Energy-Aware Traffic Engineering

Nedeljko Vasić and Dejan Kostić
School of Computer and Communication Sciences, EPFL, Switzerland
firstname.lastname@epfl.ch

ABSTRACT

Energy consumption of the Internet is already substantial and it is likely to increase as operators deploy faster equipment to handle popular bandwidth-intensive services, such as streaming and video-on-demand. Existing work on energy saving considers local adaptation relying primarily on hardware-based techniques, such as sleeping and rate adaptation. We argue that a complete solution requires a network-wide approach that works in conjunction with local measures. However, traditional traffic engineering objectives do not include energy. This paper presents Energy-Aware Traffic engineering (EATe), a technique that takes energy consumption into account while achieving the same traffic rates as the energy-oblivious approaches. EATe uses a scalable, online technique to spread the load among multiple paths so as to increase energy savings. Our extensive ns-2 simulations over realistic topologies show that EATe succeeds in moving 21% of the links to the sleep state, while keeping the same sending rates and being close to the optimal energy-aware solution. Further, we demonstrate that EATe successfully handles changes in traffic load and quickly restores a low overall energy state. Alternatively, EATe can move links to lower energy levels, resulting in energy savings of 8%. Finally, EATe can succeed in making 16% of active routers sleep.

Categories and Subject Descriptors

C.2.2 [**Computer-Communication Networks**]: Network Protocols; C.2.3 [**Computer-Communication Networks**]: Network Management

General Terms

Algorithms, Design, Management, Experimentation, Performance

Keywords

Energy-Awareness, Traffic Engineering, Online, Distributed

1. INTRODUCTION

Recently, the research community has recognized Internet's energy consumption as an important problem. The US network infrastructure requires between 5 and 24 TWh/year [20], which translates into a cost of $0.5-2.4 B/year. Although the networking equipment consumes only a fraction of the total energy used for IT, it is important to reduce the networking devices' energy consumption to cut energy costs in absolute terms. An additional important side effect of reduced energy consumption is reduced carbon dioxide emissions. Cheaper operating costs will also enable developing countries to deploy fast networking infrastructure, thereby bringing important content to more users.

The Internet's energy consumption is likely to increase as operators deploy faster, more power-hungry equipment to handle popular bandwidth-intensive services, such as streaming and video-on-demand. A large fraction of Internet traffic is generated by home users, which are currently limited by the asymmetric nature of ADSL and cable modems that have constrained uplinks. As the access links become symmetric with 50+ Mbps to home users (e.g., fiber-to-the-home), the traffic volume could dramatically increase. In addition, the cloud computing initiative proposes to locate the users' data and computation within the network. If this paradigm takes root, network traffic is bound to increase even further.

While reducing the energy consumption of network clients [2, 9], servers [13], and switches [7, 9] did receive considerable attention recently, there has been very little effort on reducing the energy consumption of the Internet backbone. If unattended, however, the energy consumption of the backbone and the routers leading to it could well become one of the Internet's dominant energy factors. Specifically, at an average access link speed of several tens of Mbps, the per-user power consumption of the core exceeds that of the access and metro links combined [22]. Since the power consumption of the router hardware greatly surpasses that of the optical equipment[22], we concentrate on saving the energy that is consumed by the routers and their line cards.

Network devices expend large amounts of energy even when they are idle or underutilized [3], and this problem is exacerbated by the need to overprovision the network to reduce jitter and packet loss. Complementary metal oxide semiconductor (CMOS) technology is reaching a plateau in power-efficiency [3], and the cooling costs of new equipment are also likely to increase. Finally, it is likely that energy costs will continue to rise, which will make the problem even worse.

Gupta *et al.* [10] suggest to save energy by putting network interfaces and other routers and switches to sleep, but do not present an actual network-wide algorithm. Existing work on energy saving considers local adaptation using primarily hardware-based techniques, such as sleeping and rate adaptation. Nedevschi *et al.* [20] propose two energy saving schemes. The first shapes traffic into small bursts at the edge routers to enable downstream line cards to sleep between two packet bursts. The second takes advantage of the fact that a device operating at a lower frequency and/or voltage can achieve a significant reduction in energy consumption. Their results argue for a few, uniformly distributed operating rates for

.

line cards. The latter approach is an important step toward energy-proportional networking hardware.

We argue that a complete solution requires a network-wide approach working in conjunction with local measures. The classic traffic engineering problem is already difficult as it has to counter short-term changes in traffic volume with quick decisions to shift traffic within the network in order to balance link utilization. Further, the algorithm should be stable even under frequent changes in traffic patterns that might lead to oscillations. Finally, adding energy-efficiency only makes the problem harder. For instance, an interface card or a router that is powered off might affect other devices in its neighborhood by, e.g., waking them up. Moreover, frequent changes in the operating rate of energy-proportional networking hardware can result in unnecessary packet delays and losses. In addition, although an optimal decision could potentially be derived using global information, such an approach is less likely to be deployed. Hence, we seek a scalable solution in which routers take independent decisions.

This paper presents Energy-Aware Traffic Engineering (EATe), a technique that takes energy consumption into account while achieving the same traffic rates between sour-ces and destinations as the energy-oblivious approaches. We assume a hardware model in which an interface can operate at various sending rates. EATe uses a scalable, stable, online technique to spread the load among multiple paths so as to increase energy savings.

While EATe can handle router hardware with a wide range of energy characteristics, we explore in detail two important points in the design space of future routers with hardware-based support for energy-saving. Our first algorithm assumes low idle power consumption and good ability to save energy across the link capacity. In this scheme, we shift links to lower energy regions while being careful not to move the corresponding links on the alternative paths to higher energy levels. We also explore an alternative model in which rate adaptation provides modest benefits and the idle consumption of the network element is relatively high. Here, we strive to remove traffic from as many links as we can to let them enter a sleep state. A related option is to make a concentrated effort to remove traffic altogether from routers and enable the entire chassis (in addition to line cards) to sleep.

In EATe, every source makes an independent, local decision based on the information it collects from its paths to the possible destinations. We therefore do not require excessive control traffic to achieve our goal. Because EATe is aware of different hardware operating rates and carefully controls the amount of traffic sent over the links, it dramatically reduces the number of energy-wasting changes between the rates that also have negative impact on performance.

Our extensive ns-2 simulations over realistic topologies show that EATe succeeds in moving 15-31% of the links (21% on average) to the sleep state, while keeping the same sending rates and being close to the optimal energy-aware solution. Further, we demonstrate that EATe successfully handles changes in traffic load and quickly restores a low overall energy state. We also show that EATe quickly moves traffic after link failure. On more energy-proportional hardware, EATe can move links to lower energy levels, resulting in average energy savings of 8%. Alternatively, EATe can succeed in making 10-24% (16% on average) of active routers sleep.

