Toward Power-Efficient Backbone Routers

Jianyuan Lu* Liang Liu† Jun "Jim" Xu† Bin Liu*

* Department of Computer Science and Technoogy, Tsinghua University, China
† College of Computing, Georgia Institute of Technology, GA

{thlujy, liuliang142857}@gmail.com, jx@cc.gatech.edu, liub@mail.tsinghua.edu.cn

ABSTRACT

Recently, a new design framework, called GreenRouter, has been proposed to reduce the power consumption of backbone routers. In a GreenRouter, a line card is partitioned into two functional parts, namely, an InTerFace (ITF) part that is relatively much lighter and a Processing Engine (PE) part that is relatively much heavier, in power consumption. This partitioning allows ITFs to share the collective processing capability of PEs, which in turn allows a significant percentage of PEs to be put into sleep mode (to save energy) during periods of light link utilizations. In this paper, we study how ITFs can distribute traffic flows to the PEs so that the offered loads on all active PEs are near-perfectly balanced over time, and kept close to a target load (say 90%), so that the number of active PEs can be minimized. Since GreenRouter's original solution to this problem is quite crude, we propose a principled solution that has much lower system overheads and achieves better load-balance. Through both simulation studies and rigorous analyses, we show our solution can, with high probability, rapidly restore the near-perfect load balance among active PEs, after each PE overload event.

1. INTRODUCTION

Recently, how to design and engineer a power-efficient backbone network has emerged as an important research topic [4]. As almost all nowadays transmission links between backbone routers are optical links that use highly energy-efficient laser technologies, backbone routers account for the majority of the power consumption of running a backbone network. Indeed, a nowadays high-end backbone router consumes a massive amount of power. For example, a Cisco CRS-1 routing system, which can provide a maximum throughput of 92 Tbps, consumes as much as 1 megawatt power with maximum configuration [5]. Therefore, making backbone routers more energy-efficient is a key component in the integrated effort towards building an energy-efficient backbone network. Unfortunately, research efforts on designing energy-efficient routers are mostly lacking.

There is indeed considerable room for improvement in the energy efficiency of backbone routers. For one thing, it has been shown in the literature that the maximum average utilization ratio of backbone links is quite low, *e.g.*, no more than 40% as reported in [7]. Ideally a router having an average utilization ratio of α should be designed to consume only α times the maximum power needed when the router is fully loaded (*i.e.*, 100% utilization). However, due to a lack of engineering for power-efficiency, a modern router typically consumes close to the maximum power, even when it is lightly loaded or idle. For example, our measurements of a HUAWEI NE40E-X8 router (sold until 2010) show that the idle power consumption is more than 97% of its maximum power consumption.

Recently, a new design framework, called GreenRouter, has been proposed to reduce the power consumption of backbone routers [9, 11]. Figure 1 illustrates the main difference between a conventional router and a GreenRouter. This GreenRouter design is based on the observation that line cards account for most of a (conventional) router's energy consumption, but have to be always on (*i.e.*, in the active mode) even when the network traffic is light. Hence in a GreenRouter, a line card is partitioned into two functional parts, namely, an InTerFace (ITF) part that is relatively much lighter and a Processing Engine (PE) part that is relatively much heavier, in power consumption. The ITFs implement physical and link layer functions, including receiving/sending packets, framing, error detecting, *etc.* The PEs implement network and upper layer functions, including IP lookup, packet classification, IP security protocol (IPSec), deep packet inspection (DPI), *etc.* As shown in Figure 1, ITFs and PEs are then connected by an additional switching fabric so that ITFs can share the collective processing capability of PEs. This (PE) resource-pooling technique allows a significant percentage of PEs to be put into sleep mode (to save energy) during periods of light link utilizations.

With such a partitioning of line cards into ITFs and PEs in a GreenRouter, each incoming packet is first received by the corresponding input ITF module, then forwarded by the first switching fabric to an active PE (*i.e.*, not in the sleeping mode) for processing, and finally forwarded by the second switching fabric to its destination output ITF module. While ITFs still have to be always on to process incoming or outgoing packets, since each ITF consumes much less power than a PE, as stated earlier, these ITFs will not become a drag for the overall energy efficiency of a GreenRouter. Note that the extra switching fabric needed for (PE) resource pooling is not a drag for the energy efficiency either, because a switching fabric accounts for only a tiny percentage (about 3%) of the energy consumption of a router. Therefore, GreenRouter trades a reasonable fixed cost (of an extra switching fabric) for potential significant power savings over time.

