This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE ICC 2011 proceedings

# Energy-Conserving Switch Architecture for LANs

Candy Yiu
Department of Computer Science
Portland State University
Portland, OR 97207
http://www.cs.pdx.edu/∼candy

Suresh Singh
Department of Computer Science
Portland State University
Portland, OR 97207
http://www.cs.pdx.edu/∼singh

***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 paper proposes a different approach to the problem which involves a new switch atchitecture combined with a simple sleep/awake algorithm. The main results we obtain are that energy consumption, measured as fraction of interfaces that are awake, scales *linearly* with load for all loads with no delay and very low packet loss. We demonstrate the behavior of the algorithm via detailed simulations using synthetic traffic and traffic traces collected at a LAN switch in our university. We obtain over 90% energy saving in all real traffic traces and zero packet loss.**

## I. Introduction

Concerns over energy consumption in the Internet has prompted significant research activity in the past few years. The reasons include socio-economic ones such as increasing the penetration of the Internet into under-developed parts of the world and environmental reasons including reduction of the overall green house gas emissions. Technological reasons for reducing energy consumption include the increasing cost of cooling and the limits placed on increasing the speeds of routers and switches due to heat dissipation in the hardware, [4], [8], [12], [3]. Our focus in this paper is on reducing the energy consumption of enterprise networks which form a very large fraction of the deployed Internet (consider the fact that most workers use computers and most workplaces have networked devices like printers etc – all of these billions of nodes are connected through enterprise networks to the Internet). A key architectural component of enterprise networks are the high fan-out switches which connect end users to the network. For instance, a very commonly used switch in our network is a 48-port gigabit ethernet switch. Traffic traces collected by us as well as others [10] show that most switches see peak utilization of less than 10%. Clearly, with such low utilization, it is evident that these switches are highly energy inefficient. In this paper we present a new switch architecture which is very energy efficient and scales energy consumption linearly with load.

Previous research work on reducing energy consumption of switches have focused on using rate adaptation on individual links or putting interfaces on links to sleep. However, due to the bursty nature of the traffic, the results at the link level demonstrate energy savings only for loads less than 30% with packet loss and increased delay. Furthermore, the delay may cause TCP to misbehave and users will see significantly degraded throughput.

The current paper develops a solution for *linear scaling* of energy usage with all loads *without any delay*. The key idea is to merge traffic from all links 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. $K$ varies dynamically and scales with load. The reason this method scales as compared to the previous approaches is that by merging traffic we improve the predictability of load as well as maximize utilization of the awake interfaces. We evaluate our solution using simulated traffic as well as traffic traces collected at a LAN switch in our university. All cases show linear scaling. In all real traffic traces, it shows over 90% energy saving with zero packet loss.

The next section provides a literature survey that motivates our solution in section III. The architecture of the merge network is described in section III-A and algorithms for merging traffic are then discussed in section IV. Results from simulations using synthetic traffic as well as traffic traces is presented in section V. We summarize the main implications of the work for future switch design and discuss important research directions in section VI.

## II. Related Work

Approaches for saving energy at the level of switches has focused on exploiting periods of link inactivity to either put interfaces to sleep [6], [7] or use loading models to change the data rate on the interface [1], [5], [11], [2]. The idea behind these algorithms is simple – the upstream interface on a link maintains a window of inter-packet arrival times. This information is then used to determine the length of time that an interface can be put to sleep such that, with a high probability, the buffers at that interface will not overflow. The constraint is the non-zero time it takes for the downstream interface to wake up. There are several variants of this basic idea that have been studied. The results are that, for loads up to 30% of link capacity, sleeping is possible. However, the energy savings fall rapidly and reach zero beyond these loads.

978-1-61284-233-2/11/$26.00 ©2011 IEEE

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.

In parallel with the work on sleeping, another research direction has been to save energy by varying the link speed based on load. Thus, a 1Gbps link can be run at 100Mbps or even 10Mbps with significant energy savings. According to [11], running a 10Gbps interface costs 10-20W while running it slower at 100Mbps-1Gbps costs only 2-4W, representing a 5x savings. Early rate adaptation algorithms were developed in [5] for edge links. The essential idea is to use buffer occupancy thresholds to switch link speeds. There is a upper and lower threshold to signal when to switch up or down in rate. The other significant contribution of this work is the specification of link management policies to make link negotiation protocols run faster. Interestingly [5] also shows that power consumption with rate adaptation works well for loads less than 25-30%.