2. BACKGROUND

2.1 Hardware support for energy-saving

Networking hardware typically contains a chassis that consumes power whenever the device is turned on. One or more network elements (e.g., line cards) can be plugged in, and they each consume power even when they are not carrying traffic. A comprehensive characterization of power consumption by a variety of network devices is presented here [18]. In this section, we briefly describe energy saving features that are likely to be supported by future networking hardware [20]. As the Internet continues its exponential growth, the line rates will increase, and so will the networking elements' power consumption (for example, going from 1 Gbps to 10 Gbps resulted in a jump from 4 W to 20 W for Ethernet cards). These energy saving features can leverage readily available, proven technologies, such as sleep states, as well as frequency and voltage scaling that are widely in PCs. With a prototype already demonstrated [19], we believe that these features will be implemented in commercial networking equipment in the near future.

Sleeping. Modern processors typically include a number of states that enable various components to sleep, along with a penalty in the form of an increasing time needed to return from those states (e.g., C-states in Intel processors [1]). For the sake of simplicity, we assume the existence of one sleep state. As in [20], we assume that the time to enter the sleep state and return from it is as short as few tens of milliseconds, given the characteristics of some of the proposed hardware [11]. The work in [20] explores the benefits of a technique in which packets are batched together and delayed, in an effort to provide a "downstream" network element with more time to spend in a sleep state. It is however difficult to keep such an element in a sleep state because the packets cannot be indefinitely buffered. We believe that this points to a need for the network-wide solution.

A chassis with line cards that are all "sleeping" should also be able to enter a sleep mode. Doing so can bring about significant energy savings as the chassis and the necessary cards (excluding line cards) can consume one half of the router's maximum energy budget [3]. Removing traffic from all router's line cards requires a network-wide solution.

Rate adaptation. We also assume that frequency change and Dynamic Voltage Scaling (DVS) [25], some of techniques that have been successfully applied to general purpose processors (P-states in Intel processors [1]), could be implemented as well in the networking hardware. Of these two, DVS is particularly appealing given that reducing the voltage has a dramatic effect (quadratic decrease) on energy consumption. These techniques could be used to make the energy consumption of the network element proportional to its operating rate. The following equation defines the active power consumption as a function of its operating rate r: $p_a(r) = C + f_r(r)$. C captures the static amount of power that is consumed regardless of the operating rate, and $f_r()$ captures the way power grows as the operating rate r increases. Nedevschi *et al.* [20] explore the potential savings of hardware capable of supporting N performance states, each corresponding to a different link rate: $r1, r2, \ldots, rN$. Their results show that a uniform distribution of operating rates yields superior energy savings relative to an exponential distribution. These authors also develop practical techniques that can match the operating rate to the link utilization, at the expense of introducing a small, bounded delay.

Network-wide Impact. To summarize, we assume that the hardware is capable of automatically adjusting its operating rate to match its utilization, and that it can sleep whenever there is an opportunity. The techniques we described so far (sleeping and rate adaptation) can be characterized as *local*. We now turn our attention to the potential for energy savings made possible by taking network-wide information into account.

It is difficult to estimate what the values and properties of the

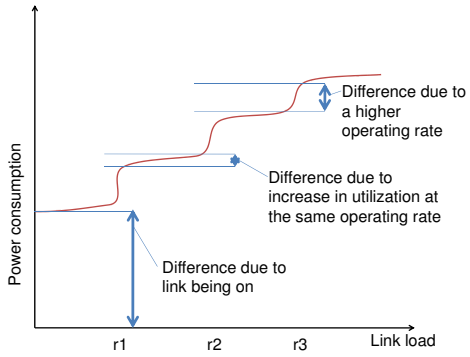


Figure 1: A case in which sleeping results in greater savings than adaptation, because the baseline consumption is relatively high and adaptation brings modest benefits.

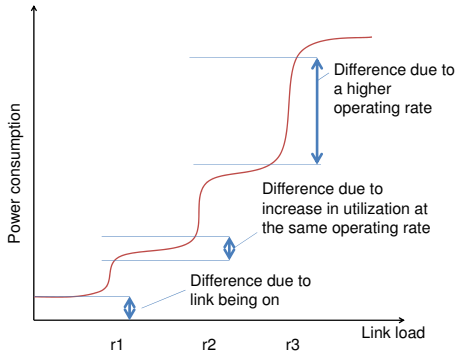


Figure 2: A case in which rate adaptation is preferable to sleeping, because large savings are possible with rate adaptation and baseline consumption is low.

various parameters (e.g., C) and functions (e.g.,$f_r()$), respectively, will be in the future hardware. We therefore concentrate on two characteristic cases located at the end points of the design spectrum, and depict them in Figures 1 and 2, which illustrate the predicted power consumption of a network element as a function of its load. These figures also show the three different operating rates that the element can use to match the offered load, in an effort to reduce the energy consumption.

High values of C, coupled with small or nonexistent savings due to frequency rate changes, would motivate the network operator to keep as few links active as possible (Figure 1). On the other hand, low values of C, along with the substantial savings that are possible from rate adaptation, would motivate the network operator to carefully load balance the network's link utilization (Figure 2).

The algorithms we developed are motivated by these findings, but they handle other cases in the spectrum of hardware characteristics (we discuss this further in Section 3.6).

2.2 Problem definition

The problem we wish to address can be described and parameterized as follows. There exists a set Z of source-destination pairs (i), which we refer to as sources and denote as z^i. The capacity of a link l is c_l, while its utilization is y_l. Multiple paths exist for each source-destination pair, and the matrix entry element H^i_{lj} is 1 if a source z^i uses the link l on a path j at its disposal (0, otherwise). Thus, z^i_j represents the amount of traffic sent over j-th path for z^i.

The problem at hand can now be stated as follows: given the sending rates at traffic sources our goal is to insure that all traffic reach its intended destination while using the minimal amount of energy for that task. Additionally, we cannot exceed any given link capacities. Equation 1 formally states the problem.

$$\begin{aligned} &minimize && \textstyle\sum_l e(y_l, c_l) \\ &subject\ to && \mathbf{y} < \mathbf{c}, y_l = \textstyle\sum_i \sum_j H^i_{lj} z^i_j, \forall l \\ & && \mathbf{z} = \mathbf{s}, z_i = \textstyle\sum_j z^i_j, \forall i \end{aligned} \quad (1)$$

3. APPROACH

In this section, we describe the way in which EATe finds a solution for Equation 1, while leveraging future hardware that can achieve considerable energy savings by adjusting its operating rate. For hardware that is likely to be deployed in the near future (Section 2.1), the energy consumption function $e(x)$ is not convex and it is not doubly-differentiable (e.g., Figure 2)[1]. Thus, we cannot easily solve this problem using traditional techniques. The problem can be formulated as mixed-integer programming, which is known to be NP-hard. Generally, mixed-integer problems are solved using heuristics such as branch-and-bound, cut generation, etc. However, none of these heuristics meets all challenges faced by a traffic management algorithm:

Efficiency. Any computationally- or memory-intensive task placed in a router's critical path would jeopardize the protocol's deployment. Thus, we seek an efficient solution, with small computational and memory requirements.

