Greening Router Line-Cards via Dynamic Management of Packet Memory

Vijay Sivaraman[†], Arun Vishwanath[§], Diet Ostry[⋆], Marina Thottan[*],
[†]University of New South Wales, [§]IBM Research-Australia, [⋆]CSIRO, [*]Alcatel-Lucent Bell Labs
Emails: {vijay@unsw.edu.au, arvishwa@au.ibm.com, diet.ostry@csiro.au, marinat@alcatel-lucent.com}

***Abstract*—Continued scaling of switching capacity in the Internet core is threatened by power considerations: Internet Service Providers face increased Carbon footprint and operational costs, while router manufacturers encounter upper limits on switching capacity per rack. This work studies the role of packet buffer memory on the power consumption of backbone routers. Our first contribution is to estimate from published data-sheets the energy costs of SRAM/DRAM packet-buffer memory, showing that it accounts for over 10% of power consumption in a typical router line-card; we then show using empirical data from core and enterprise networks that much of this memory is used for only a small fraction of time. Our second contribution is to develop a simple yet practical algorithm for putting much of the memory components to sleep and waking them as needed, while being able to control resulting traffic performance degradation in the form of packet loss during transient congestion. Lastly, we conduct a comprehensive evaluation of our scheme, via analytical models pertaining to long range dependent traffic, using simulations of off-line traffic traces taken from carrier/enterprise networks as well as on-line TCP flows in ns2, and by implementing our scheme on a programmable-router test-bed. Our study is the first to show the feasibility of, and energy savings from, dynamic management of packet buffer memory in core routers in the market today.**

I. Introduction

The power density of modern core routers is becoming a serious concern for Internet Service Providers (ISPs) – a single telecommunications rack today consumes tens of kW of power, and requires complex cooling systems to manage heat dissipation. In addition to the large carbon footprint, the high power consumption and cooling costs account for a significant fraction of the ISP's operational expenses. Though routing equipment is becoming more power efficient, the increase in efficiency is outpaced by annual increase in throughput capacity [1], meaning that the problem is likely to worsen with time.

The gravity of the problem has motivated major chip vendors, equipment manufacturers, service providers and academic researchers world-wide to collectively [2] find ways to manage and reduce the power consumption of telecommunications networks. The problem needs solutions at multiple levels, ranging from more efficient chips and components, to higher-level power management techniques that turn off (or underclock) components and sub-systems at certain times, or even redesign the Internet for power efficiency. While several such schemes have the potential to achieve considerable power savings, they involve significant architectural and/or protocol changes in the network. The cost and risk associated with such drastic changes increase the barrier to adoption by ISPs, thus stretching the time-horizon at which they become practical for wide-scale deployment. By contrast, in this paper we propose a power saving scheme that is admittedly more modest in its energy savings (around 10%), but requires minimal changes to existing router design, carries little risk of impacting network performance, is almost entirely transparent to network operators, and is ready for incremental deployment today.

Our specific focus is on adapting the packet buffer memories in core routers for improved energy efficiency. Today's backbone routers operate with Gigabytes of packet buffers per line-card to handle worst-case congestion scenarios. We present evidence (in §II-A) that such buffers account for nearly 10% of the power consumed by a typical router line-card. Further, we examine data collected over several years from nearly a hundred links in carrier and enterprise networks, and show (in §II-B) that high link-load (indicative of congestion) is a relatively rare occurrence, implying that it is wasteful in energy to keep the entire packet buffer memory always-on. We therefore propose (in §IV) that router buffer size be adapted dynamically to track buffer usage, allowing much of the off-chip buffer memory to be put to sleep when not needed, thus saving energy. Putting memory to sleep creates the risk of packet loss that could have been avoided with always-on buffers. Our scheme can be tuned to reduce this risk at the expense of reduced energy savings.

We validate our mechanism for dynamic buffer control by analysis, simulation and experimentation. In §V we develop a novel method to analytically estimate power savings and loss rates associated with dynamic buffer control under a fractional-Brownian long range dependent input traffic model. We apply our scheme to off-line traffic traces from operational networks in §VI-A, to on-line ns2 simulations of several thousand TCP flows in §VI-B, and implement it on an FPGA-based programmable router in §VI-C. Our evaluations show that dynamic buffer control has the potential to save most of the energy associated with off-chip buffering, and its impact on traffic performance can be made close to negligible. We hope that our study will persuade router manufacturers to incorporate dynamic buffer adaptation in core routers, and for network operators to trial them, as a relatively simple and safe way of reducing router power consumption.