In this paper, we tackle a critical research problem in the GreenRouter design, that is, upon the arrival of a packet to an ITF, the ITF needs to decide which active PE the packet should be forwarded to. Ideally, we would like to keep the offered load on all active PEs perfectly balanced and close to 90% or so, so that the number of active PEs can be minimized to save power to the maximum extent possible. This goal seems easy to achieve, *e.g.*, by letting every input ITF uniformly spread its incoming packets to all active PEs. However, to prevent the reordering of packets in an application (TCP or UDP) flow, all packets in a flow should be forwarded to the same PE, which such packet-level spreading cannot ensure. With this constraint, our research problem becomes much harder: Each ITF has to decide, without communicating with other ITFs (about their load conditions and assignment decisions), for each new incoming flow, which PE it should be assigned to, and yet these decisions, made independently by individual ITFs, collectively result in a near-perfect load-balance across all active PEs (with high probability).

GreenRouter's solution – to this problem of assigning flows to active PEs for near-perfect load-balance – is quite crude, has high systems overheads, and cannot consistently achieve a high level of load-balance, as will be explained shortly in Section 2. In this work, we propose a principled solution that has much lower system overheads and achieves better load-balance. Its basic idea is as follows. Each ITF groups its incoming flows into flow bundles via hashing (on the <*src IP, src port, dst IP, dst port, protocol*> five-tuple). To ensure correct packet ordering, all flows – and all their packets – within a bundle are forwarded through an active PE. In other words, the entire bundle is assigned to an active PE.

Now the question becomes how to assign these bundles to the set of active PEs to achieve and maintain a near-perfect load-balance. Our approach starts very simple: Each ITF starts off assigning each of its bundles to an active PE uniformly randomly (via another hash function). By doing so, ITFs collectively forward on average the same amount of incoming traffic to all active PEs. In fact, even this very simple approach works pretty well – *i.e.*, achieves an acceptable level of load-balance – most of time. However, with small but nonnegligible frequency, load imbalance does occur as an unavoidable consequence of this randomization (via hashing) process, and the births and the deaths of elephant (*i.e.*, high-rate) flows. It is to correct such load imbalances where a principled approach becomes necessary. In our solution, when an active PE becomes overloaded, it selects a small number of flow bundles and asks the corresponding ITFs to relocate them. Each input ITF receiving such a request makes a randomized decision, independent of other affected input ITFs, which other active PE the bundle at issue should be relocated to. Each such randomized decision is computed using (an independent instance of) a randomized algorithm that takes input as the load factors of all active PEs reported infrequently (say every 1 second) to all input ITFs. Through both simulation studies and rigorous analyses, we show this randomized algorithm can, with high probability, rapidly restore the near-perfect load balance among active PEs.

The rest of the paper is organized as follows. In Section 2, we introduce previous works related to our work. In Section 3, we formalize our load-balancing problem and describe our approach to it. We also show the performance guarantee through theoretical analysis in this section. In Section 4, we evaluate the effectiveness and power-efficiency performance of our load-balancing scheme. Section 5 concludes this paper.

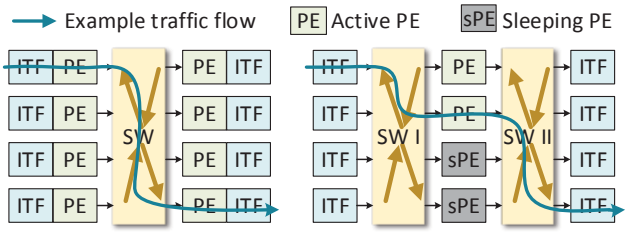

Figure 1: The comparison of conventional router (Left) and GreenRouter (Right)

2. RELATED WORK

In this section, we describe previous works on reducing the power consumption of routers. They can be grouped into the following three categories by the "greening" technique each of them employs.

Component sleep: Their main idea is to put some components of a router, such as network processor, traffic manager *etc.*, into sleep or low-power mode when all links are idle [8, 12, 6, 10]. These techniques do not help much for a backbone router, since its high-speed links are almost never completely idle.

Architecture redesign: We have described the *GreenRouter* architecture [9, 11] in Section 1. The goal of this work is to propose a new and principled load-balancing scheme for the *GreenRouter*. In the original scheme, flows are also grouped into bundles via hashing, but a (nonempty) bundle typically contains only one or two active flows, whereas in the new scheme, a bundle contains on average thousands of active flows. Hence, in the original scheme, a bundle may frequently transition between the empty state (containing no active flow) and the nonempty state (containing at least one active flow). Whenever a bundle transitions from the empty state to the nonempty state, the corresponding ITF needs to decide which PE to assign (more precisely to reassign) this "new bundle" to. Therefore, an ITF in the original scheme needs to make three to four orders of magnitude more assignment decisions than an ITF in the new scheme. In other words, the original scheme incurs much more system overheads than the new scheme.