Responsiveness. Relying on a central authority to solve the problem is not possible given the computational power of existing hardware. For instance, in [3] the authors show that it takes a few hours to solve the problem by using an existing offline algorithm with full network information. We seek responsiveness, which can only be provided by an online algorithm.

Scalability. Although it is tempting to try to gather global network information, such an approach would not scale due to the need for a high update rate from each router (several times a second), and to the sheer messaging complexity.

Stability. Instability would lead to frequent changes in operating rates. This is undesirable for two reasons. First, it takes a significant amount of energy to switch between two operating rates. Second, a network element might not be able to serve the packets while it is switching rates, which could lead to packet loss and a further increase in latency.

EATe meets this requirements by relying on a fully distributed, online algorithm in which each intermediate router periodically reports its link utilization, while edge routers, based on this information, distribute traffic across alternative paths in a way that maximizes energy saving. Traffic distribution is done in a stable fashion to prevent unnecessary changes in rate operation.

EATe assumes the existence of multiple paths between any source-destination pair (alternatively called ingress and egress). These paths can either be precomputed off-line, or determined at runtime by the routing protocol. We do not require the paths to be disjoint. Since alternative paths are necessary for ISP's ability to tolerate link and router failures, they are likely to exist in their topologies. In the evaluation (Section 4), we show that the ISP topologies indeed have a sufficient degree of redundancy for our protocol's operation.

[1]"Convexifying" the non-convex objective in the problem formulation might remove the requirement of dealing with the mixed-integer programming problem. However, such an approach could miss opportunities for energy saving or push links to higher operating rates, as it might not be able to carefully observe the discrete power levels of the rate-adaptation capable hardware.

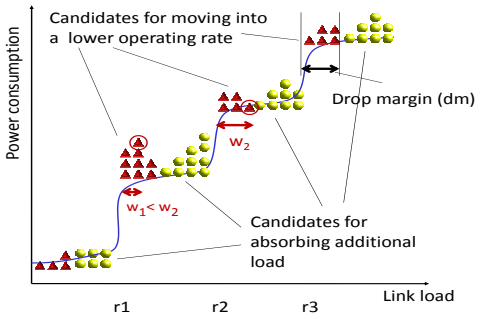


Figure 3: An example superimposing the link utilization distribution (before EATe runs) on the curve relating energy consumption and link load.

3.1 Leveraging rate adaptation

Next, we describe how EATe leverages the hardware-based energy improvements described in Figure 2. In this environment, it is expected that the ISPs would want to make full use of the hardware's rate adaptation characteristics. The baseline traffic management algorithm does however not take hardware properties into account, and it is thus likely that some links would lie close to a lower operating rate (within the *drop margin* dm, Figure 3). Our idea is to shift the load from these links ("triangles") to other links on alternative paths for the same source-destination pair ("circles"), while being careful not to move these links into a higher energy state.

The drop margin plays an important role in preventing simultaneous adjustments that could preclude energy savings. Suppose that we can shift traffic from any link, regardless of its distance from the lower operating rate. Consider two links that are fairly distant from the lower operating rate. We might end up moving both of them only half-way, without fully succeeding for either one of them. Alternatively, a link might end up absorbing additional traffic in a way that precludes its own transition to a lower operating rate. We use the drop margin to help us select the appropriate number of links that we will attempt to move so as to maximize energy savings. The details on how we determine the drop margin are presented in Section 3.5.

$$X^i = -\sum_{M(z_j^i)>0} \Delta z_j^i \quad (2)$$

$$B^i = |\{z_j^i | M(z_j^i) = 0\}|$$

$$\Delta z_j^i = \begin{cases} -M(z_j^i) * z_j^i & M(z_j^i) > 0; \\ X^i / B^i & M(z_j^i) = 0; \end{cases} \quad (3)$$

$$M(z_j^i) = \begin{cases} \min_l(w_l) & H_{lj}^i = 1 \wedge w_l < \mathbf{dm}; \\ 0 & \text{otherwise}; \end{cases}$$

$$w_l = (y_l - R(RI(y_l) - 1)))/y_l$$

$$\sum_j \Delta(z_j^i) = 0, \forall i \quad (4)$$

The EATe algorithm (Equations 2-4) that leverages rate starts by collecting information about links that are the best candidates for having their traffic removed. A router will mark the paths that feature links lying close to the lower operating rate by inserting their distance (w_l) from the closest lower operating rate, normalized by its current utilization. Figure 3 depicts such links with triangles. Since EATe tends to move links that are closest to the lower rate, an intermediate router will rewrite the distance only if it has a lower value to report, as depicted in Figure 3 ($w_1 < w_2$). To compute the normalized distance (w_l), intermediate routers use the function $RI(y_l)$ to determine the energy state's index that can handle the load y_l, while the function $R(x)$ retrieves the operating rate for the energy state index x. The destination will report back to the source the amount of traffic that has to be removed ($min(w_l)$) in order to move a link to a lower energy level. The intuition behind using the normalized distance is that it represents the fraction of traffic that each flow passing over the link needs to remove to enable the link to operate in a lower energy state.

Next, the sources attempt to shift the traffic away from the candidate links. An edge router computes periodically (every maximum RTT for all paths it sees) a change in traffic Δz_j^i as shown in Equation 3. If a path passes over a link which is a candidate for moving to a lower energy state, the source is supposed to decrease the traffic proportionally to the link's normalized distance from the lower level. Further, B^i is the number of paths that can absorb additional load without increasing their energy levels, while the X^i is the traffic which should be spread evenly among these paths. It is worth noting that all decisions are taken locally. Finally, Equation 4 ensures that the sending rates are kept constant.

Although the general idea sketched above looks straightforward, there are a few points worth discussing. Equation 2 assumes that there is enough spare bandwidth on alternative paths to absorb additional load without moving one of their links to higher energy levels. However, this is not always the case. Sometimes, it takes more than one round for some links to be moved to the lower energy level, and some of them never get to be moved. A link's chances depend on the spare bandwidth present on alternative paths (A_j^i), which we discuss in Section 3.4.

