Merging Traffic to Save Energy in the Enterprise

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

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

ABSTRACT

In typical enterprise networks, a large fraction of ports see utilization of less than 5% at peak times and close to zero utilization otherwise. Therefore, the normal architecture of one switch port per end-host is very wasteful because of the need for high port density switches to support numerous end users. In this paper we propose merging traffic from multiple end-hosts and feeding that to small port density switches that can replace the high port density switches. The energy savings from such a redesign are significant. The innovative part of this paper is the design of a low-power Merge network that is used to merge traffic from N incoming links to be fed to K switch ports and for sending traffic from the K-port switch to N links. Further, we present algorithms to enable network designers to re-architect their networks using the merge network, and a feasibility study using our College of Engineering network as a working example to illustrate how this approach would work and the resultant energy savings of almost 47%.

Categories and Subject Descriptors

C.2.6 [**Internetworking**]: Routers; B.4.1 [**Data Communication Devices**]: Interconnections

General Terms

Design, Experimentation

1. INTRODUCTION

Concerns about energy consumption in the Internet has prompted significant research activity recently. The focus of much of the work has been on reducing energy costs of data centers [15, 11, 12], incorporating energy as an optimization metric in network-level route optimization [19, 8, 17], and reducing energy consumption at the link level by rate adaptation and sleeping [10, 6, 9, 16, 7].

In this paper we examine the problem of energy efficiency as it relates to *enterprise networks*. These networks are highly under-utilized with most switch ports often seeing less than 5% load on average [10, 14]. The approach we investigate aggregates traffic from multiple links prior to feeding them to a switch. This *merge stage* or *merge network* is a separate entity that performs real-time traffic aggregation allowing us to replace present-day high port density switches with switches having far fewer ports. This automatically reduces energy use in both the chassis as well as the line cards while ensuring that end users see no degradation of service. Specific contributions in this paper include:

- Design of the Merge network and a prototype hardware realization of a small merge network that merges 4 links into 2.
- Introducing a Merge network at a switch has the unfortunate drawback of eliminating the 1-1 mapping between end systems and switch interfaces. This causes many link layer protocols to fail. We present the design of a *port virtualization* software module in Click which allows us to restore this 1-1 mapping.
- We study the question of selecting appropriate merge network/switch combinations in order to maximize energy savings while meeting a loss constraint.
- We show that using our approach in our College of Engineering network can save up to 47% energy.

The remainder of this paper is organized as follows. The next section provides a brief review of related work with particular focus on energy measurements of entire switches which is relevant to the work described here. In section 3 we provide details of our own college-wide network and the traffic loading summary seen on individual links. Following that, section 4 describes the design of the Merge network as well as design of the port virtualization in section 4.3. Section 5 presents the algorithm for traffic aggregation using the merge network. Later in section 6 we demonstrate the potential savings possible if traffic is aggregated using our approach.

2. RELATED WORK

Priya et al [14] provide a comprehensive measurement of traffic per port and energy consumption for switches in an enterprise network. While the specific topology of the network and the switch types are not provided, the results are nevertheless very enlightening. Their network has 90

switches (of four types) with chassis power varying between 55 – 150 W. The total number of ports across all switches is 6710 of which 716 were disabled. Their observations show that the average utilization among a majority of active ports was under 5% with a standard deviation of less than 20. Roughly 13% of the ports showed zero throughput. The average network power consumption for the 6-day observation period was 18,229 W with only a 5% variation with traffic. Our own measurements of loading patterns in our network support these results, as we show later in section 3. *The singular conclusion we can draw is that enterprise networks are highly over-provisioned and consume very large amounts of power while doing little work.* However, given the current switch design and Layer 2 protocols, there are few options (i.e., each end device needs its own dedicated port thus resulting in expensive high port-density switches) in how to reduce energy draw.

Sleeping and rate adaptation at switch interfaces have been studied as two ways to reduce energy consumption. However, savings using these techniques is limited due to the energy consumption breakdown within switches. For example, Chabarek et al [8] provide energy measurements of two Cisco routers – a GSR 12008 and a 7507. In the case of the GSR, their measurements show that the chassis alone consumes 200W of power. When we add a network processor and the switch fabric, the power draw increases to 500W. If we add two four-port Gigabit Ethernet cards, the cost goes to 650W. Broadly, the energy cost splits as: chassis 30%, fabric 46%, and interfaces 24%. Thus, even with perfectly optimal sleeping, we can at most power off the interfaces on the linecards yielding less than 24% savings. Note that we cannot power off linecards arbitrarily since it takes a few seconds for them to come online again resulting in either high latency or high packet loss. In order to allow linecards to be put to sleep, we developed a sleeping algorithm [18] in which traffic is moved from interfaces on sleeping linecards to interfaces on linecards that are powered on. That does allow us to get close to optimal sleeping performance (for example, 24% savings in the above example).

The approach studied in this paper replaces high port density switches with small port density switches that use significantly less power. Simultaneously, in order to support a 1-1 mapping between end systems and switch interfaces (necessary for layer 2 protocol compatibility), we introduce a Merge network and a port virtualization software within the switch to maintain the 1-1 mapping. As we will show, this yields dramatic savings in the enterprise and far higher savings than are possible with sleeping algorithms alone.