In the original scheme, a "new bundle" is assigned to the active PE that is currently least loaded. This greedy heuristic does not work well due to the following greed synchronization problem. PEs update ITFs on their load conditions rather infrequently, so during two consecutive updates, hundreds of such assignments could be made independently by distinct ITFs. Most of them will go to the least loaded PE, possibly making it overloaded. This greedy heuristic would be even worse suited for the new scheme, where a bundle, as explained earlier, becomes thousands of times heavier (*i.e.*, containing more flows and traffic).

Traffic engineering: Zhang *et al.* [16] study how an Internet Service Provider (ISP) network can save power through traffic engineering. In this approach, network traffic in certain links and routers will be rerouted through other parts

of the network topology so that these links and routers can be put into the sleep mode. Similar approaches have been proposed also in [8, 1].

3. THE LOAD-BALANCING PROBLEM

In the following, we precisely state the load-balancing problem in Section 3.1 and describe our approach in Section 3.2. In Section 3.3, theoretical analysis is shown to ensure good performance of our new scheme.

3.1 Systems Model

As explained earlier, each ITF splits incoming traffic into flow bundles via hashing on the flow identifier of each packet. Each bundle typically aggregates thousands of flows together. We choose this level of aggregation for two reasons. First, we observe through our measurements of Internet traffic traces (described later) that the rate of each such bundle is with high probability quite stable over time, which helps maintain load-balance for as long as possible. Second, most of such bundles are still "light" enough to be relocatable if needed. To ensure correct packet order, all packets within a bundle are forwarded (*i.e.*, assigned) to the same (active) PE for processing.

For the simplicity of presentation, we make the following three homogeneity assumptions. First, the number of ITFs is the same as the number of PEs, which we denote as N. Second, each ITF and each PE has an identical capacity of 1.0. Third, all ITFs use the same hash function, and hence split their traffic into the same number of bundles, which we denote as E.

Each ITF needs to assign, each of its E flow bundles, to an active PE. These assignments are maintained in a mapping table, implemented simply as an array of E entries to allow for line-speed lookups. At each ITF, each bundle is assigned to an active PE uniformly randomly (say using another hash function) to start with. These assignments change over time as a result of load-rebalancing actions taken by the ITF.

There are three types of load-imbalance events that can trigger a load-rebalancing action. The first type is that a single active PE becomes overloaded. An active PE is considered overloaded when its load reaches or exceeds a pre-specified overload threshold β. This threshold β is typically set to around $0.95 = 95\%$ in practice, because when the load of a PE exceeds that, its quality of service (QoS), in terms of packet delay and drop rate, starts to deteriorate. The second type is that the average load across all active PEs reaches or exceeds a pre-specified target load α. When this happens, a PE that is currently in the sleep mode needs to be turned on, and loads shifted from other active PEs to this new active PE. The target load α is typically set to $0.9 = 90\%$ to minimize the number of active PEs without significantly impacting the QoS. The third type is when the average load across all active PEs drops below a threshold slightly smaller than α (say $\alpha - 0.03$). When this happens, the least loaded active PE will be put to sleep, and its loads distributed to other active PEs.

In this work, we focus on the first type of load-imbalance event in describing our load-rebalancing scheme for two reasons. First, the first type of event happens much more frequently than the second and the third types, as we observe in real-world traffic traces. Second, our load-rebalancing scheme works effectively also for restoring load-balance in a load-imbalance event of the second or the third type. Before we proceed to describe our load-rebalancing scheme, however, we need to briefly describe the role a PE plays in the scheme.

As a part of the load-rebalancing scheme, each PE needs to measure the rate of each bundle sent to it from an ITF. This is straightforward (through packet/byte counting), since when a PE receives a packet, the PE knows which ITF sends the packet and its bundle ID (by hashing its header). Each PE reports only its overall load (not that of each bundle) periodically and infrequently (say once a second) to the router controller and every ITF; Clearly, a very small amount of communication is involved here. The load reports from all PEs are used by the router controller to determine whether an aforementioned load-imbalance event of the second or the third type occurs. For an ITF, however, these load reports serve as parameters to the randomized load-rebalancing scheme, if and when the ITF is asked by an overloaded PE to help diffuse its excess load, as will be explained next.