3.2 Leveraging sleeping of links

As depicted in Figure 1, it is possible that future hardware will feature a high value of C (high baseline power consumption) and small savings from adjusting the operating rate. Specifically, since implementing the logic for putting a network element to "sleep" is easier and less expensive than sophisticated scaling techniques, it is likely that the first power-aware networking hardware generation will feature only the sleep functionality. In this case, a traffic engineering algorithm should aim to aggregate traffic on as few links as possible, to allow the rest of the links to sleep. In this section, we show how EATe accomplishes this goal.

$$M(z_j^i) = \begin{cases} 1 & H_{lj}^i = 1 \wedge y_l < \mathbf{dm}; \\ -1 & H_{lj}^i = 1 \wedge y_l > \mathbf{U_b}; \\ 0 & \text{otherwise}; \end{cases} \quad (5)$$

Intuitively, in this scenario EATe tries to push as many links as possible into a sleep mode, while observing two boundary conditions. First, EATe is not allowed to increase the maximum link utilization determined by an ISP. Second, the energy states between sleeping (utilization of 0) and the bottleneck link utilization are considered as one large energy level. In this case EATe does not need to worry about operating rate adaptations, because potential benefits would likely be marginal (Figure 1). This enables EATe to increase the number of links that are capable of absorbing additional load without significantly affecting the overall energy consumption. EATe's equations are similar to those from previous section except packet marking $M()$. It is shown in Equation 5, where U_b represents the upper bound on link utilization. An ISP might decide to set U_b to a lower value to be able to accommodate a significant increase in traffic demand without activating links that are sleeping. On the other hand, if the ISP's goal is to maximize the

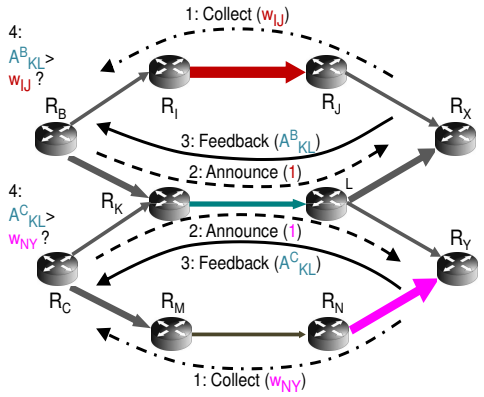


Figure 4: EATe in action with two edge routers (R_B and R_C) simultaneously attempting to move traffic from links $R_I - R_J$ and $R_N - R_Y$ onto the same link ($R_K - R_L$). Router R_K uses forward announcements and an XCP-like controller to compute explicit feedback for each source to prevent overshooting the target utilization.

number of inactive links when EATe takes control, it would set U_b to a higher value to maximize opportunity for traffic aggregation.

3.3 Leveraging sleeping of routers

It is well known that the chassis of a router consumes power even if just one of its links is active. Thus, the third technique we use in EATe seeks to shift traffic away from all links belonging to a router, enabling the entire chassis to sleep. Here, we assume that control-plane protocols could be modified to allow routers to stay in a sleep state for longer periods of time. The mechanism we use is similar to the one for putting links to sleep. An important change is in the value that is reported during the collect phase. Each intermediate router adds up the link utilization of its links and reports the sum to each source that sends traffic through it. Each source then shifts as much traffic as it can from the router with the smallest aggregate utilization it knows about. Thus, if we imagine the algorithm operating over a set of rounds (each equal to the maximum RTT), in each round the router with the smallest aggregate utilization "wins" the right to have all its traffic removed.

3.4 Ensuring stability

While shifting traffic from one link to another, we face the following fundamental problem: how do we ensure that two or more sources, which are simultaneously making independent adjustments, do not inadvertently increase the utilization of the target link beyond its desired level? If we do not prevent this from happening, we might experience 1) oscillations, when the sources move traffic from the "offending" link back to the original set of links, and back again [16], 2) unnecessary packet loss, and 3) wasted energy due to rate changes. We refer to this problem as the one of *ensuring stability*. Figure 4 shows one such scenario, in which the sources R_B and R_C simultaneously shift traffic from the top and bottom paths onto link $R_K - R_L$.

We ensure stability by using explicit feedback from intermediate routers. Specifically, the sources *announce* their intent to shift traffic on select paths (step 2 in Figure 4). Within these paths, routers count the number of announcements (n) in an interval of T_p seconds (max RTT). Each router then computes the slack ($R(RI(y_l)) - y_l$) until the next higher operating rate, given the current utilization of its link on the path. In addition, the router takes the queue size (Q) into account, computes the amount of new load it is willing to accept on the link (Φ), and spreads it across the set of sources that announced their intent to move traffic onto the link. Strictly taking the full slack and dividing it by the number of flows could cause oscillations due to traffic changes. Thus, we need a controller that is stable. One such controller is the one used in XCP [15] and TeXCP [14]. We adapt this controller for our needs, and compute the aggregate feedback Φ as follows:

$$\Phi = \alpha_x \cdot T_p \cdot (R(RI(y_l)) - y_l) - \beta_x \cdot Q \quad (6)$$

where α_x and β_x are constants chosen in a way that ensures stability [15]. If $\Phi > 0$, $A^i_j = \Phi/n$ is the per-source feedback (step 3 of Figure 4). Therefore, we extend Equation 3 by bounding Δz^i_j to A^i_j.

3.5 Selecting the drop margin

EATe is sensitive to the drop margin (dm) parameter. The drop margin implicitly determines the number of links that EATe tries to push to lower energy levels ("triangles" in Figure 3). Too small a number would lead to very few drop candidates, i.e., links ready to be pushed to lower operating rates. On the other hand, choosing too large a drop margin would decrease the number of alternative paths ("circles" in Figure 3). Setting the proper value of the drop margin is therefore an important challenge.

We used the following analysis to derive the drop margin value for our experiments. First, we assume that the link utilization within two adjacent energy levels is uniformly distributed. Then, we also assume that the links that are willing to accept additional load without sacrificing energy consumption will take an equal share of the aggregate traffic that is going to be shifted from the links which are within the drop margin. In this case, we have:

$$\begin{aligned} L_{down} &= \mathbf{dm} * N \\ T_{shift} &= \mathbf{dm}^2/2N * (R(i+1) - R(i)) \\ L_{up} &= \mathbf{dm}^2/(2 * (1 - \mathbf{dm})) * N \\ E_{savings} &= L_{down} - L_{up} \end{aligned} \quad (7)$$

L_{down} represents the number of links that we want to push to the lower energy levels, with N being the total number of links in the network. The aggregate traffic that is going to be shifted is T_{shift}, while the number of links that will go to higher energy levels in return is L_{up}. Finally, we maximize $E_{savings}$ by first differentiating Equation 7. Solving the resulting equation for the optimal $\mathbf{dm}$ value yields 0.42.

3.6 Deployment issues

Collecting and disseminating information. To maximize the protocol's deployment chances, it is important to minimize the changes that are required in existing packet headers and protocol implementations. Collect messages need to be able to carry the smallest link "waste" (normalized distance to lower operating rate). In case of routers that need to sleep, these messages need to carry the aggregate router utilization. EATe edge routers need to be able to announce their intention to shift traffic on a given link. In addition, EATe core routers need to be able to send back explicit feedback containing the amount of load that can be shifted onto their links. Finally, the edge routers need to determine the maximum RTT (and set T_p, the interval between traffic adjustments, to that value). While EATe could use separate packets to transmit the rates, the max RTT, and the feedback (as in TeXCP [14]), we believe that it might be possible to transmit the required information by marking bits in the packet header , e.g., by leveraging a recently proposed framework for deploying explicit feedback congestion control protocols [24].

The core routers themselves would require only small chan-ges. First, they would have to track the utilization of every link, which should not be difficult. Second, they should count the number of sources that are interested in shifting traffic onto each of their links. The counting process lasts for a limited time interval, and should therefore not pose a problem. The edge routers would have to run a more complex algorithm that decides how to balance the load among paths. However, this is in accordance with the current Internet practice in which edge routers are allowed to be more intelligent, at the expense of slower packet forwarding rates.