3. TRAFFIC CHARACTERISTICS OF OUR NETWORK

In order to motivate our approach as well as provide an environment for testing, we conducted a study of the College of Engineering network. The topology of the network is shown in Figure 1. All the edge switches are connected to workstations and most of the edge switches are actually stacked switches. In some cases, pairs of switches/routers are connected by more than one link. The overall summary of the number and type of switches is given in Table 1 (all the switches are Cisco switches). The table also summarizes the energy draw of each switch type at maximum throughput and at 5% throughput. As is evident, there is only a 11% difference between these two numbers in the grand total. The implication is that even though the 3750 series of switches are designed to be energy efficient, their energy consumption does not scale with load.

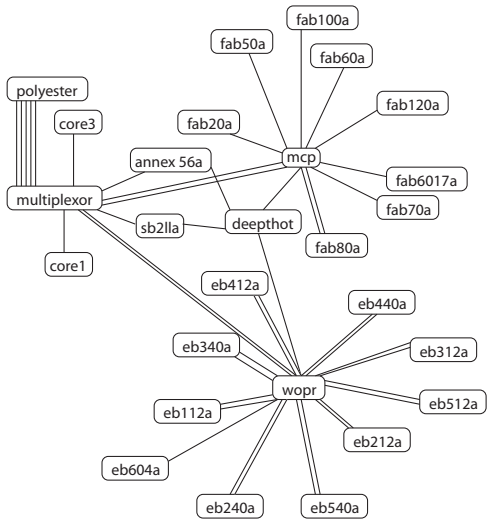

Figure 1: Topology of Studied Network.

Switch Model	Number of Switches	Power at Max Tput (W)	Power at 5% Tput (W)
3750 G-24TS	3	169	134
3750 48TS	33	66	60
3750 G-48TS	20	152	134
3750 24PS	1	57	55
3750 24TS	1	45	41
3750 G-12S	2	100	75
3750 G-24T	1	98	86
3550 12G	1	190	190
Grand Total (Max/5%) = 6,315/5,584W			

Table 1: Overview of switch type.

As part of a larger study on enterprise networks, we collected traffic traces over the course of a work day mid-week for all the interfaces at all the ports. The results of that study including a detailed analysis will be forthcoming but for our purposes in this paper, we are mainly interested in peak and average utilization statistics as a motivation for the merge concept. In Figure 2 we provide a CDF of the per-port utilization seen (ratio of total up and downlink traffic divided by bandwidth of connection) for the top 20% of the ports. As is evident, the ports tend to be very under utilized with more than 95% showing a utilization of well below 2%. The ports showing a greater than 8% utilization are connected to servers and to other switches. In Figure 1 each switch is made up of one or more stacked switches – we compute the peak utilization through all the switches per set of stacked switches and plot a CDF of that in Figure 2 as well. The peak utilization is computed by dividing traffic through each switch into 1s buckets and taking a maximum of the ratio of the total throughput during a bucket and the maximum possible throughput through the *active ports* on that switch. Not surprisingly, the peak utilization is 25% for one switch and well below 10% for all the rest.

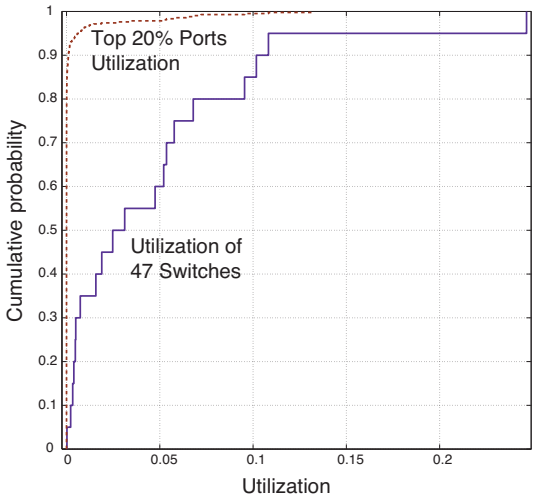

Figure 2: Utilization of the top 20% ports in the network as well as switches.

A conclusion of this study, which is also supported by [14], is that enterprise networks tend to be heavily over-provisioned. The design of the networks is guided by the need to provide connectivity to end users and given the manner in which switches are designed (i.e., one port per end host) there is little choice in how the topology evolves.

4. TRAFFIC AGGREGATION

Given the very low utilization of most ports on a switch, our approach calls for merging traffic from multiple links and feeding the merged stream to a switch with fewer ports. Figure 3 illustrates the idea behind traffic aggregation. As shown, traffic to/from N links are merged to K thus reducing the required port density of the switch from $N \times N$ to $K \times K$. As an example, if we merge 48 100Mbps links to 24 using a 48×24 merge network, we can replace the 48-port switch with a 24-port switch resulting in energy savings. Of course, there are several issues that need to be dealt with in order for this concept to work. A $N \times K$ merge needs to have the property that if at most K packets arrive on the N *uplinks* (i.e., from the N links into the switch) then there are no packet losses and they are all forwarded to the K port switch[1]. On the other hand, if more than K overlapping packets arrive then the *earliest* K get forwarded while the remaining are dropped. On the *downlink* (i.e., from the switch to the N links), the merge network needs to be able to forward packets from any of the K switch ports to any of the N downlinks and be able to forward up to K downlinks simultaneously.

An important requirement of the merge network is that it operate *entirely in the analog domain*. In other words, the merge network in no sense 'receives' packets. Rather, it operates much like a railroad switch where a train is switched from one line to another – the train is not received or buffered before being forwarded. It simply takes an alternative path. We do the same thing in our merge network where packets are dynamically switched to follow some path through the merge network. The reasons we chose to build the merge network in this manner are threefold: *this design ensures very small latency, the energy cost of the merge network is minimal and this design allows us to make the merge network relatively transparent to the PHY and MAC layer protocols.* In the remainder of this section we describe the uplink merge network, the downlink merge network and finally the port virtualization implementation needed to perform the $N : K$ mapping correctly to allow layer 2 protocols to function.