3.2 Our Load-Rebalancing Scheme

In this section, we describe in details our load-rebalancing scheme that reassigns the excess loads from an overloaded active PE to other relatively lightly-loaded PEs. Our load-rebalancing scheme is triggered whenever the load of an active PE γ (say $=96\%$) exceeds the overload threshold β (say $=95\%$). The overloaded PE then scans through the set of flow bundles ITFs assign to it for a subset of them to be shifted to other active PEs. The goal is to shift just enough excess load elsewhere for its remaining load to be around the aforementioned target load α (say $= 90\%$). In achieving this goal, we would like to relocate as few bundles as possible, because each such relocation incurs some systems overheads on the affected ITF. For example, to completely avoid packet reorder during the relocation, packets in the bundle at issue may need to be delayed for a short period of time (say a few milliseconds) before starting to be forwarded to its new PE. Clearly, selecting the "heaviest" (*i.e.*, with highest traffic rates) bundles to relocate minimizes the number of such bundles. However, if a flow bundle is too heavy, say having a rate larger than a threshold C, it is not a suitable candidate for relocation either, because relocating these "obese" bundles could risk overloading another active PE. Therefore, in our scheme, the overloaded active PE selects a subset S of heaviest bundles that are not heavier than C and that have a total weight of around $\gamma - \alpha$, for relocation. Once the set S of bundles to be relocated is chosen, the overloaded active PE sends, to the owner of each such bundle, a relocation request containing the ID of the bundle.

An input ITF, upon receiving such a request for relocating a bundle, executes the following randomized algorithm to decide where the bundle should be relocated. Suppose the active PEs are numbered $1, 2, \ldots, m$ and their recently reported loads are $l_1, l_2, \ldots, l_m$. For each active PE i, we consider its spare capacity to be $max\{0, \alpha - l_i\}$, written as $(\alpha - l_i)_+$ in short, where α is the aforementioned target load. Our algorithm is simply to select an active PE j with probability $p_j = (\alpha - l_j)_+ / \sum_{k=1}^{m} (\alpha - l_k)_+$ (*i.e.*, proportional to its spare capacity $(\alpha - l_j)_+$), and assigns the bundle to it.

In the course of resolving a (active) PE overload event, every such assignment is computed using the same randomized algorithm parameterized with the same probability distribution, because the involved ITFs have the same load readings

$< l_1, l_2, \ldots, l_m >$ of the active PEs. Each such assignment, however, is made independent of others, even when an ITF makes multiple such assignments (when it needs to relocate multiple such bundles). As explained earlier, making these random assignments independently avoids the aforementioned greed synchronization problem. In the next section, we will prove that this randomized algorithm, albeit simple, resolves a (active) PE overload event – without causing a new one – with overwhelming probabilities.

Many load-balancing scheduling policies have been proposed and analyzed in distributed systems literature [2, 3]. Our randomized algorithm actually implements one such policy called *Proportional Branching* [2]. Despite that, our contributions here are twofold. First, we carefully adapt this policy to our context both to minimize the systems overhead and to maximize its load-balancing efficacy. Second, we are able to derive a tight probability tail bound – that is unique to our context – on load-balancing inefficacy .

3.3 Theoretical Guarantee of Our Scheme

In this section, we state and prove the following theorem that tightly bounds the probability that ITFs, in an attempt to resolve a (active) PE overload event using the aforementioned scheme, collectively make another PE overloaded. We first introduce some notations that will be used in the theorem and the proof. Recall that the loads of active PEs before the rebalancing scheme are denoted as $l_1, l_2, \cdots, l_m$. Define $L_\alpha^- = \sum_{j=1}^m (\alpha - l_j)_+$ is the total free space of PEs. Let $w_1 \le w_2 \le ... \le w_n \le C$ be the weights of the n bundles to be relocated from the overloaded (active) PE and define $W = \sum_{i=1}^n w_i$ as their total weight. Suppose an active PE has load $l < \alpha$ before and has load l' after the rebalancing. The following theorem bounds the probability that $l' > \beta$ (*i.e.*, overloaded).

THEOREM 1. *If $W > \beta - l$, we have*

$$\mathbb{P}(l' > \beta) < \left(\frac{W}{L_\alpha^-}\right)^{\frac{W}{w_n}} \left(\frac{\alpha - l}{\beta - l}\right)^{\frac{\beta - l}{w_n}} \left(\frac{L_\alpha^- - \alpha + l}{W - \beta + l}\right)^{\frac{W - \beta + l}{w_n}} \quad (1)$$

Otherwise, we have $\mathbb{P}(l' > \beta) = 0$.