[11] takes a network-level view of sleeping and rate adaptation. For sleeping, they propose buffering packets at the entrance to the network in order to create bursts. This allows an interface to remain open for long times and sleep for long times. Similarly, for rate adaptation, the upstream interfaces at the edge of the network estimate load (again by buffering packets to create bursts) and then use this load information to switch link rates. The paper examines the impact of interface transition times (either sleep to wake or to switch between rates) and burst sizes on utilization and added delay. The results show increased delay of about 5ms for low losses while allowing sleeping or rate adaptation to scale with load of upto 30 – 40%. For higher loads, they begin to see more losses since buffers overflow at the input switches and load prediction algorithms begin to show large error.

## III. Re-Architecturing the Switch

A fundamental problem we face in putting interfaces to sleep or changing their rates is the unpredictability of traffic on that interface. Indeed, as we noted, these mechanisms show no energy savings even for loads as low as 30% because of the unpredictability of traffic arrivals. Our approach to this problem is to use the merge network and a simple sleep/awake algorithm in order to better use the resources.

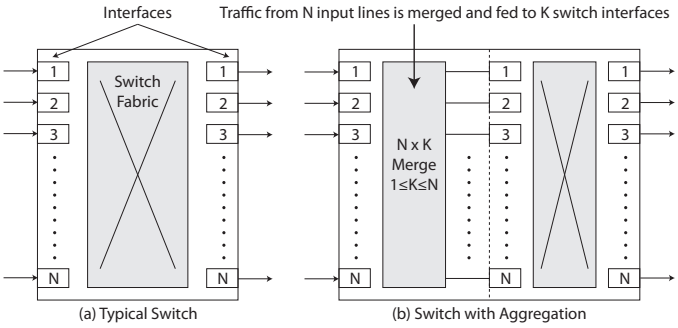


Fig. 1. Modified switch architecture.

Figure 1(a) shows a typical switch architecture while Figure 1(b) 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 Figure 1(b), we place a $N \times K$ 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.

### A. Architecture of the Merge Network

The function of a merge is to send $N$ input links to some $K \leq N$ 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 a 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 $K$ then 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* of 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, then 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 $4 \times 1$, $4 \times 2$, $4 \times 3$, and $4 \times 4$ merges.

In order to enable this form of non-deterministic routing in the $N \times K$ mesh, we require a special hardware element that we call the *selector* whose functional behavior is shown in Figure 2(a). 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 Figure 2(a). However, if two packets arrive at the two inputs, the earlier one is sent along the solid outgoing line while the

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.

later one is sent out along the dotted line. If the links have slotted transmissions and packets arrive simultaneously, then we assume some static ordering (say based on incoming line numbers) to determine which packet gets sent out over which outgoing line.

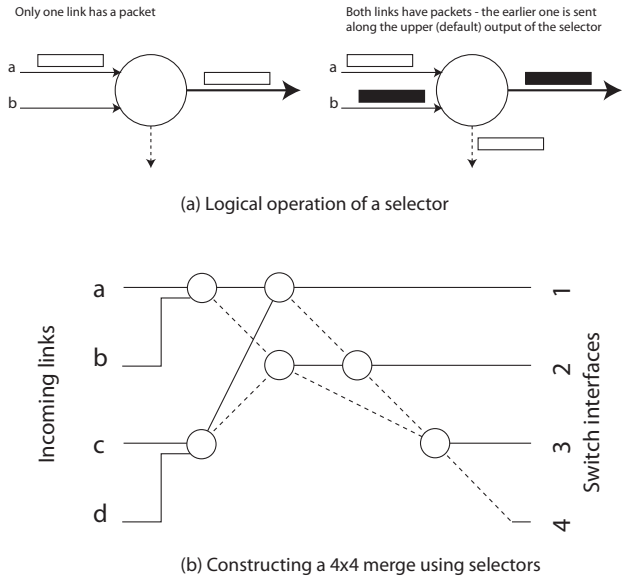


Fig. 2. Example of a merge network.

To illustrate how the selector can help us build a merge network, consider the $4 \times K$ example from previously. As before, label the interfaces 1, 2, 3, 4 and the corresponding incoming lines are labeled *a, b, c, d*. Figure 2(b) shows a merge network that supports all possible merge combinations. The solid lines emanating from each selector is the default output while the dotted line is the deflection route. To configure the network to perform a $4\times1$ 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 $4\times3$ merge, we simply shut off interface 4 and 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.