[1]Note that if we use hubs, packets arriving at overlapping times will all be destroyed which is one reason that we need to design a special merge stage network. Also, most hubs simply repeat a packet to all other links which is undesirable.

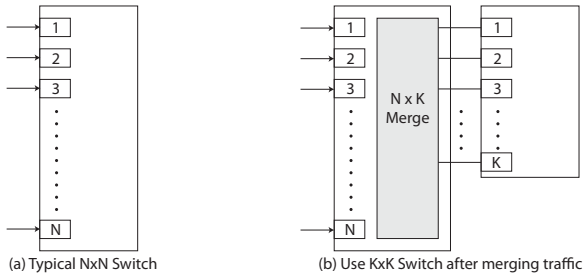

Figure 3: Reducing switch port density.

4.1 Uplink Merge Network

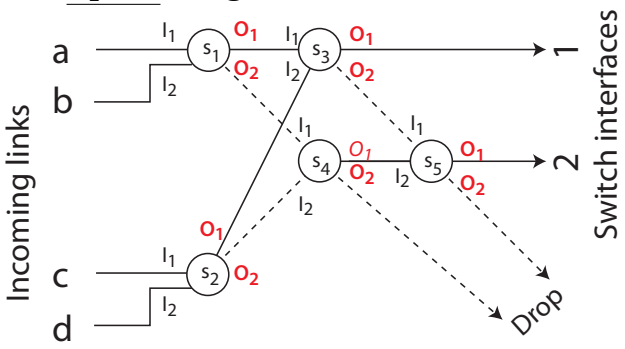

Figure 4: A 4×2 uplink merge network.

O_1 is the preferred output. The earliest packet is always sent to O_1 and if there is a second overlapping packet, that is sent to O_2.

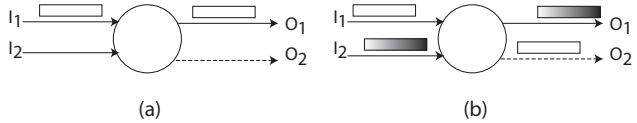

Figure 5: Operation of a selector.

The goal of the uplink merge network is to forward up to K packets from N incoming links to the K switch ports. An example of operation of a 4×2 merge network is provided in Figure 4. Here we have four incoming links labeled $a - d$ and two switch ports 1 and 2. If a packet arrives at interface d followed by a packet along interface a (overlapping) then the expected outcome is that d's packet is sent to switch port 1 and a's packet is sent to port 2. Any other packet arriving

at a time when switch ports 1 and 2 are busy should be discarded *without* colliding with the transmissions already in progress.

In order to build such a $N \times K$ merge, we need a special hardware element we call a *selector* whose functionality is illustrated in Figure 5. There are 2 incoming links and two outgoing links. If a packet arrives on only one of the two incoming links I_1 or I_2, it is always forwarded to the same outgoing link O_1. However, if packets arrive along both incoming links then *the earlier* arriving packet is sent out along O_1 while the later packet is sent out along O_2. Given a selector circuit, it is now a simple matter to construct a $N \times K$ merge that behaves as discussed above. Consider again Figure 4 that shows the implementation of a 4×2 merge using 5 selectors. The solid lines in the example indicate output O_1 of the selector while the dotted line is output O_2 of the selector. In this example, if more than two packets overlap then all but the first two are dropped. The earliest arriving packet is sent to output link interface 1 and the second earliest packet is sent to interface 2. The general design of a $N \times K$ merge is a simple generalization of this example. A $\log N$ depth binary tree made up of selectors gives us a $N \times 1$ merge. Next, we take all the $N-1$ deflected outputs O_2 (dotted lines from s_1, s_2, s_3 in Figure 4) of all the selectors and form a binary tree with those to get a $(N-1) \times 1$ merge. This process iterates K times to create a $N \times K$ 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 Figure 4, the depth is 3 while the number of selectors used is 5. We assert that this is the optimal solution in that no other design will have fewer levels or fewer selectors.

Theorem 1: The minimum depth of a $N \times K$ merge network is $\log_2 N + K - 1$ and the number of selectors needed is $\sum_{i=1}^{K}(N-i)$.

Proof. Consider a $N \times 1$ merge first. This can be seen as selecting the earliest arriving packet out of N possibilities and therefore we need exactly $N-1$ pairwise comparisons which corresponds to $N-1$ selectors. If we do a $N \times 2$ merge, then we need to select the two earliest arriving packets. This can be done by selecting the first packet and then selecting the second out of the remaining $N-1$. Therefore we need a total of $N-1+N-2$ pairwise comparisons. This generalizes in the obvious way and we get a total number of selectors of $\sum_{i=1}^{K}(N-i)$.