PROOF. The result follows when we let $X = l' - l$, $b_i = w_i$, $\forall i = 1, ..., n$ (so that $B = W$), $p = \frac{\alpha - l}{L_\alpha^-}$, and $a = \beta - l$, in Lemma 1 below. Note that p is the probability for each of these n bundles to be relocated to this PE according to the aforementioned randomized algorithm. □

We present some numerical results of (1) to put the probability bound into perspective. We use a (active) PE overload event that occurs during our model-based simulation study to be described in Sec. 4.2. We assume that 256 active PEs are used when the event happens. We set the target load α to 90% and the overload threshold β to 95%. During the overload event, the overloaded PE has a load of $0.96 = 96\%$, so the total load to be relocated is $W = 96\% - 90\% = 6\%$.

During this event, the total spare capacity L_α^- across all active PEs is about 0.8, which is 80% of a single PE's maximum capacity. We set the "obese threshold" C to 0.7%, *i.e.*, no bundle exceeding 0.7% of a PE's maximum processing capacity is eligible for relocation. This obese threshold is reasonable since it is exceeded by only the heaviest 1% of all flow bundles, according to our measurements of the real-world traces (described later in Sec. 4). Substituting these parameters into (1), and by the union bound, we obtain that the probability that at least one PE becomes overloaded after load-rebalancing is no more than 2.48×10^{-14}.

The following lemma is a Chernoff bound on the weighted sum of i.i.d. Bernoulli random variables. It is sharper than the Chernoff bound on the weighted sum of independent (but not necessarily identically distributed) Bernoulli random variables derived in [13].

LEMMA 1. *Let $\{b_i\}_{i=1,...,n}$ to be n positive numbers such that $0 < b_1 \le b_2 \le ... \le b_n$. $\{X_i\}_{i=1,...,n}$ are n identical and independent Bernoulli variables with probability p, $X = \sum_{i=1}^n b_i X_i$. Let $B = \sum_{i=1}^n b_i$. If $B > a$, we have*

$$\mathbb{P}(X > a) \le \left(\frac{B(1-p)}{B-a}\right)^{\frac{B}{b_n}} \left(\frac{p(B-a)}{a(1-p)}\right)^{\frac{a}{b_n}} \quad (2)$$

Otherwise, we have $\mathbb{P}(X > a) = 0$.

PROOF. Clearly, we need only to prove the case of $B > a$. Define $f(x,t) \equiv pe^{tx} + 1 - p$ so that $E[e^{tb_i X_i}] = f(b_i, t)$. Hence $E[e^{tX}] = E[\prod_{i=1}^n e^{tb_i X_i}] = \prod_{i=1}^n f(b_i, t)$. By the Chernoff bound, we obtain

$$\mathbb{P}(X > a) \le \min_{t>0} \frac{E[e^{tX}]}{e^{ta}} = \min_{t>0} \frac{\prod_{i=1}^n f(b_i, t)}{e^{ta}} \quad (3)$$

We claim that for any $i \in \{1, 2, ..., n\}$, we have

$$f(b_i, t) \le \big(f(b_n, t)\big)^{\frac{b_i}{b_n}} \quad (4)$$

To prove inequality (4), it suffices to prove that $\big(f(x,t)\big)^{\frac{1}{x}}$ is an increasing function of x on $(0, \infty)$, given any $t > 0$. To do so, we show that, its first derivative with respect to x, given any $x, t > 0$, is no smaller than 0, as follows.

$$\begin{aligned} &\frac{d \ln\left[\big(f(x,t)\big)^{\frac{1}{x}}\right]}{dx} \\ =&\frac{ptxe^{tx} - (pe^{tx} + 1 - p)\ln(pe^{tx} + 1 - p)}{x^2(pe^{tx} + 1 - p)} \end{aligned} \quad (5)$$

Note that the denominator of (5) is positive, given any $x, t > 0$. Therefore, to prove (5) is no smaller than 0, it suffices to prove that the nominator $ptxe^{tx} - (pe^{tx} + 1 - p)\ln(pe^{tx} + 1 - p) \equiv g(x,t)$ is no smaller than 0, given any $x, t > 0$. To prove that $g(x,t) \ge 0$ for any $x, t > 0$, note that $g(0,t) = 0$, and

$$\begin{aligned} \frac{d\,[g(x,t)]}{dx} =& pte^{tx} + pt^2xe^{tx} - pte^{tx}\ln(pe^{tx} + 1 - p) - pte^{tx} \\ =& pte^{tx}\big(tx - \ln(pe^{tx} + 1 - p)\big) \\ \ge& pte^{tx}\big(tx - \ln(pe^{tx} + (1-p)e^{tx})\big) = 0 \end{aligned}$$