The rest of this paper is organised as follows: §II motivates our study on packet buffers. Relevant background work is summarised in §III. In §IV, we present our dynamic buffer adjustment algorithm, while §V and §VI evaluate the algorithm via analysis, simulation and experimentation. The paper is concluded in §VII.

II. The Case for Reducing Router Buffer Energy

We now describe why it is worthwhile to reduce power consumption associated with packet buffers in core routers.

A. Energy Cost of Packet Buffers

Line-cards in modern core routers are usually equipped with two types of packet buffer memory: (i) a few Gigabytes of dynamic RAM (DRAM), which provides the bulk of the packet storage, and (ii) several Megabytes of static RAM (SRAM), which act as the packet cache. These off-chip memory components are typically arranged in hierarchical configurations [3]. Focusing first on DRAM, power calculators from Micron [4], a popular vendor of DRAM components, show that a state-of-the-art DDR2 SDRAM chip of 1 Gigabit capacity consumes about a Watt of power under moderate-stress conditions. When there are few read/write operations, corresponding to lower workloads, the power consumed is lower but still non-negligible. It should also be noted that router manufacturers often use specialised low-latency DRAMs such as Fast Cycle RAM (FCRAM) and Reduced Latency DRAM (RLDRAM), which consume about 40% more power than mass-market DDR2 or DDR3 SDRAM.

Next, the SRAM, which implements the packet cache, typically consumes more power than the bulk DRAM buffers: for e.g., a 4 Megabyte SRAM chip (with synchronous pipelined burst and with no bus latency NoBL) from Cypress [5] consumes around 4 Watts. More importantly, a large fraction of the power consumption of SRAM is due to the static component arising from leakage current, which is largely invariant to the load (i.e. frequency of read/write operations). Based on these power specifications, earlier works [6], [7], [8] have estimated packet buffers to account for between 5 and 10% of the power consumed by a router. As an example, Cisco's CRS-1 platform has reported that of the 375 Watts consumed by a line-card, memory accounts for 72 Watts (19%) [9], and around 10% of the total power of the system is attributable to buffer memory.

In addition to powering the packet buffer memory, it is also important to consider the power required to drive the memory controller circuitry, which implements the logic to move packets between main memory, cache memory, and on-chip memory. This is particularly challenging in high-speed routers which typically have long pipelines, as shown in [3]. Other complicating factors can include multiple output queues (e.g. for class-of-service support), and multiple memory channels/banks across which packets are spread. Memory controllers, which have the intelligence for managing and moving packets across these buffers, are integrated into custom ASICs on most routers, making it very difficult to estimate their energy footprint accurately. Nevertheless, we be obtain a reasonable estimate by noting that the DRAM memory controller on the AMD Opteron 6100-series multi-core processor [10]) accounts for 15-20% of the chip's power consumption. When used in a network processor such as EZchip NP-4 [11] (which operates at 50 Gbps and consumes 35 Watts), the DRAM controllers would conservatively account for 5-7 Watts. Since modern routers have complex memory pipelines across DRAM, SRAM, and on-chip buffers, and have separate ingress/egress queueing ASICs (as in CRS-1), it would be reasonable to expect that memory controllers consume at least half as much power as the memory chips themselves.

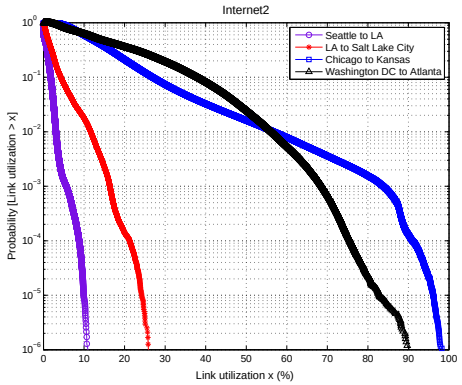


Fig. 1. CCDF of link load from the Internet2 backbone network.

Using the arguments presented above for DRAM and SRAM memory, the power consumed by packet buffers (i.e. memory chips and controllers) can be conservatively estimated to be around 10% of the total power of a line-card in modern core routers. This number is high enough to motivate this study, particularly because buffers are meant to absorb congestion, which is a relatively infrequent event in operational networks, as discussed next.

B. Link Congestion in Operational Networks

The aim of this section is to present empirical evidence that the buffer capacity available in routers to deal with worst-case congestion situations are needed only rarely. Towards this end, we obtained and analysed traces of link loads (at granularities of seconds, minutes, hours, and days) spanning several years, over nearly a hundred links from backbone and enterprise networks.