Hardware characteristics. By letting the routers decide the type of explicit feedback they provide, EATe easily accommodates heterogeneous equipment (different link speeds, number of operating rates, and number of links at the routers). In addition, EATe can easily adapt its policy for picking candidates that are to be moved to the lower energy levels for various network devices (characterized by the different C parameter and the $f_r()$ rate-based power consumption function. In essence, the routers would simply prefer links whose potential for energy savings is larger. For example, in the case of hardware with significant savings due to rate adaptation and large C, EATe would prioritize shifting down the links that are in the leftmost and the rightmost regions in Figure 3.

Scope of deployment. EATe is perhaps best suited for traffic management within an ISP, where there is a sufficient amount of trust among the routers. EATe does not change ingress or egress points for any source-destination pair, and it does not change the amount of traffic entering and leaving the ISPs network; hence, EATe does not affect the ISP preference for one egress for a particular set of source-destination pairs over others. Finally, EATe balances traffic only across paths that are given to it by the routing algorithm.

EATe could be used in wider deployments, e.g., involving the endhosts, if the protocol endpoints can be trusted to send traffic at the calculated rate. Not all protocol participants would have to be aware of the multiple paths, though, as the edge routers could shape the traffic on behalf of other participants.

Impact on reliability. EATe does not affect the time to detect failure, and it does not interfere with any lower level failover mechanisms (e.g., SONET). However, EATe might need time to: 1) wake up any target links (traffic recipients) that might be sleeping and 2) shift traffic toward them. We explore this aspect in our evaluation.

Impact on latency. Besides achieving significant power savings, EATe should ensure that all client traffic meets the relevant service level objectives (SLOs). Details about the trade-off between power savings and latency under different network utilization is available here [23]. In short, the impact of traffic aggregation on latency is negligible at low utilization levels (up to 40%) which is a maximum operating region for most ISPs [6]. Our discussions with an ISP in Europe reveal that link utilization is kept around 20% so that, even under failure, no link utilization goes above 40%. This is one more reason why EATe's impact on latency should be small. In any case, the ISPs can control the maximum link utilization by changing the U_b parameter as explained in Section 3.2.

4. EVALUATION

Our experimental evaluation addresses the following questions: **1)** Can EATe save energy using different network hardware and achieve low aggregate link utilization, without affecting the sending rates? **2)** Is EATe stable under changing traffic demands? **3)** How far is EATe from an optimal solution? **4)** Is EATe's impact on network reliability acceptable?

4.1 Experimental setup

As the aforementioned power-saving features have not yet been implemented in commercial routers, we primarily use ns-2 simulations to explore the benefits of our approach on small- and large-scale topologies. Our baseline is the TRUMP [12] code, which presents the state of the art in traffic management. The goal was to demonstrate EATe's savings against a traditional (energy-agnostic) TE algorithm. The experimental setup is summarized in the following.

ISP	Cities	Links	Flows	Max paths	Avg paths
Abovenet	19	68	20	5	3
AT&T	115	296	50	6	3.72
Genuity	42	110	30	4	3.27
Sprint	52	168	50	8	4.5
Tiscali	41	174	50	10	4.44

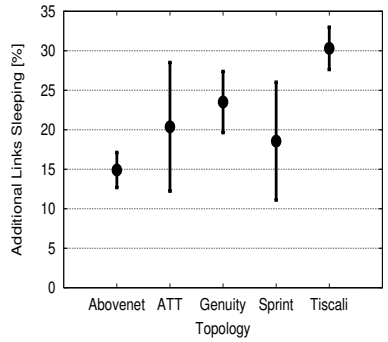
Table 1: Summary of Rocketfuel ISP topologies

Topologies. We run our algorithm on the ISP topologies published by the Rocketfuel project[21]. Table 1 summarizes these topologies, listing for each the number of links, flows, and alternative paths for any given source-destination pair. The alternative paths are computed by choosing several intermediate points in the network, patching together the corresponding shortest paths, and choosing the set of paths which results in the highest degree of edge disjointness.

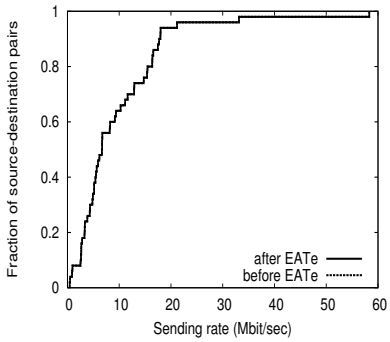
We leave the link latencies as determined by the Rocketfuel mapping engine. These topologies do not originally have link capacities assigned. We keep the values chosen in [12]: links are assigned 100 Mbps if they are connected to an end point with a degree of less than seven, otherwise they are assigned 52 Mbps. These settings mimic the reality in which core routers have fewer links that are running at higher speeds, relative to other routers in the topology.

Future hardware. We run all experiments under the assumption that each link is capable of operating in one of four uniformly distributed operating rates (this was the number of rates used in [20]). In rate adaptation experiments, we assume the use of hardware capable of frequency and dynamic voltage scaling, leading to quadratic energy savings between different rates. We also assume that the hardware can automatically adjust the operating rate to match the offered load. Thus, the savings we report are on top of those that are possible with local adaptation when it is applied to the results obtained by first running TRUMP.

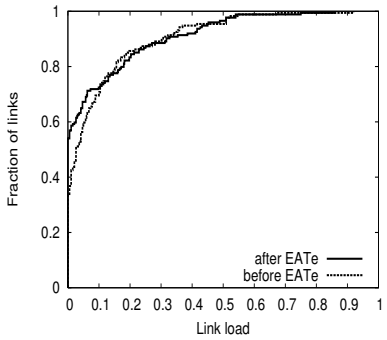
Traffic matrix. In each major category of experiments we run 11 simulations, varying the values of the TRUMP parameter w between 0.5 to 1.0. The parameter w is used to control the trade-off between the users' and the operators' goals (the higher w, the more preference TRUMP gives to the operators' goals). Typically, higher values are needed to ensure TRUMP's convergence, which is why we start from $w = 0.5$. Different values of w result in different sets of active links and in different link utilization distributions. In addition, different values of w cause TRUMP to choose different sets of per-host sending rates. Thus, by varying w, we effectively subject EATe to 11 different traffic matrices for each of the 5 topologies. The details of the traffic presented to TRUMP are similar to those used to evaluate TRUMP itself [12]. While we leave more sophisticated traffic generation (e.g., as in REPLEX [5]) for future work, we believe that the traffic demands that we impose upon TRUMP, and indirectly upon EATe via the traffic that is admitted by TRUMP (e.g., Figure 5(b)), are adequate to illustrate the energy savings that will be attainable.



(a) Percentage of links made inactive state (relative to active links before EATe's run).



(b) EATe does not change sending rates (Tiscali topology). CDFs are identical.



(c) CDF of normalized link utilization before and after running EATe (Tiscali).