The general structure of a $N\times K$ merge network is a simple generalization of the one shown in Figure 2(b). A $\log N$ depth binary tree made up of selectors gives us a $N\times1$ merge. Next, take all the $N-1$ deflected outputs (dotted line in Figure 2(a)) of all the selectors and form a binary tree with those to get a $(N-1)\times1$ 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 specific by two numbers – the *depth* of the network and the total *number* of selectors used. For the network shown in Figure 2(b), the depth is 4 while the number of selectors used is 6. Indeed, we can prove that the minimum depth of a $N \times N$ merge network is $\log_2 N + N - 2$ and the number of selectors needed is $N(N-1)/2$.

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 [13]. We note the energy draw of a single selector is 1.65 $\mu$W only. *For $48 \times 48$ merge we need 1128 selectors for a total cost of 0.0042W. This is very small compared to typical 0.9W energy draw of a gigabit interface [10].*

## IV. MergeSleep Algorithm

Given the merge network described above, it now becomes possible to creatively merge packets from multiple input lines for processing by few input interfaces. However, we may pay a penalty in terms of potential packet loss if $K$ is smaller than the number of overlapping packets that arrive. Indeed, the system may be approximated as a $K$ server loss system where the arrival process consists of $N$ merged streams and the service time is proportional to packet length (no buffers). If the merged arrival process is Poisson and service time is also Poisson, we get a loss probability equal to Erlang's B formula since packets arriving when the servers are busy are dropped [9].

In this section we begin by defining "optimality" shows of merging and then describe our MergeSleep algorithm that turns interfaces on/off depending on load. This algorithm almost linear scaling of energy with cumulative load at the cost of small packet lost.

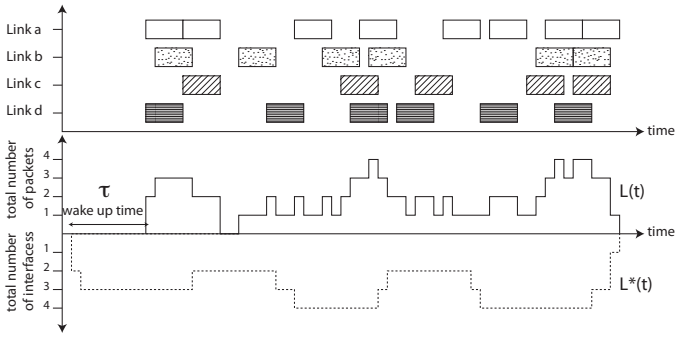


Fig. 3. Optimal aggregation of packets.

A key parameter that affects the amount of sleeping possible is $\tau$, the time to wake up a sleeping interface. If $\tau$ 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 $\tau$ time before they need to process a packet. The energy cost therefore includes the $\tau$ wakeup time. Consider the example in Figure 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 non-trivial value of $\tau$ as indicated in the figure, we can plot the number of interfaces that need to be awake or being 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 $\tau$ 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 and 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 $L^*(t)$.

*MergeSleep*

Assume we have a $N \times N$ merge network. Our load prediction algorithm works as follow:

- a silding window load estimator calculates the load ratio $r$ every $t$ seconds.
- initially, there are $N \times r \times \alpha$ open interfaces. $\alpha$ is a tunable parameter. When $\alpha = 1$, the number of opened ports are exactly equal to load, if the number of packets overlap more than the number of opened ports, there will be packet loss. If $\alpha = 2$, we double the amount of open ports to reduce packet loss due to bursty arrivals.
- the basic idea is to wake up interface $i+1$ when interface $i$ is busy and put interface $i+1$ to sleep when interface $i$ is idle again.

Figure 4 shows the state diagram for interface $i$. There are five states:

- IDLE: interface is awake but idle. It will go to BUSY state when it begins receiving packets. It will go to SLEEP state when the interface $i-1$ becomes idle.
- BUSY: interface is awake processing packet. It will go back to IDLE state when it finishes processing the last packet. It will go to ABOUT TO SLEEP state if interface $i-1$ done processing packet.
- SLEEP: interface is asleep, and if a packet comes when an interface is in SLEEP state, the packet is dropped. It will go to ABOUT TO WAKE state if interface $i-1$ becomes busy.
- ABOUT TO AWAKE: interface is scheduled to wake up but it takes $\tau$ to be fully awake. It will go to IDLE state when the time $\tau$ is up. Or it will go to SLEEP state interface $i-1$ becomes idle before time $\tau$ is up.
- ABOUT TO SLEEP: interface is scheduled to go to sleep but it is in the middle of processing a packet. It will go to SLEEP state when it finishes process the packet. It will go to BUSY state if starts of a new packet at interface $i-1$.

