woken up τ time previously at point b.

Once we have determined L^*, the algorithm itself is relatively straightforward and can be stated as follows. At any time t:

- Note that like L, L^* is a step function. At each inflexion point of L*, say at some time t', compute $I = L^*(t' + \in) - L^*(t')$. If $I > 0$, wake up I interfaces; else, put $(-I)$ interfaces to sleep. $\in$ is some very small number and is simply used to indicate a point in time right after the inflexion point of $L^*(t')$.

The algorithm is run at every inflexion point of $L^*(t)$. Furthermore, as new information about future load becomes available, L^* is updated.

Performance Study

In order to characterize the energy savings possible when using a merge network, we use simulated traffic traces as well as actual traffic traces collected at various interfaces of a LAN switch (gigabit interfaces). The traces were fed into a simulator built to simulate the merge network.

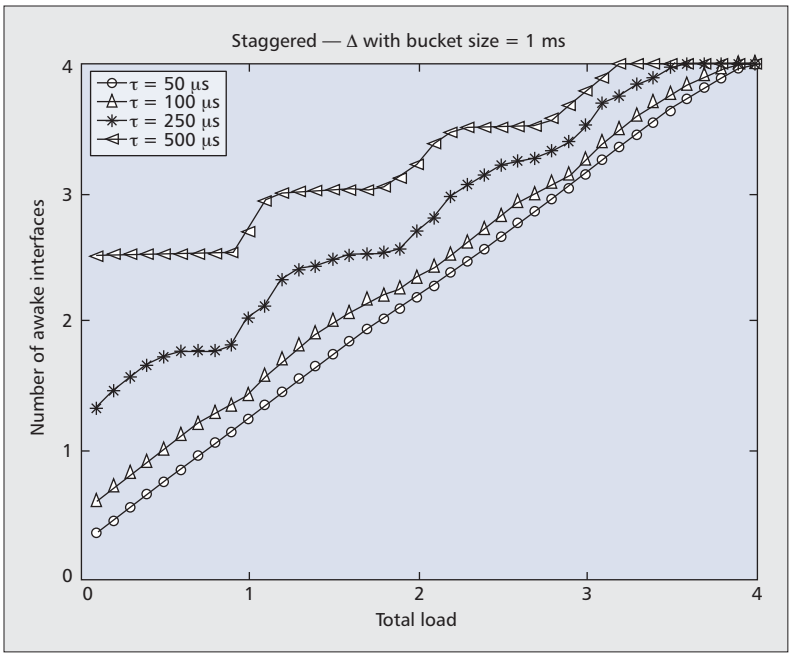

Figure 5. *Performance of a 4 × 4 merge network for varying loads.*

Table 1 summarizes the traffic traces from one of our LAN switches. As is evident, many interfaces on the switch are underutilized. We ran our aggregation algorithm on this data using an 8 × 8 merge network and we monitored the number of interfaces used over time. The bucket size used is 0.5 ms and $\tau = 100$ μs. The main result we obtain is that the maximum number of interfaces used (averaged over time) is 1.2. In other words, by using a merge network, we can power off an average of at least 6.8 interfaces of the switch out of 8 resulting in an energy savings of almost 85 percent.

Since the loading of our own network is so low, we used simulated traffic to examine the behavior of our algorithm over all loads. We assume, 4 incoming links and the aggregate load thus varies from 0 to 4 packets/packet time. Figure 5 plots the energy savings as a function of load over all loads. We ran the simulation for a bucket size of 1 ms and four different values of τ to study the impact of wakeup time on energy savings. As expected, as wakeup time decreases, the amount of sleeping is greater. However, even when the wakeup time is 0.5 ms we still see energy savings for loads up to 3/4 or 75 percent.

Caching Study — We used the LAN traces to understand the impact of merging traffic on cache locality. For our experiments we used a cache size of 2000 entries per interface. We noticed that several traces, such as interfaces P1, P3, P6, and P7, showed no cache misses at all even when using a cache size less than 500 entries. So we concentrated on interfaces P2, P4, P5, and P8, which did show some non-locality in addresses. The results are as follows. When we do not use a merge network at all, the misses are 0.614 percent because the cache needs to be populated. The 4 × 1 merge shows a higher miss rate of 0.77 percent, the 4 × 2 merge has a miss rate of 0.67 percent, and the 4 × 3 merge has a miss rate of 0.89 percent. These miss rates are not significant and demonstrate the feasibility of using the merge in real networks.

Conclusions

This article presents the idea of a *merge network* that enables us to merge traffic from N incoming links prior to feeding them to $K \le N$ switch interfaces. The number of active switch interfaces K can be varied depending on traffic load. This method is shown to linearly scale the number of awake interfaces with load. An important open issue is the changes required to layer 2 protocols to allow using the merge network concept and the effect of merging on TCP due to increased latency. We believe that merge networks can be incorporated directly into current switch designs; and, given the simplicity of the merge network, the added cost will be minimal.

References

[1] J. Chabarek *et al.*, "Power Awareness in Network Design and Routing," *Proc. IEEE INFOCOM '08*, Phoenix, AZ, Apr. 2008.
[2] G. Shen and R. S. Tucker, "Energy-Minimized Design for IP over WDM Networks," *J. Opt. Commun. and Net.*, vol. 1, no. 1, June 2009, pp. 176–86.
[3] L. Peh and V. Sorteiou, "Dynamic Power Management for Power Optimization of Interconnection Networks Using On/Off Links," *11th Symp. High Power Interconnects*, 2003.
[4] M. Gupta and S. Singh, "Energy Conservation with Low Power Modes in Ethernet LAN Environments," *Proc. IEEE INFOCOM (Minisymposium) '07*, Anchorage, AK, May 6-12 2007.
[5] C. Gunaratne *et al.*, "Reducing the Energy Consumption of Ethernet with Adaptive Link Rate (ALR)," *IEEE Trans. Computers*, vol. 57, no. 4, Apr. 2008, pp. 448-61.
[6] S. Nedevschi *et al.*, "Reducing Network Energy Consumption via Sleeping and Rate-Adaptation," *Proc. USENIX NSDI '08*, San Francisco, CA, Apr. 16-18 2008.
[7] G. Ananthanarayanan and R. Katz, "Greening the Switch," *Proc. USENIX HotPower '08*, San Diego, CA, Dec. 7, 2008.
[8] C. Yiu and S. Singh, "Merging Traffic to Save Energy in Enterprise Networks," Portland State Univ., Dept. of Comp. Sci., tech. rep., Oct. 2010.

Biographies

Suresh Singh [M] (singh@cs.pdx.edu) received his Ph.D. in computer science from the University of Massachusetts at Amherst in 1990 and his B.Tech., also in computer science, from the Indian Institute of Technology in 1984. He is currently a professor in the Department of Computer Science at Portland State University. His research interests include green technologies for Internet devices, energy-efficient computation, wireless networking protocols, information theory, and performance analysis.

Candy Yiu [S] (candy@cs.pdx.edu) is a Ph.D. candidate in the Computer Science Department at Portland State University. She received her M.S. degree in computer science from Portland State University in 2009, an M.Phil. from Hong Kong Baptist University in 2003, and her B.S. from the same university in 2001 in computer science. Her research interests are in wireless communications, protocol design and analysis, low-energy Internet research, and wireless multimedia. She is a student member of the ACM.