Selecting 1 out of N is done most efficiently when the $N-1$ comparisons are performed in a balanced binary tree giving us a depth of $\log_2 N$. To select the second out of the remaining $N-1$ we also do this in a binary tree *but the result of the last comparison of a $N \times 1$ merge is required here.* Therefore, if we get the output of the $N \times 1$ merge at depth $\log_2 N$ then the second output of a $N \times 2$ merge can only be got after performing one additional comparison giving us depth $\log_2 N + 1$. Therefore, for a $N \times K$ merge, we get a depth of $\log_2 N + K - 1$. □

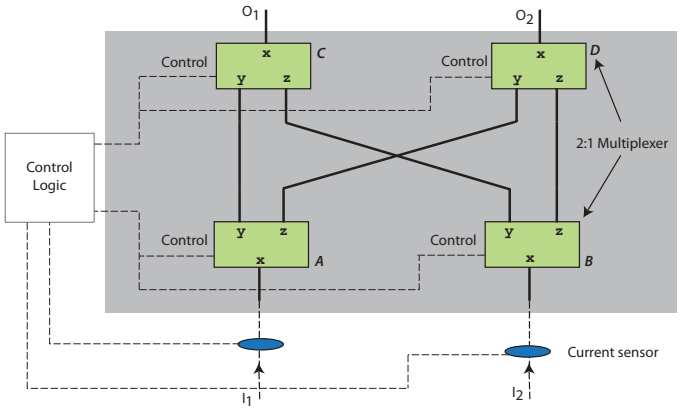

Figure 6: Design of a selector.

4.1.1 Hardware Prototype

We built a 4x2 uplink merge network using easily available hardware. The key part of the merge network is the selector and we illustrate its design in Figure 6 using a very low power analog multiplexers, the ADG904 [1]. In general, $M:1$ multiplexers switch one of the M inputs to a common output based on a control signal that is $\log_2 M$ bits wide. Figure 6 provides a schematic of the selector circuit – the selector itself consists of the shaded areas the other parts of the circuit are common to the entire merge network. As shown, we use four 2:1 multiplexers with associated control circuitry. The operation of the selector depends on appropriately configuring the four Muxes for individual packets. Consider the two cases illustrated in Figure 5 and how those get implemented in Figure 6. For the first case when there is a single packet arriving along I_1, the packet enters Mux A along input $x.A$. This Mux is configured to forward the signal out along output $A.y$ by the control logic whose input is the binary value produced by the current sensors [3]. Simultaneously, Mux B is configured to forward $x.B$ to output $B.z$ (even though there are no packets arriving into B in this example). Muxs C and D are reversed in operation (this is done simply by toggling a control bit). In this example, $y.C$ is configured to connect to the output $C.x$ and $z.D$ is configured to output to $D.x$. Consider the second example illustrated in Figure 5(b) where we have two packets coming into the selector at overlapping times. Since the packet begins arriving at I_2 first, the configuration of the four Muxes is set as follows: $x.B \rightarrow B.y$, $x.A \rightarrow A.z$, $z.C \rightarrow C.x$, and $y.D \rightarrow D.x$. *We need current sensors [3] because packet transmissions are asynchronous and there is no way for the merge network to know that a transmission is imminent. Therefore, our solution is to detect the signal on the wire using a current sensor whose output is fed to the control logic for appropriately configuring the four Muxes.* A challenge in building the merge network is the signal attenuation as well as the switching time of the Muxes. The ADG904 [1] model Muxes we used have very low insertion loss (1.1dB) but a 10ns switching time. This delay means

that 10 bits of the preamble of a gigabit rate 802.3 frame[2] will be lost or corrupted since the Mux is in transition. In this initial prototype, we delay the signal for 10ns by adding a loop of wire after the current sensors. This gives enough time for the Muxes to be set.

For the 4x2 merge network we use 5 selectors. Since each selector is made up of four Muxes, we have a total of 20 Muxes in our design. The energy draw of each Mux is 1.65μW for a total of 35μW for the Muxes. The additional logic requires almost 32μW of energy for a grand total of 67μW for the 4x2 merge network. We believe that this energy requirement can be reduced by an order of magnitude by building the entire circuit in CMOS rather than relying on use of discrete components as we did. However, even our relatively simple design shows that the merge network is very low energy thus supporting our main point that merging traffic will save energy.

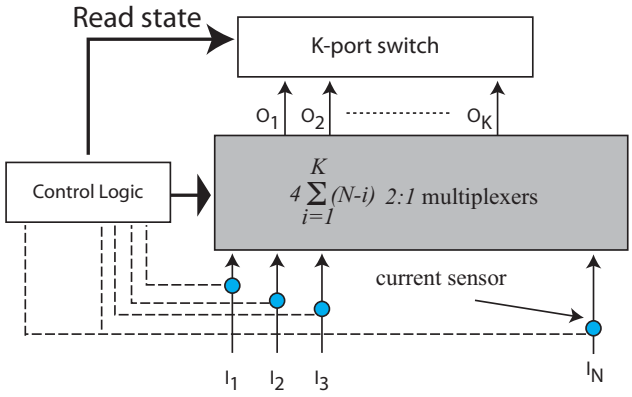

Figure 7: Design of a merge network for the uplink.

4.2 Downlink Merge Network

Consider now the problem of sending packets from a switch with K ports to N links. Unlike the uplink merge network, the implementation here is simpler since we do not have to select K out of N but rather send packets from one of K interfaces to one of the N links. A simple way of accomplishing this is illustrated in Figure 8 assuming we have $1:N$ multiplexers and $1:K$ OR gates (the shaded area is the part of the merge network while the control logic will be inside the switch). It is easy to see that using this circuit we can send K packets to K distinct links simultaneously and without conflict (from K different switch ports). However, in order for this part of the merge network to work correctly, we need to embed the control logic within the switch. This is required because packet headers need to be examined in order to determine which of the N links they must be sent out on and this function is performed within the switch. Also, since any switch interface can be configured to transmit packets to any of the N links, it is easy to cause out-of-order and even conflicting transmissions to a particular link (by transmitting packets for a destination from multiple switch interfaces). Therefore, we need to ensure that the internal logic of the switch and merge network combination is correct. The control logic for the downlink merge network needs $\log_2 N$ bits to control each of the $1:N$ Muxes shown in Figure 8. Since there are K interfaces, this translates to $K \log_2 N$ bits for controlling the merge. These address bits are calculated by the software running in the switch. Building the downlink merge using off the shelf components is constrained by the availability of Muxes in limited configurations of $1:8, 1:4$, and $1:2$. For instance, each $1:N$ Mux can be made using $N-1$, $1:2$ Muxes for a total of $K(N-1)$ Muxes. *In our 4x2 prototype, we use two 4:1 Muxes for a cost of 3.3μW.*