## V. 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 College of the Engineering Network at our University. The topology of the network is shown in Figure 5. Figure 6 plots the average utilization per

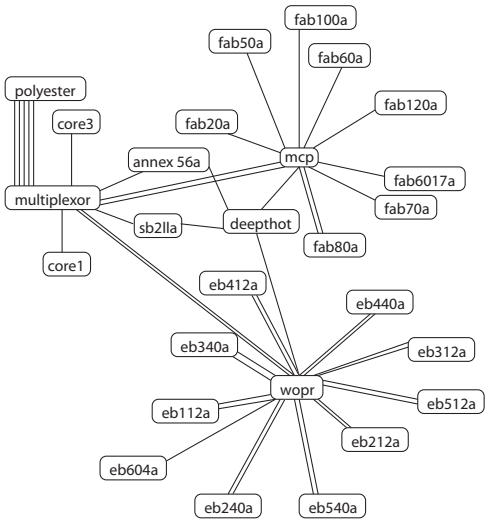


Fig. 5. Topology of Studied Network.

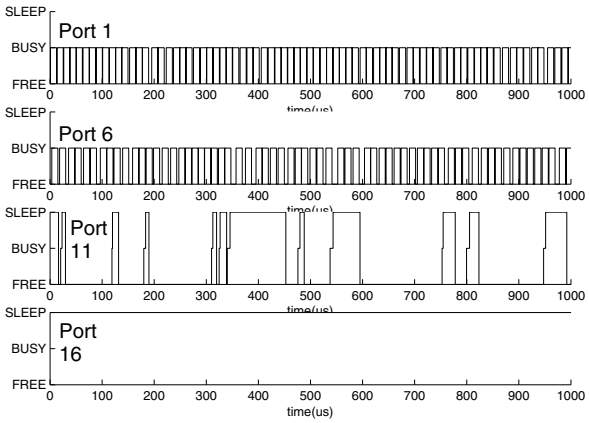


Fig. 6. States of interfaces 1,6,11,16

port for the top 20% ports in the network while Figure 7 plots the average switch utilization. Note that most ports show very low utilization of less than 5% and ost switches (with one exception) show utilization of less than 15%. This means that we can be very aggresive about merging traffic prior to feeding it into the switch.

We ran our algorithm on the traces for individual switches using a $48 \times 48$ merge. The main result we obtain is that most of the interfaces were very lightly loaded. The load is on average below 10%. By using a merge network, we can power off most interfaces and the energy savings is $97\%$ without any packet losses.

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 48 incoming links connected to a 48 port switch via a $48 \times 48$ merge network. The load varies from 0.0 to 0.9 over each link and follows a Poisson distribution.

Figure 6 plots the states(FREE,SLEEP,BUSY) of interfaces 1, 6, 11, and 16 for load is 0.2. In this trace, the packet size