Figure 5: EATe's performance when moving links to the inactive state.

4.2 Leveraging sleeping of links

In the first set of experiments, we explore the performance of EATe when it tries to leverage the sleep mode of future line cards (Section 3.2).

Figure 5(a) shows that EATe completely removes traffic, averaged over 11 values of w (error bars represent confidence intervals), from 15-31% of the links (21% on average) that were active after TRUMP's completion, while keeping the sending rates at the same level. These links can enter a sleep state and would in fact remain sleeping (because the sources would not be directing traffic towards them), unless there were significant changes in the traffic volume. Figure 5(b) shows a CDF of sending rates (traffic matrix) for data source-destination pairs in this experiment and confirms that EATe preserves the throughput levels set by TRUMP.

These results show that EATe can effectively use a straightforward energy-saving feature such as the sleep mode, which is likely to appear soon in the networking hardware. In today's hardware, the ratio between idle and maximum power is high (e.g., 0.8 [3]). Thus, a vast majority of the achieved percentage of sleeping links would materialize as real energy savings. For example, if we take power consumption figures for existing popular hardware from [3] (an OC-48 card consumes 70 W in the idle state, and approximately an additional 15 W when it is transmitting at its maximum rate), and apply them to the link utilizations in a representative topology (Tiscali, with 30% additional links sleeping), we observe a substantial net energy reduction of 28%.

EATe produces consistently better savings on Tiscali than it does on the rest of the topologies. This topology offers a large number of alternative paths to absorb traffic from any given link, thereby providing EATe with more opportunities to completely shift traffic away from a larger number of paths (and links). Abovenet does not offer many alternatives (40% of paths have no alternative choice), which explains why in this case the savings are smaller than on the rest of the topologies.

Figure 5(c) shows a CDF of the normalized link utilization, for the Tiscali topology, before and after EATe runs (the rest of the topologies are qualitatively similar). Apart from clearly showing the links that have all traffic removed (on the Y axis), this figure indicates that EATe did not significantly perturb the link utilizations, and did not make dramatic increases to accomplish this task.

Finally, we compare EATe's performance to the optimal, off-line solution. In order to compute the optimal number of links that need to remain active, we feed the source-destination pairs, the sending rates, and the topology to a mixed-integer programming problem formulated in MATLAB. Due to time constraints, we run these experiments for only one value of w. As Figure 6 demonstrates, EATe is within 15% of the optimal solution. We did not have sufficient time to observe the results for larger topologies; this is consistent with previously reported experiences [3].

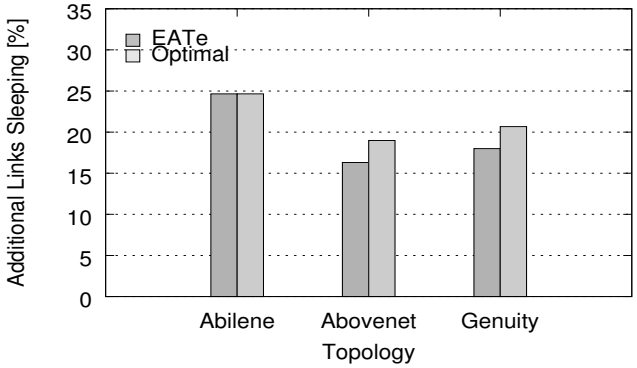
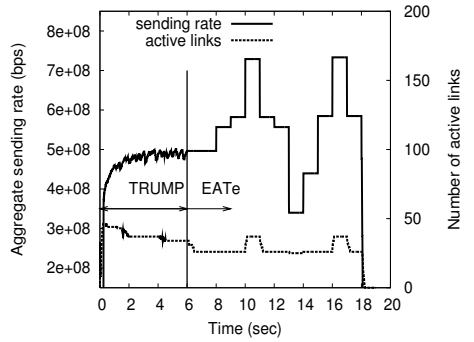


Figure 6: EATe's ability to move links to the inactive state is close to optimal.

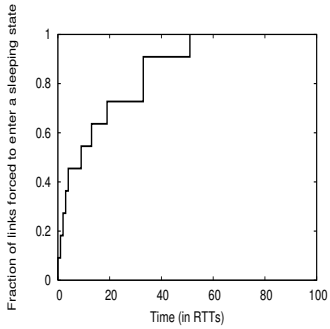
4.2.1 Stability under traffic changes

To demonstrate EATe's stability and the ability to handle repeated changes in the traffic volume, we start an experiment in the Abovenet topology with 20 source-destination pairs and $w = 0.55$. Figure 7(a) shows how the aggregate sending rates and the number of active links vary over time. EATe makes only a small number of changes to power states of the links after it starts running at $t = 6$ seconds, further reducing the energy consumption (shown on the Y2 axis) relative to the non-energy aware approach. In addition, EATe quickly converges and keeps the number of active links stable. Figure 7(b) shows the CDF of convergence time for the links that are moved to the inactive state. The figure depicts that EATe takes only 10 RTTs for 50% of the links, and around 50 RTTs for all of the links to enter the inactive state.

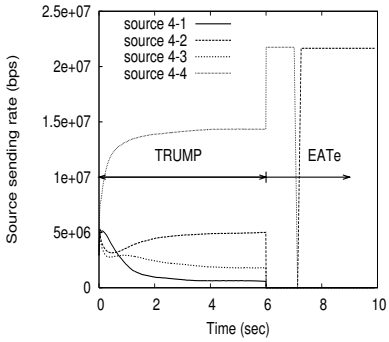
Starting at $t = 8$ seconds (Figure 7(a)), we stress EATe by repeatedly changing the traffic volume every second. Specifically, we increase or decrease the traffic demands by a random amount that is up to 50% of the original traffic allowed by TRUMP. EATe's strategy is to accommodate as much traffic as possible by using the cur-



(a) EATe quickly achieves a stable, low-energy state, and successfully handles repeated changes in the traffic volume.



(b) EATe converges within 10 RTTs for 50% of links (CDF: time to get links to sleep starting at $t = 6$ s)



(c) EATe quickly restores traffic after link failure at at $t = 7$ s.

Figure 7: Experiments in the Abovenet topology with sleeping of links under traffic changes and link failure.

rently active links, and to wake up some of the sleeping links only if it is necessary. Accordingly, EATe can increase the number of the sleeping links if that traffic volume decreases sufficiently. This experiment shows that EATe can quickly: 1) deal with dramatic changes in the traffic demand with few or no changes in the number of active links, and 2) reach a stable point even under changing traffic demands.

4.2.2 Handling link failures

To demonstrate that EATe does not significantly affect network reliability, we conduct an experiment in the Abovenet topology involving link failure. Figure 7(c) shows the traffic originating from one particular source (4) across four alternative paths. Once EATe starts running at $t = 6$ seconds, it consolidates traffic originating from three paths onto a path labeled 4-4 without changing the sending rate. At $t = 7$ seconds we fail a link on this path. We then observe that EATe quickly moves the affected traffic onto another alternative path (in this case 4-2), with small amount of packet loss. We note that it is likely that any other online traffic engineering algorithm would also lose the packets traveling over the affected path. Here we assume that it takes 100 ms for the link failure information to propagate to the sources, and 10 ms to wake up a target link (upper bound on the estimate in [20]).