Substitute (4) into (3), we have

$$\mathbb{P}(X > a) \le \min_{t>0} \frac{\big(f(b_n, t)\big)^{\frac{B}{b_n}}}{e^{ta}} \quad (6)$$

$$= \left(\frac{B(1-p)}{B-a}\right)^{\frac{B}{b_n}} \left(\frac{p(B-a)}{a(1-p)}\right)^{\frac{a}{b_n}} \quad (7)$$

We obtain equation (7) by solving for the unique root of $\frac{d}{dt}\left[\ln \frac{\big(f(b_n,t)\big)^{\frac{B}{b_n}}}{e^{ta}}\right] = 0$ and plugging it into (6).

□

4. SIMULATION STUDIES

4.1 Traces

We use 32 real-world Internet traffic traces, shown below in Table 1, in our simulations. These traces were collected by CAIDA [14] on two bidirectional 10 Gbps (in each direction) links, using two monitors located at Chicago (C) and San Jose (S) respectively. One bidirectional link, measured by the Chicago monitor, is from Seattle to Chicago (direction A) and the reverse direction (B); the other is from Los Angeles to San Jose (direction A) and the reverse direction (B). Each trace is 1-hour long collected at 13:00-14:00 UTC between 2013 and 2015. The name of each trace is coded as YYMMDDLR, where Y, M, D, L, R represent Year, Month, Day, Location (of the monitor), diRection respectively. For example, the first trace, named 130117SA was collected by the San Jose monitor in the direction A link on January 17th, 2013. The average traffic rate of each trace is 3.21 Gbps, corresponding to an average load of 32.1% on the 10Gbps link.

#	Traces	#	Traces	#	Traces	#	Traces
1	130117SA	9	130815SA	17	140320CA	25	141218CA
2	130117SB	10	131024CB	18	140320CB	26	141218CB
3	130221SB	11	131024SA	19	140320SA	27	150219CA
4	130425SA	12	131121CA	20	140619CA	28	150219CB
5	130425SB	13	131121CB	21	140619CB	29	150521CA
6	130529SB	14	131121SA	22	140619SA	30	150521CB
7	130620SA	15	131219CA	23	140918CA	31	150917CA
8	130620SB	16	131219CB	24	140918CB	32	150917CB

Table 1: The trace information.

4.2 Model-Based Simulation

In this section, we evaluate the efficacy of our scheme in maintaining and restoring load-balance, using a probability distribution measured from these traces on the traffic rate of a bundle. In these simulation runs, we assume that the router has $N = 256$ ITFs, each ITF splits its traffic into $E = 128$ flow bundles, and the overload threshold is $\beta = 0.95$.

We repeat the following two-step process 10,000 times and average the measurements, to arrive at each data point. First, we perform initial bundle-to-PE assignments via hashing (called "pure hashing" in the sequel): We generate E flow bundle rates in the i.i.d. fashion, according to this probability distribution, at each ITF and dispatch them uniformly randomly (*i.e.*, bundle to PE assignment via uniform hashing) to all active PEs. We measure the percentage of such assignments that lead to a PE overload event, and report the results in Figure 2 (Left). Second, if an overload event is triggered by such an assignment, the load-rebalancing algorithm is executed, possibly repeatedly, until the load-balance is restored or 100 attempts have been made. We measure the number of times the algorithm is executed in each overload event, and report the results in Figure 2 (Right).

As shown in Figure 2 (Left), even with initial load assignment via (pure) hashing, the probability that at least one PE is overloaded is quite low unless the target load α becomes very close to the overload threshold $\beta = 0.95$. For example, even when $\alpha = 0.9$, the overload probability is only about 10^{-2}. In other words, for about every 100 initial load assignments via hashing, 1 of them causes a PE overload event.

As shown in Figure 2 (Right), in the event of a PE being overloaded, the number of times the load-rebalancing algorithm needs to be executed to restore load-balance is close to 1 (*e.g.*, about 1.08 when $\alpha = 0.9$) unless the target load α becomes very close to the overload threshold $\beta = 0.95$. This demonstrates that our load-rebalancing algorithm is very effective in restoring load-balance.