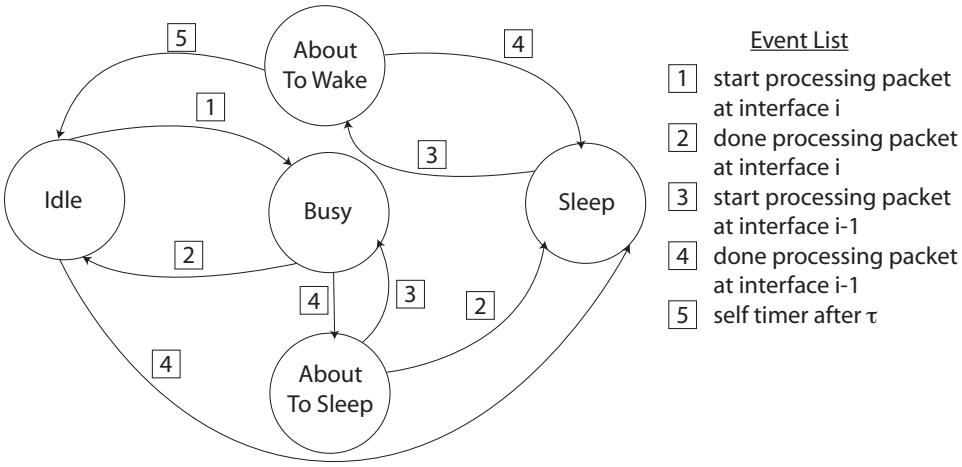


Fig. 4. Interface state diagram

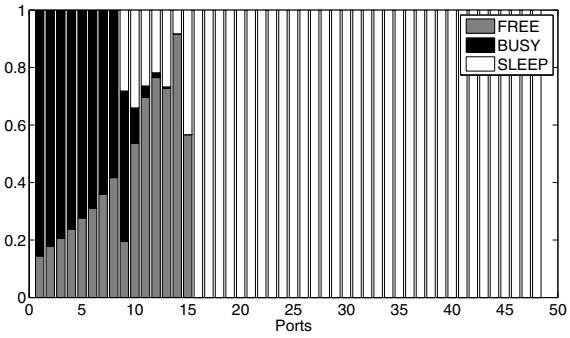


Fig. 7. Summary of States of the interface 1,6,11,16.

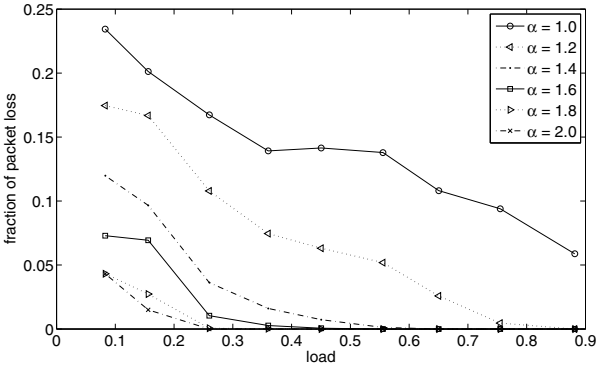


Fig. 9. packet losses vs load

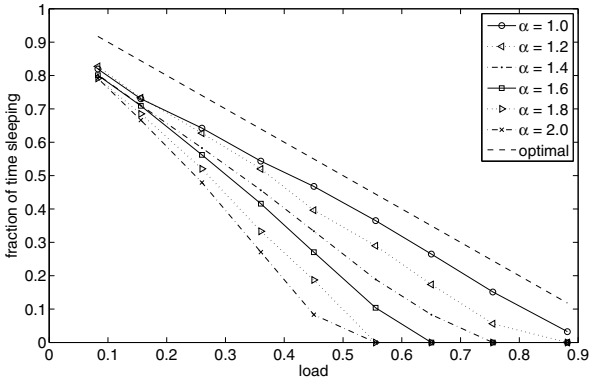


Fig. 8. sleeping vs load

of 1470MB is used and the switch consists of 48 ports. As we can see, interface 1 and 6 are mostly busy. Interface 11 jumps between busy and FREE state with slight sleeping. Interface 16 onwards are in SLEEP state. Figure 7 shows most interfaces are in asleep all time. A $\alpha = 1$ is used. Energy saving is more than 90%.

We conducted a study of the effect of varying $\alpha$ on load and loss behavior. Figure 8 shows the fraction of sleeping using different $\alpha$ factor with different load. The dotted line is the optimal saving. As we can see, with $\alpha = 1$, it is very close to optimal and scale linearly with load. Figure 9 shows the correponding packet loss. In $\alpha = 1$ case, the packet losses are above 20% which is not practicle. Note that load above 0.4 yields almost no saving. And load less than 0.2 results more than 60% saving when $\alpha = 2$ (less than 0.05 packet loss). The sweet spot would be using $alpha = 2$ for load less than 0.2 and $alpha = 1.6$ for load between 0.2 and 0.4. When load exceed 0.4, we should open all interfaces. Due to the unpredictability of traffic, we should update $\alpha$ based on the preivous load statistic.

Figure 10 shows the best value of alpha for all loads which maximize the energy saving for packet loss less than 1%. The solid line indiates the energy saving and the dotted line shows the corresponding $\alpha$ value used. The $\alpha$ value is smaller when load increses. The reason is that when load is high, most of the interfaces are open. In fact, when the case of load is 0. and $\alpha$ is 1.2, $load \times \alpha = 1.08$ which is greater than 1. It means that all interfaces are open without sleeping.