4.3 Leveraging sleeping of routers

In this set of experiments we explore EATe's ability to remove traffic from links in a way that maximizes the number of routers with no traffic, and enables them to enter a sleep state (Section 3.3). After TRUMP's run, only a small fraction of routers ends up unused. Figure 8(a) shows that EATe succeeds in enabling to sleep an additional 10-24% (16% on average) of the routers that were active before EATe, while keeping the sending rates at the level determined by TRUMP. In the absence of significant changes in traffic, EATe will keep traffic away from these routers, allowing them to remain sleeping.

Figure 8(b) shows that EATe puts routers to sleep by making small overall changes in link utilization (shown is the Tiscali topology; EATe behaves similarly on the other topologies). We also compute the router utilization as a sum of the attached link utilizations. Figure 8(c) shows the router utilization before and after EATe. Although each source ranks routers independently and makes a local traffic-shifting decision, routers with a utilization level close to 0 (the area of interest is in the lower left part of the graph) end up in having all their traffic removed. We also see that

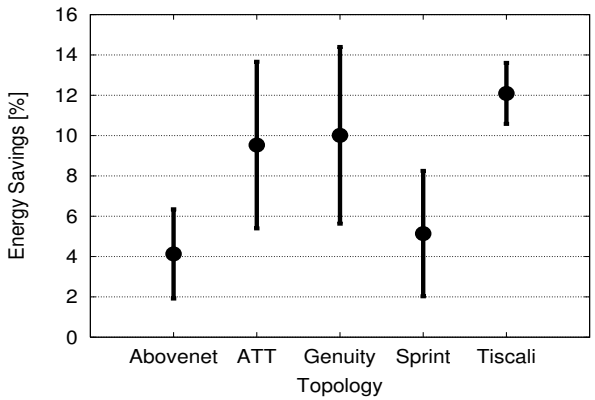


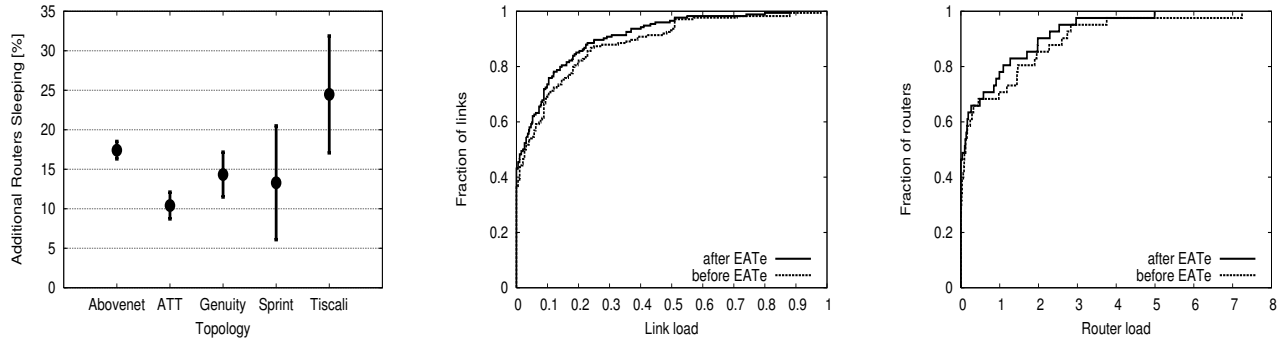
Figure 9: Energy savings by moving links to lower energy levels when rate adaptation is preferred.

EATe does not significantly increase the utilization of routers that remain active. Therefore, we expect the actual energy savings to be close to the fraction of routers that went to sleep.

4.4 Leveraging rate adaptation

Next, we examine EATe's performance on more sophisticated future hardware that will waste little energy while idle, and will be able to vary the operating rate to match the offered load. Figure 9 depicts the energy savings in the Rocketfuel study topologies, while the sending rates are kept constant. Computing the actual energy savings is difficult because the hardware does not yet exist, but to get a general idea of the possible savings, we apply values to our model that are similar to those in [20]. We assume that frequency and Dynamic Voltage Scaling are in place, but that due to various artifacts they result in quadratic savings between two adjacent operating rates. In addition, we assume that a line card wastes an additional 20% of its maximum power due to intrinsic hardware losses. With these settings, we can see from Figure 9 that the average energy savings are 8%.

Figure 10 zooms in on the behavior of EATe on one representative topology (Tiscali) and shows a CDF of the link utilization before and after EATe runs. The artifacts due to EATe's adjustments manifest themselves as jumps in the number of links that have utilizations just short of the next discrete rate, as intended.



(a) Percentage of routers (relative those active before EATe's run) that can sleep.

(b) CDF of the normalized link utilization before and after EATe (Tiscali topology).

(c) CDF of router utilization before and after EATe (Tiscali topology).

Figure 8: EATe's performance when moving routers to the inactive state by removing traffic from all their links.

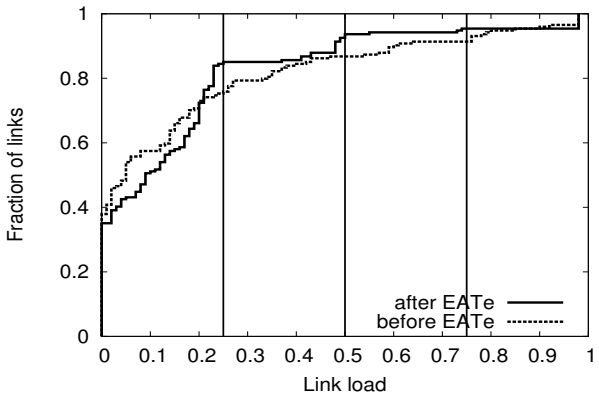


Figure 10: CDF of link utilization before and after EATe succeeds in moving links to lower energy levels (Tiscali topology, $w = 0.65$). Vertical bars correspond to distinct operating rates supported by the hardware.

The flat parts in the CDF right after the discrete operating rates (denoted by vertical lines) represent an evidence that EATe successfully pushes down links that are close to lower operating rates. EATe's energy savings exhibit high variance for different values of w for all topologies. The cause for this behavior is the power characteristic of the model (quadratic savings between two adjacent operating rates) and the differences in post-TRUMP link utilizations. EATe gains the most when it shifts a link from the highest setting to the next lower level. In the case shown, there were not many links at the highest rate that could be moved down.

Relative savings among the topologies are similar to those for the case of sleeping links, and can be explained by the number of alternative paths that are available for each link.

In this case, EATe's savings are smaller than in the case of moving links to a sleep state because the hardware which uses rate-adaptation is closer to the ideal goal of power-proportionality. Nevertheless, EATe achieves considerable savings by using a network-wide online approach. EATe makes a small number of changes to the power states of the links. Most importantly, in the absence of significant traffic changes EATe keeps links in their chosen power states and avoids costly switchings between different operating rates.