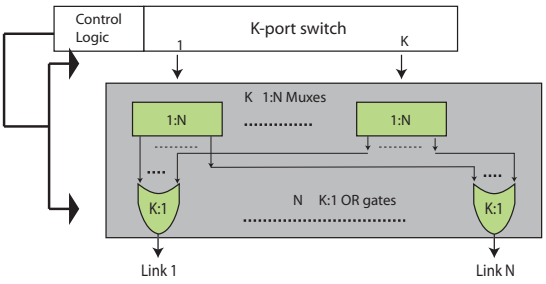

Figure 8: Design of a merge network for the downlink.

4.3 Port Virtualization

Modern day switches perform complex data forwarding tasks beyond simple unicast. For instance, switches today support QoS (Quality of Service) and VLANs (IEEE 802.1P and 802.1Q), Spanning Tree construction and maintenance (IEEE 802.1D, 802.1H), link aggregation (IEEE 802.1AX), port-based access control (IEEE 802.1X), MAC security (IEEE 802.1AE), congestion notification (IEEE 802.1Qau), resource discovery (IEEE 802.1AB), plus other standard and non-standard services[3]. Given this plethora of protocols supported by switches, we need to understand if these protocols can continue to be supported when we add in a merge network. To start with, it is clear that by breaking the 1-1 mapping between links and switch interfaces we are potentially causing any number of failure modes for these different MAC layer services. For instance, port-based security

[2]A 802.3 frame is preceded by a 7 byte preamble used for clock recovery as well as to indicate the start of a frame at the receiver.

[3]A very brief summary of these standards is as follows. QoS and VLANs are supported via 4 bytes in the Ethernet MAC frame that includes 3 bits for priority and 12 bits for a VLAN tag or identifier. Spanning tree construction and bridging require the exchange of BPDUs (Bridge Protocol Data Units) and running a spanning tree construction algorithm. Spanning trees support group communication for Dynamic VLANs (MAC based membership) as well as Static VLANs (port based membership). Link aggregation refers to combining multiple links between two devices to increase aggregate bandwidth. Port based access control is a security mechanism where a new device is allowed full network access only after being authenticated by a third party (also in the network). MAC security is a mechanism that provides security for connectionless data that allows the detection of unauthorized frames and dropping them at switches. It also has built in mechanisms to prevent replay attacks etc. Congestion notification is a mechanism that allows congestion information to be propagated to a source MAC as well as to higher protocol layers. Finally, resource discovery is a link layer mechanism for nodes to learn about the capabilities of others in the neighborhood via exchange of control packets.

(802.1X) relies on an explicit port-based connection of an end system to a switch. Similarly, spanning tree construction and VLAN construction also require an explicit use of the network topology. Congestion notification is meaningless if we no longer have the concept of a 1-1 link between two communicating end points. The interesting question we address is how to enable the switch to perform all of its functions even in the presence of a merge network.

The key solution we develop is that of building a software layer within the K-port switch called the *port virtualization* layer shown in Figure 9. The port virtualization layer is responsible for mapping packets coming on the uplink to one of the N virtual ports and on the downlink for scheduling packets for transmission over one of the K physical ports to appropriate downstream links. Let us consider the complexity of both of these operations in turn. On the uplink, it is essential that a packet which arrives along link n (where $1 \leq n \leq N$) be sent to virtual interface n regardless of which physical switch port k ($1 \leq k \leq K$) it actually arrived on. This mapping is needed to ensure that security protocols like 802.1X and other protocols for spanning tree and VLANs work unchanged. In our solution, the *uplink control logic* reads the Mux control bits of all the Muxes in the uplink merge network and infers the link that a particular packet arrives on. To explain this further, consider the 4×2 merge network from Figure 4 again. The five selectors in the figure are named s_1 – s_5. As shown in Figure 5, each selector has one preferred output (O_1). Therefore, we can represent the state of a selector using one bit as follows: if I_1 is sent to O_1 (by default, then, I_2 is sent to O_2) we denote this by 0 and if I_1 is sent to O_2 (and I_2 is sent to O_1) we denote this by 1. This state representation is unique and maps uniquely to the control bits of the four Muxes that make up the selector. Table 2 lists the state of the five selectors for a given link (a, b, c, d) to switch interface (1 or 2) mapping. Say a packet is arriving at interface 1 of the switch and interface 2 is idle. If the selectors s_1 and s_3 are in state 0 with the others IDLE (denoted by X in the figure), then we can infer that the transmission is coming in from link a because selector s_1 needs to be in state 0 so that a's packet is sent out along O_1 and s_3 also needs to be in state 0 for the same reason. The mapping of $a \rightarrow 2$ however can occur in two ways – $s_1 \rightarrow s_4 \rightarrow s_5$ or $s_1 \rightarrow s_3 \rightarrow s_5$. If two or more of the four uplinks are actively transmitting, we can determine which two make it to the switch's two interfaces by examining the setting of the five selectors in Table 2. For example, $c \rightarrow 1$ and $d \rightarrow 2$ will be inferred by the state, $s_1 = \text{X}, s_2 = 0, s_3 = 1, s_4 = 1, s_5 = 1$ (we are taking an OR of the two rows from the figure).