We focus on the Internet2 network [12] in this section, due to link load data it freely provides on its national long-distance network. Load from over fifty 10 Gbps links at 10-second granularity between the years Nov 2007 and Nov 2010 was analysed. Fig. 1 shows the complementary cumulative distribution function (CCDF) of the link utilisation, i.e. the probability that in a random 10-second interval, the load exceeds $x\%$ of the link capacity. The top two curves, corresponding to links from Washington DC to Atlanta and from Chicago to Kansas, were found to be amongst the most heavily loaded links in the Internet2 core. In spite of that, the chance that either of these links had load over 60% in any chosen 10-second interval was no more than one in a hundred. The other two curves, corresponding to Seattle to Los Angeles and Los Angeles to Salt Lake City, are more typical of most links on Internet2, with load never exceeding 30%. In fact the average load on many links was well below 20%.

The reader is referred to our earlier work in [13] for observations using another major Tier-1 carrier ISP and two enterprise networks. Our analysis corroborates that links in carrier networks today typically have low load for the most part, suggesting that large buffers in core routers are used rarely. Note that this does not preclude transient spikes in link loads during which times large buffers may well be put to use.

C. Buffer Occupancy - Analysis / Simulation

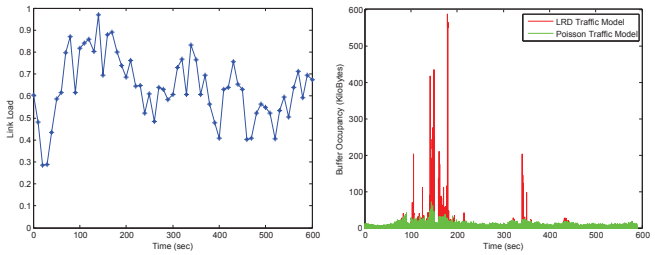
We now explicitly depict buffer usage, which we derive from analysis of the traffic load traces presented above, and ns2 simulations.

1) Trace Analysis: We generated synthetic traffic of matching load corresponding to the link load data obtained from operational networks. This is then fed into a custom FIFO queue simulation to generate the buffer occupancy trace. The traffic was generated using a simple Poisson model and a more sophisticated long-range dependent (LRD) model that uses an underlying fractional Gaussian noise (fGn) process with Hurst parameter $H = 0.85$ (the model is described in more detail in §V). To illustrate the outcome, we consider the Internet2 link from Chicago to Kansas, and show in Fig. 2(a) the load on that link over a 10-minute period of high load (reaching 96%) observed on 17 Sep, 2009. Fig. 2(b) shows the buffer occupancy derived by feeding the packet trace to a FIFO queue simulation (for computational tractability we scaled the link speed down from 10 to 1 Gbps). As the figure shows, buffer occupancy barely exceeds 40 KB under the Poisson model, while the buffer occupancy shows more burstiness with the LRD model, and is seen to spike occasionally to over 500 KB (which corresponds to about 4 ms of buffering at the link rate). The main point presented here is that although the traffic loads on links in operational networks can lead to large excursions in buffer occupancy, these occurrences are very rare, and it is wasteful in energy to have all buffers active at all times to deal with such rare events.

2) ns2 Simulation: To understand buffer occupancy in the presence of closed-loop TCP traffic, we conducted tens of simulations in ns2 using various topologies, link speed settings, number of flows and mixes of short-and long-lived flows. Due to space constraints, we present findings from one specific scenario that had TCP flows from 1000 users multiplexing at a 1 Gbps core link. A vast majority of the TCP flows were short-lived – file transfer sizes were Pareto distributed and think times between transfers followed an exponential distribution. The scenario is described in detail in §VI-B. Fig. 3 plots the queue occupancy for two link loads over a five second interval. Fig. 3(a) has 5 flows per user (5000 flows in total) creating a load of 91%, while Fig. 3(b) has 1 flow per user (total 1000 flows) creating a load of 41%. In both cases the buffer size was set to to the delay-bandwidth product of 31.25 MB. The heavy load scenario shows buffer occupancy to be high much of the time (between 5-25 MB), which presents reduced opportunity for putting buffers to sleep to save energy. The low load scenario on the other hand shows that buffer occupancy seldom exceeds a few tens of KB, presenting ample scope to save energy by putting off-chip buffers to sleep for much of the time.