5. RELATED WORK

Gupta *at al.* [10] were the first to raise the issue of the potential energy/power savings in the wired Internet. To the best of our knowledge, our work is the first embodiment of their vision of network-wide coordinated sleeping. The problem of managing energy consumption costs in desktop PCs [2] and LAN switches [7, 9] was the first to receive attention. There has also been work on Adaptive Link Rate (ALR) for Ethernet [8].

Chabarek *at al.* [3] have recently argued for power awareness in network design and routing. They conduct valuable experiments with popular routers and create a router power consumption model. They then proceed to apply this model in realistic network configurations and use mixed-integer optimization techniques to reduce power consumption while preserving a good network performance. They report running times of a few hours for their off-line approach, and its inability to complete for two realistic topologies. Similarly, Chiaraviglio *et al.* [4] try to determine the minimum set of routers and links that can accommodate a given traffic demand. However, the paper does not discuss how the algorithm would be deployed and how the set of active elements would transition from one configuration to another as the traffic demand is changing. Finally, assuming ideal knowledge about incoming traffic, Mahadevan *at al.* [17] quantify the potential power savings in data centers by resorting to topology control and the workload placement. Relative to these works, EATe offers a mechanism which delivers on the promise of power savings. It is an online and scalable approach that could be deployed along with future hardware that supports sleeping and rate-adaptation.

6. CONCLUSIONS AND FUTURE WORK

While reducing the energy consumption of network clients, servers, and switches did receive considerable attention recently, there has been very little effort on reducing the energy consumption of the Internet backbone and the routers leading to it. If unattended, however, their energy consumption could become one of the Internet's dominant energy factors as the Internet traffic continues to grow exponentially. This paper takes a fresh look at the traffic management problem from the standpoint of energy-awareness. First, we assume that energy-aware networking hardware will be deployed in the near future. We then devise three performance-preserving techniques that produce considerable energy savings on top of what could be expected from a state-of-the-art traffic management algorithm deployed over the future hardware. To the best of our knowledge, we present the first online, energy-aware traffic management

technique. With just a simple mechanism such as the sleep state of line cards, EATe can help bridge the gap to the energy-proportional networking hardware.

7. ACKNOWLEDGMENTS

We are grateful to the TRUMP authors for having allowed us to use their source code and the experimental setup. Nedeljko Vasić is supported by an IBM PhD Fellowship.

8. REFERENCES

[1] Power and Thermal Management in the Intel Core Duo Processor. *Intel Technology Journal*, 10, 2004.

[2] Y. Agarwal, S. Hodges, J. Scott, R. Chandra, P. Bahl, and R. Gupta. Augmenting Network Interfaces to Reduce PC Energy Usage. In *NSDI*, 2009.

[3] J. Chabarek, J. Sommers, P. Barford, C. Estan, D. Tsiang, and S. Wright. Power Awareness in Network Design and Routing. In *INFOCOM*, 2008.

[4] L. Chiaraviglio, M. Mellia, and F. Neri. Energy-aware Backbone Networks: a Case Study. In *GreenComm*, 2009.

[5] S. Fischer, N. Kammenhuber, and A. Feldmann. REPLEX: Dynamic Traffic Engineering Based on Wardrop Routing Policies. In *CoNEXT*, pages 1–12, New York, NY, USA, 2006. ACM.

[6] J. Guichard, F. le Faucheur, and J. P. Vasseur. *Definitive MPLS Network Designs*. Cisco Press, 2005.

[7] C. Gunaratne, K. Christensen, and B. Nordman. Managing Energy Consumption Costs in Desktop PCs and LAN Switches with Proxying, Split TCP Connections, and Scaling of Link Speed. *Int. J. Netw. Manag.*, 15(5):297–310, 2005.

[8] C. Gunaratne, K. Christensen, B. Nordman, and S. Suen. Reducing the Energy Consumption of Ethernet with Adaptive Link Rate (ALR). *IEEE Trans. Comput.*, 57(4):448–461, 2008.

[9] M. Gupta, S. Grover, and S. Singh. A Feasibility Study for Power Management in LAN Switches. In *ICNP*, 2004.

[10] M. Gupta and S. Singh. Greening of the Internet. In *SIGCOMM*, 2003.

[11] R. Hays. Active/Idle Toggling with Low-Power Idle. http://www.ieee802.org/3/az/public/jan08/hays_01_0108.pdf, 2008.

[12] J. He, M. Suchara, M. Bresler, J. Rexford, and M. Chiang. Rethinking Internet Traffic Management: from Multiple Decompositions to a Practical Protocol. In *CoNEXT*, 2007.

[13] T. Heath, A. P. Centeno, P. George, L. Ramos, Y. Jaluria, and R. Bianchini. Mercury and Freon: Temperature Emulation and Management for Server Systems. In *ASPLOS-XII*, 2006.

[14] S. Kandula, D. Katabi, B. Davie, and A. Charny. Walking the Tightrope: Responsive yet Stable Traffic Engineering. In *SIGCOMM*, 2005.

[15] D. Katabi, M. Handley, and C. Rohrs. Internet Congestion Control for High Bandwidth-Delay Product Networks. In *Proceedings of ACM SIGCOMM*, August 2002.

[16] A. Khanna and J. Zinky. The revised ARPANET routing metric. *SIGCOMM Comput. Commun. Rev.*, 19(4):45–56, 1989.

[17] P. Mahadevan, P. Sharma, S. Banerjee, and P. Ranganathan. Energy Aware Network Operations. In *IEEE Global Internet Symposium*, 2009.

[18] P. Mahadevan, P. Sharma, S. Banerjee, and P. Ranganathan. A power benchmarking framework for network devices. In *NETWORKING*, 2009.

[19] M. Mandviwalla and N.-F. Tzeng. Energy-Efficient Scheme for Multiprocessor-Based Router Linecards. In *IEEE SAINT*, 2006.

[20] S. Nedevschi, L. Popa, G. Iannaccone, S. Ratnasamy, and D. Wetherall. Reducing Network Energy Consumption via Sleeping and Rate-Adaptation. In *NSDI*, 2008.

[21] N. Spring, R. Mahajan, D. Wetherall, and T. Anderson. Measuring ISP Topologies with Rocketfuel. *IEEE/ACM Trans. Netw.*, 12(1):2–16, 2004.

[22] R. S. Tucker. Modelling Energy Consumption in IP Networks. In *Cisco Green Research Symposium*, 2008.

[23] N. Vasić and D. Kostić. Energy-aware Traffic Engineering. Technical report, 2008.

[24] N. Vasić, S. Kuntimaddi, and D. Kostić. One Bit Is Enough: a Framework for Deploying Explicit Feedback Congestion Control Protocols. In *COMSNETS*, 2009.

[25] B. Zhai, D. Blaauw, D. Sylvester, and K. Flautner. Theoretical and Practical Limits of Dynamic Voltage Scaling. In *DAC*, 2004.