In Figure 9, the function of the various modules is as follows: The uplink port mapper reads the selector state of all selectors in the uplink merge network and determines which virtual port each packet in each of the K interfaces belongs to. This packet is then sent on to the appropriate virtual interface. The downlink control logic and downlink scheduler work hand in hand. When a packet is picked for transmission from some virtual port (say n) it is sent to a physical interface k. Just before interface k transmits the packet, the downlnk control logic sets the control bits of the associated downlink Muxes so that the packet is sent out along link n correctly.

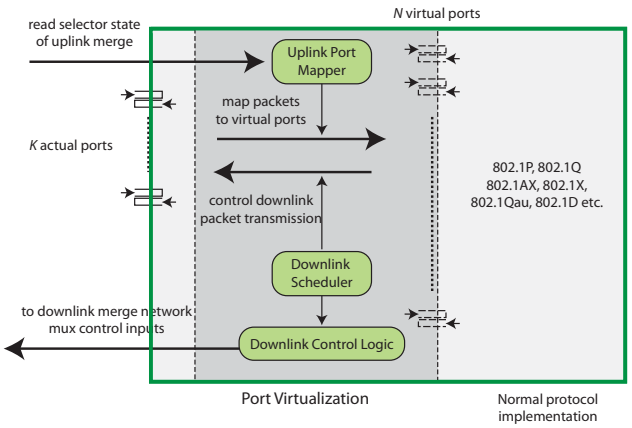

Figure 9: Software architecture of a switch with a merge network.

	s_1	s_2	s_3	s_4	s_5
$a \rightarrow 1$	0	X	0	X	X
$a \rightarrow 2$	0	X	1	X	0
	1	X	X	0	1
$b \rightarrow 1$	1	X	0	X	X
$b \rightarrow 2$	1	X	1	X	0
	0	X	X	0	1
$c \rightarrow 1$	X	0	1	X	X
$c \rightarrow 2$	X	1	X	1	1
	X	0	0	X	0
$d \rightarrow 1$	X	1	1	X	X
$d \rightarrow 2$	X	1	0	X	0
	X	0	X	1	1

Table 2: Determining the link $(a - d)$ that a packet arrives from into switch ports 1 or 2.

Given a port virtualization as described, most of the existing MAC layer services can be implemented without any constraint since the merge network is completely transparent and the N virtual ports are mapped 1-1 to N actual links. However, any data traffic that requires QoS guarantees may be affected at high loads (we have not studied this issue in detail yet). The implementation of virtual ports also helps with PHY layer mechanisms including auto-negotiation and fault tolerance. Typically, if no packet or electrical pulses are received on an Ethernet link for 50-150ms then the link is considered down. As a result, pulses are sent at some rate (normal or fast) to serve as a keep-alive signal in all networks. In addition, however, the pulses can be sent at a fast rate and used to encode binary information (called a link code word) that enables the two end-points to perform some handshaking. In our modified architecture, the protocol works as-is for the uplink direction due to port virtualization. On the downlink, however, the K physical interfaces need to broadcast these pulses to all of the downlinks. A second aspect of this signalling is rate negotiation on the link whereby the two end points agree on a transmission rate (10/100/1000). The IEEE is close to voting on the IEEE 802.3az [4] standard that allows pairs of nodes on a link to switch rates based on loading. This protocol can be run even with the merge network but, since a packet may be routed

to any of the K switch interfaces, we need to switch the rate of *all* N links simultaneously.

4.3.1 Implementation

We use a 2-port Click-based router [2] as the experimental platform for studying the Merge network implementation. The 4x2 merge network hardware is connected to the two gigabit interface cards on the PC running Click. We also use two 32-bit PCI Digital I/O cards for uplink and downlink control of the merge [5]. Within Click, we have built a 4-port port virtualization layer that essentially implements four pairs of queues in the kernel (one per interface). Packets from the uplink arriving at either of the two interfaces are appropriately queued at one of the four queues before the 802.3 protocol is called. For the downlink, packets are sent from one of the four queues to either of the two interfaces *after a short delay* to allow the Mux state to be set. In our current implementation we have only tested the basic 802.3 protocol (default implementation that is publicly available) for low load conditions. For low loads up to 20% per link, we do not see any packet loss in the merge network in the switch.

5. SELECTING THE RIGHT MERGE NETWORK

As our discussion above has shown, building a very low-power merge network is possible. Therefore, the natural question that comes up next is, how do we use merge networks in enterprise LANs? The question can be phrased specifically, for example, given a 144 port switch (or stacked switch), should we replace it with a 144×48 merge network and a 48-port switch or can we go to a 144×24 merge network and a 24 port switch? In answering this question we need to be mindful of the following constraints:

1. For current and projected traffic characteristics, packet loss in the merge network needs to be kept below some threshold. Thus, the maximum degree of merging that still meets the loss constraint would be a good choice.

2. Switches tend to be replaced with newer models after some number of years (commonly 5 years). Thus, selection of a merge network/switch combination needs to satisfy the loss constraint for this length of time after which a new configuration will be deployed.