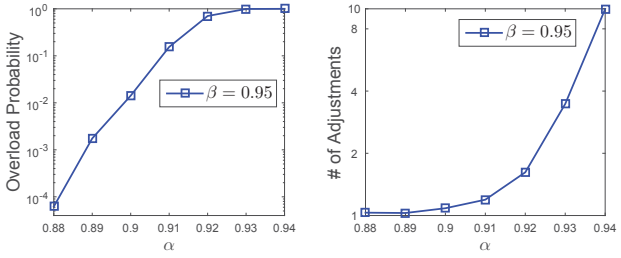

Figure 2: (Left) the overload probability (Y-axis is in the log scale); (Right) the average # of adjustments to eliminate the overload phenomenon at PEs (Y-axis is in the log scale)

4.3 Trace-Driven Simulation

4.3.1 Model for Router Power Consumption

In this section, we evaluate the efficacy of our scheme in maximizing the power savings of a GreenRouter (through minimizing the number of active PEs needed via near-perfect load-balancing). We assume the following power consumption parameters, based on measurements done in [15] and by ourselves. A conventional router consumes 1 unit of power (say per second). Among that, all line cards, the base, and the switching fabric consume 0.7, 0.27, and 0.03 units respectively. A GreenRouter has an extra switching fabric so it consumes 1.03 units if no PE is put to sleep. For a green router, ITFs and PEs account for 85% and 15% respectively of the total power consumption of line cards, which is 0.7 units as mentioned above. We assume that the power consumption of each component of a router/GreenRouter is a constant under different load conditions unless it becomes inactive (*i.e.*, put to sleep). This assumption is reasonable, since it was reported in [1] that the difference between idle and peak power consumptions of a component is very small (*e.g.*, less than 10%).

4.3.2 Simulation Results

Using the 1-hour long traffic traces shown in Table 1, we perform a 1-hour packet-level simulation of a 32-port × 10 Gbps per port GreenRouter. Hence we assume the router has $N = 32$ ITFs and 32 PEs. Note this parameter N is smaller than in the model-driven simulation earlier, because the packet-level simulation is much more computationally expensive. We assume $E = 128$, $\alpha = 0.9$, and $\beta = 0.95$, like in the earlier simulation. We assume that active PEs report their current loads to both the router controller and all ITFs very second. As explained earlier, we assume the router controller turns off an active PE if the average load drops below the $\alpha - 0.03$ threshold. However, the aggregate rate of even 16 traffic traces, which we will explain below is

the smallest workload, varies so little over the 1-hour period that this threshold is never reached in any simulation run.

As explained earlier, the average traffic rate of each trace is 3.21 Gbps. To simulate various load conditions on this router, we would have to feed a fractional number (*e.g.*, 1.25) of traffic traces to a 10 Gbps port. However, there is really no proper way of splitting or merging traffic traces that is needed to arrive at such a fractional number. Hence we feed exactly one trace to each port/ITF, but vary the number of ports/ITFs instead of fixing it at $N = 32$. More precisely, to impose a load of approximately 16.0% on the router, we set the number of ports/ITFs to 16 and feed traces #1-16 to them, one trace per port, respectively. To impose a load of approximately 32.1%, we add 16 ports/ITFs to the router, and feed traces #17-32 to them respectively. If we have more than 32 ports/ITFs to feed, we start to reuse traces from #1 (*i.e.*, feed trace #1 to port 33, #2 to port 34, and so on). Regardless of the number of ITFs/ports the router needs to have to impose a certain load, however, we assume their total power consumption remains 0.15 * 0.7 units, as explained above. Note also that while we vary the number of ITFs/ports, the number of PEs is fixed at $N = 32$ so their total packet processing capacity remains 320 Gbps.

As shown in Figure 3 (Left), a GreenRouter can deliver significant power savings when the average load is low, which is typically the case in backbone links [7]. For example, when the average load is 32.1%, the power consumption of a GreenRouter is only about 65% that of a conventional router. As the target load α of active PEs is set to 90%, a GreenRouter consumes slightly more power than an idealized (unrealistic though due to its abysmal QoS) GreenRouter (with target load α set to 100%).

Figure 3 (Right) shows that a GreenRouter performing the proposed load-balancing operation achieves much better QoS, in terms of at least an order of magnitude lower packet drop rates (by the active PEs) under all (average) load conditions, than one performing "pure hashing" (*i.e.*, assigning bundles to active PEs via hashing at the beginning and no load-balancing thereafter). In this simulation study, we assume each PE has a packet buffer that can hold up to 2K packets; packet drop occurs at a PE when its buffer overflows. Both curves mostly trend higher because the average load of an active PE generally increases and approaches the target load $\alpha = 0.9$ from below when the aggregate load to all ports/ITFs of the router increases, although they may trend down at certain points due the rounding-up effect (*e.g.*, 9 PEs are actually used when only 8.1 PEs are needed to bring the average load of a PE below 90%).