Another interesting observertion is that the merge network yields a specific distribution of packets over the ports. Figure

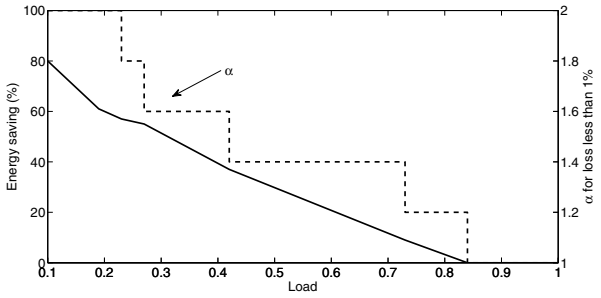


Fig. 10. Energy saving and $\alpha$ value for packet loss less than 1%

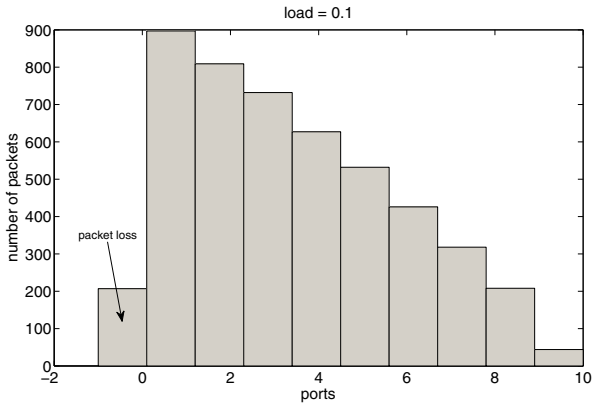


Fig. 11. Distribution of packets over all ports when load is 0.1

11 shows the distribtion of packets over all ports when load = 0.1. The port labelled -1 indicates packet loss. The shape of the distribution is a staircase. This creates the opportunity of energry saving at the tail of the stair case.

## VI. Conclusions

This paper presents the idea of a *merge network* that enables us to merge traffic from $N$ incoming links prior to feeding them to $K \leq N$ 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] M. Allman, K. Christensen, B. Nordman, and V. Paxson. Enabling an energy-efficient future internet through selectively connected end systems. In *Proceedings USENIX HotNets'07*, November 2007.

[2] G. Ananthanarayanan and R. Katz. Greening the switch. In *Proceedings USENIX HotPower'08*, San Diego, CA, December 7 2008.

[3] G. Appenzeller, I. Keslassy, and N. McKeown. Sizing router buffers. In *Proceedings ACM SIGCOMM'04*, Portland, OR, August 2004.

[4] J. Chabarek, J. Sommers, P. Barford, C. Estan, D. Tsiang, and S. Wright. Power awareness in network design and routing. In *Proceedings IEEE INFOCOM'08*, Phoenix, AZ, April 2008.

[5] C. Gunaratne, K. Christensen, S. Suen, and B. Nordman. Reducing the energy consumption of ethernet with adaptive link rate (alr). *IEEE Transactions on Computers*, 57(4):448–461, April 2008.

[6] M. Gupta, S. Grover, and S. Singh. A feasibility study for power management in lan switches. In *Proceedings IEEE ICNP'04*, Berlin, Germany, October 5-8 2004.

[7] M. Gupta and S. Singh. Energy conservation with low power modes in ethernet lan environments. In *Proceedings IEEE INFOCOM (Minisymposium)'07*, Anchorage, AK, May 6 – 12 2007.

[8] I. Keslassy, S. Chuang, K. Yu, D. Miller, M. Horowitz, O. Solgaard, and N. McKeown. Scaling internet routers using optics. In *Proceedings ACM SIGCOMM'03*, Karlsruhe, Germany, August 2003.

[9] L. Kleinrock. *Queueing Systems Volume I: Theory*. Wiley-Interscience, 1975.

[10] Priya Mahadevan, Sujata Banerjee, and Puneet Sharma. Energy proportionality of an enterprise network. In *First ACM SIGCOMM Workshop on Green Networking*, New Delhi, India, August 30 2010.

[11] S. Nedevschi, L. Popa, G. Iannaccone, S. Ratnasamy, and D. Wetherall. Reducing network energy consumption via sleeping and rate-adaptation. In *Proceedings USENIX NSDI'08*, San Francisco, CA, April 16-18 2008.

[12] A. Wassal and M. Hasan. Low-power system-level design of vlsi packet switching fabrics. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 20(6), June 2001.

[13] C. Yiu and S. Singh. Merging traffic to save energy in enterprise networks. Technical report, Portland State University, 2010.