3. We are constrained in our selection of the merge network by the limited availability of switches. Thus, most common switch configurations have 12, 24, or 48 ports. If the traffic can be merged to $K = 18$ ports, we will need to select a 24-port switch. Thus resulting in lower energy efficiency.

4. Some switches offer additional services such as power over Ethernet that are required at certain locations in the enterprise. Thus, we are further constrained in which merge network/switch combination will be feasible in those locations.

Packet losses in the merge network occur (on the uplink) when more than K overlapping packets arrive from the N links. To estimate the loss probability for a given K, we can use Erlang's B formula [13] if the merged traffic follows a Poisson distribution and if packet lengths are exponentially distributed. The reason this is possible is because the K switch ports can be viewed as servers in a queueing system. We thus get a K-server loss system where packets get dropped if all servers are busy. The loss probability then is,

$$p_{\text{loss}}(N \times K) = \frac{\rho^K / K!}{\sum_{l=0}^{K} \rho^l / l!} \quad (1)$$

where $\rho = \lambda/\mu$ with λ being the aggregate arrival rate from N links and μ is the mean service rate (packet length). Then the merge is feasible if $p_{\text{loss}}(N \times K) \leq \alpha, \forall i, 1 \leq i \leq m$. where α is the maximum acceptable loss. In Figure 10 we plot the minimum K required for a given aggregate load ρ to meet a loss constraint of $\alpha = 0.001$. For example, for a mean load λ of 10 we need $K = 15$ ports. If we assume that the maximum loading of a link is 5% (section 3) this means that we can merge 200 links to $K = 15$ ports while still meeting the loss constraint (assuming, of course, that the aggregate traffic model is Poisson).

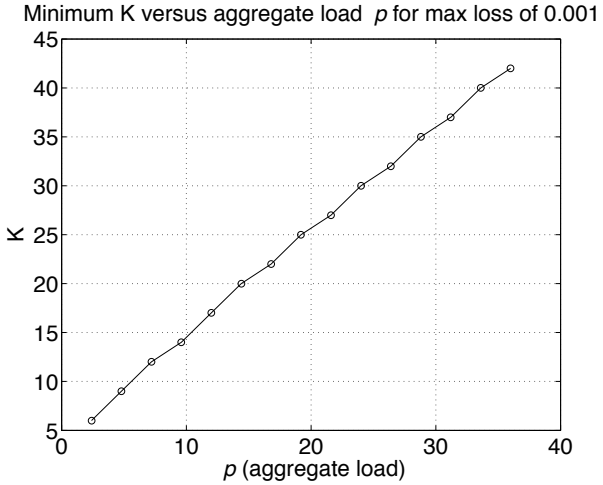

Figure 10: Minimum K as a function of total load for maximum loss probability of 0.001.

5.1 Unknown and Non-Identical Distributions

In reality, the traffic model from each of the incoming links is different, unknown, varies over the course of a day, and tends to be very bursty. Therefore, the discussion above cannot be directly used to design an appropriate merge network. However, a solution we implemented uses traffic shaping techniques at the upstream interfaces such that the traffic flowing along each link is approximately Poisson. We then design the merge network assuming some *maximum* loading of each of the links and apply the Erlang formula to find K.

Traffic shaping has been well-studied in the context of Diffserv networks, ATM networks and access networks where QoS guarantees are mandated by ISPs. Among the various hardware challenges in implementation of traffic shaping are increased buffering requirements and the overhead of maintaining state to create the appropriate output traffic model. However, in our case, since we are primarily considering links connecting end-hosts to switches in an enterprise network, the problem of buffering large amounts of data at

the network interface does not arise and traffic shaping is easy to implement. We implemented a simple algorithm in the Linux kernel where we modified the driver to initiate copy to the gigabit interface card only when a flag is set. The flag is set or reset based on a model of the output traffic pattern that is maintained in the kernel. We examined the resultant traffic arriving at the switch port for loads of up to 30% and see a very good fit to the Poisson distribution. A consequence of performing this form of traffic shaping is the increased latency. However, this value remained well below 10μs for all packets across all loads. One important note, however. Since load is very bursty, the output Poisson process has a time varying mean $\lambda(t)$ that is piecewise constant. Figure 11 provides one example of the traffic shaping applied to real-time traffic from one end-host. The plot at the top is the original trace that is obtained by capturing the times when write() is called. However, we inserted a shim layer where these writes are trapped so as to enforce traffic shaping. The second trace is based on the actual time when the NIC card transmits packets after the traffic shaping operation. The third traces plots the time varying $\lambda(t)$ for the output process. It is clear that the very bursty original trace is far better behaved after shaping.

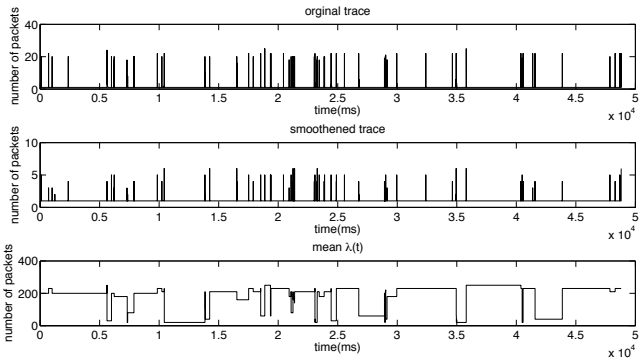

Figure 11: Traffic shaping for one link.