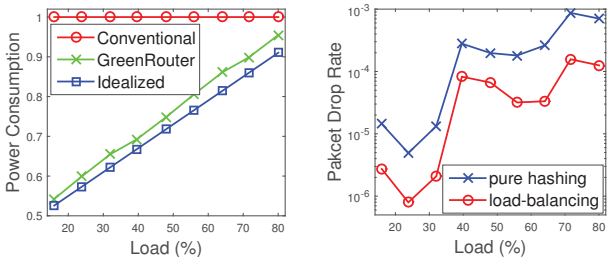

Figure 3: (Left) The power consumption of routers under different loads; (Right) The average packet drop rate

5. CONCLUSIONS

In this paper, we propose a randomized algorithm, that by near-perfectly balancing the loads across active PEs, allows a GreenRouter to deliver significant power savings, compared to a conventional router, by putting to sleep as many PEs as possible. Through both mathematical analyses and simulation studies, we show that our solution can restore near-perfect load balance after each PE overload event with high probability. With loads across PEs near-perfectly balanced, a GreenRouter also delivers acceptable QoS while aggressively pursuing power savings.

Acknowledgment: This work is supported in part by US NSF through awards CNS-1423182 and CNS-1248117. This work is also sponsored by NSFC (61373143, 61432009), 863 project (2013AA013502), Jiangsu Future Networks Innovation Institute: Prospective Research Projection Future Networks (No. BY2013095-1-03).

6. REFERENCES

[1] J. Chabarek, J. Sommers, P. Barford, C. Estan, D. Tsiang, and S. Wright. Power awareness in network design and routing. In *Proceedings of IEEE INFOCOM*, 2008.

[2] Y.-C. Chow and W. H. Kohler. Models for dynamic load balancing in a heterogeneous multiple processor system. *IEEE Transactions on Computers*, 100(5):354–361, 1979.

[3] J. De Jongh. *Share scheduling in distributed systems*. PhD thesis, TU Delft, Delft University of Technology, 2002.

[4] M. N. Dharmaweera, R. Parthiban, and Y. A. Sekercioglu. Toward a power-efficient backbone network: the state of research. *IEEE Communications Surveys & Tutorials*, 17(1):198–227, 2015.

[5] G. Epps, D. Tsiang, and T. Boures. System power challenges. In *Cisco Routing Research Seminar*, 2006.

[6] J. Fan, C. Hu, K. He, J. Jiang, and B. Liu. Reducing power of traffic manager in routers via dynamic on/off-chip scheduling. In *Proceedings of IEEE INFOCOM*, 2012.

[7] J. Guichard, F. Le Faucheur, and J.-P. Vasseur. *Definitive MPLS network designs*. Cisco Press, 2005.

[8] M. Gupta and S. Singh. Greening of the internet. In *Proceedings of ACM SIGCOMM*, 2003.

[9] Y. Kai, Y. Wang, and B. Liu. Greenrouter: Reducing power by innovating router's architecture. *Computer Architecture Letters*, 12(2):51–54, 2013.

[10] J. Kuang and L. Bhuyan. Optimizing throughput and latency under given power budget for network packet processing. In *Proceedings of IEEE INFOCOM*, 2010.

[11] B. Liu, J. Lu, Y. Kai, Y. Wang, and T. Pan. Power-proportional router: Architectural design and experimental evaluation. In *Proceedings of IEEE IWQoS*, 2014.

[12] S. Nedevschi, L. Popa, G. Iannaccone, S. Ratnasamy, and D. Wetherall. Reducing network energy consumption via sleeping and rate-adaptation. In *Proceedings of USENIX NSDI*, 2008.

[13] P. Raghavan. Probabilistic construction of deterministic algorithms: approximating packing integer programs. *Journal of Computer and System Sciences*, 37(2):130–143, 1988.

[14] C. Walsworth, E. Aben, kc claffy, and D. Andersen. The caida anonymized internet traces. `http://www.caida.org/data/passive/`.

[15] L. J. Wobker. *Power consumption in high-end routing systems*. `https://www.nanog.org/meetings/nanog54/presentations/Wednesday/Wobker.pdf`.

[16] M. Zhang, C. Yi, B. Liu, and B. Zhang. Greente: Power-aware traffic engineering. In *Proceedings of IEEE ICNP*, 2010.