For downlink traffic, from the switch to a end-host, note that packet loss is low because the switch automatically buffers packets going out the interfaces if all the interfaces are busy. The only occasion we see packet losses is if there is a long sustained burst of traffic for more than K end hosts (for a $N \times K$ merge) such that internal memory overflows at the switch.

6. CASE STUDY: COLLEGE OF ENGINEERING NETWORK

In order to demonstrate the effectiveness of traffic aggregation and the application of our algorithms, we use the network from Figure 1 as our model. We consider how much energy will be consumed if we were to replace existing switches with smaller port-density switches using merge networks such that packet losses are kept below 0.001 per link. The approach we follow considers each of the 20 edge switches in Figure 1 individually. The energy consumption of the merge networks is derived by simply extrapolating the energy cost of selectors that we have currently fabricated and multiplying that with the number of selectors needed (Theorem 1) plus a 10% increment to account for the cost of control logic. We note that even for the largest merge network of size 144×48 the energy cost was below 0.1W.

As we see in Figure 2, the average utilization of the ports is below 2% for 95% of the cases. Therefore, for our study, we use a figure of 10% utilization per port as the target and design the merge network(s) on that basis. We assume that the traffic is Poisson (using traffic shaping) and then apply the Erlang formula to find the optimum K to save energy. Table 3 summarizes the results of this optimization.

The first column in the table lists the 20 different switches from Figure 1 while the second column indicates the original switch configuration (in some instances there are as many as five stacked switches). The energy numbers shown next are derived from the Cisco data sheets. For each of the 20 switches, we compute a merge such that the target loss rate of 0.001 is maintained. If we compare the final figures, we see a reduction of energy needs from 4264W to 2274W or a 47% reduction. We could have achieved even greater energy savings but are constrained by the limited availability of switches (i.e., limited port combinations).

7. CONCLUSIONS

The paper discusses the idea of merging traffic from end-hosts prior to feeding them to a small port density switch. Doing this allows us to obtain at least 47% energy savings in our College network. An important implication of this work is that enterprise networks can be made very lean by using the merge networks. Given the fact that enterprise networks form a large component of the Internet, this degree of energy savings has enormous global impact. As part of our ongoing work, on the hardware side we are continuing to build a more energy efficient merge networks in CMOS while on the software side we are modifying the kernel of Click to allow us to implement other Layer 2 protocols as discussed in the paper.

Acknowledgement

This work was funded by an internal Portland State University reserach development grant awarded in 2010 and partly by the NSF under award CNS-0722008.

8. REFERENCES

[1] Amd switches and multiplexers. http://www.analog.com/en/ switchesmultiplexers /products/ index.html, 2010.

[2] Click modular router. http://read.cs.ucla.edu/click/click, 2010.

[3] Ct current transformer. http://www.bergoz.com, August 2010.

[4] Ieee 802.3az. http://www.ieee802.org/3/az/index.html, July 2010.

[5] Pci digital i/o boards. http://www.generalstandards.com/digitalio3.php, August 2010.

[6] 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.

Name	Current Model(s)	Energy (W)	Merge (10% Max Load)	Model (10% Max Load)	Energy
eb240a,eb540a,eb512a eb440a, eb412a	G-48TS, 48TS	218W each	48×12, 48×24 each	G-12SD, 48TS each	117W each
eb312a, fab60a, fab100a, fab50a	G-48TS, 2 x 48TS	284W each	48×12, 96×48 each	G-12SD, 48TS each	138W each
eb212a	G-24TS, 2 x 48TS	301W	24×12, 96×48	G-12SD, 48TS	138W
eb112a	3 x 48TS	198W	114×48	48TS	66W
eb340a	2 x G-48TS, 48TS	370W	96×24, 48×24	G-24T, 48TS	143W
fab80a	G-48TS, 48TS, 24PS	275W	48×12, 48×24, none	G-12SD, 24TS, 24PS	174W
fab20a	2 x 48TS	132W	96×48	48TS	66W
fab120a	5 x 48TS	330W	96×48, 144×48	48TS, 48TS	132W
annex56a	G-48TS	152W	48×12	G-12SD	72W
eb604a	24TS	45W	unchanged		45W
fab70a, fab6017a	48TS	66W	unchanged		66W each
sb2lla	G-24TS	169W	unchanged		169W
Total (Original)		4264W	Total (with Merge)		2274W

Table 3: Applying merge to our network (a "G" in the model means a gigabit interface otherwise it is a 100Mbps interface).

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

[8] 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.

[9] 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.

[10] 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.

[11] László Gyarmati and Tuan Anh Trinh. How can architecture help to reduce energy consumption in data center networking? In *e-Energy '10: Proceedings of the 1st International Conference on Energy-Efficient Computing and Networking*, pages 183–186, New York, NY, USA, 2010. ACM.

[12] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, and N. McKeown. Elastictree: saving energy in data center networks. In *USENIX NSDI*, 2010.

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

[14] 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.

[15] X. Meng, V. Pappas, and L. Zhang. Improving the scalability of data center networks with traffic-aware virtual machine placement. In *IEEE INFOCOM*, 2010.

[16] 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.

[17] G. Shen and R. S. Tucker. Energy-minimized design for ip over wdm networks. *Journal of Optical Communications and Networking*, 1(1):176 – 186, June 2009.

[18] S. Singh and C. Yiu. Putting the cart before the horse: merging traffic for energy conservation. *IEEE Communications Magazine*, 2011 (to appear).

[19] N. Vasic and D. Kostic. Energy-aware traffic engineering. In *Technical Report, School of Computer and Communication Scieces, EPFL, Switzerland*, 2009.