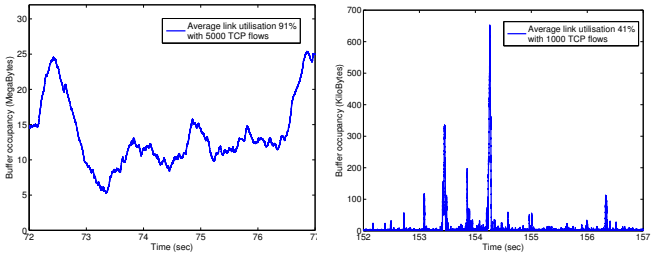
III. Related Work

The work in [14] suggests that choosing appropriate combinations of grooming at the optical WDM layer and switching at the IP layer can reduce overall network energy consumption, while [15] suggests choosing the right configuration of interfaces and chassis to achieve the desired switching capacity and minimise energy consumption. Other approaches recommend



(a) Link load (b) Buffer occupancy

Fig. 2. 10-min link load and buffer occupancy traces on an Internet2 link.



(a) 1 Gbps core link (load=91%) (b) 1 Gbps core link (load=41%)

Fig. 3. Buffer occupancy from ns2 of 1000 and 5000 TCP flows.

selectively turning off or underclocking network elements such as interfaces and line-cards [16] to save energy during periods of low load. We refer the reader to a survey paper [19] for a more comprehensive discussion of proposals for energy conservation in telecommunications networks. Although the above approaches are promising, they require major architectural and/or protocol changes to the network (e.g. delaying packets to aggregate them into bursts, and new routing protocols). These incur high costs and/or overhead for ISPs, making them less likely to be deployed in the immediate future. By contrast, our approach in this paper stands a much better chance of incrementally being deployed in the short term.

Our work is also partially inspired from recent debates regarding the size of buffers needed at core Internet routers. The work in [20] shows that buffers can be safely reduced by two to three orders of magnitude, and even reduced to as low as a few tens of packets [21]. While the debate about the right amount of buffering continues [22], reality remains that vendors continue to build routers with large buffers. That being the case, our approach, whereby router buffers are dynamically activated only when needed (thereby conserving energy), is likely to be more favourable to operators, since it eliminates the risk of adverse impact on traffic performance while still yielding a tangible benefit in terms of energy savings. Moreover, since we adjust buffer size at run-time, ISPs can gradually become comfortable with the idea of operating with reduced active buffers, making them more likely to adopt routers built with smaller buffers in the future.

IV. Dynamic Buffer Adjustment

In this section, we develop a simple and practical dynamic buffer adjustment algorithm.

A. Buffer Architecture

Different routers employ different packet buffer memory architectures. Thus, in this paper we consider a fairly generic three-level hierarchical model, obtained from [3], and depicted

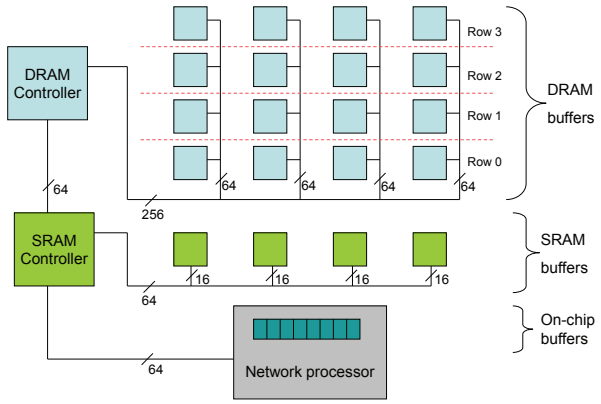


Fig. 4. Generic model of buffer architecture.

in Fig. 4. As shown in the figure, packet memory comprises on-chip (within the Network Processor (NP) or ASIC) buffers, off-chip cache (SRAM), and off-chip bulk memory (DRAM). On-chip buffer memory has capacity B_I, which is typically a few tens or hundreds of Kilobytes; for e.g. EZchip's 10-Gbps NP has 256 KB of on-chip packet memory while the Metro NP used in Cisco's CRS-1 router can hold 188 packets on-chip. The SRAM cache capacity is denoted by B_S, which is of the order of a few MB, while the bulk DRAM has capacity B_D, of the order of several GB. The buffer memory can support multiple FIFO queues (per interface and/or class-of-service), and head and tail blocks of packets for each queue are moved between memory hierarchy levels as needed in a pipelined fashion.

At 40 Gbps line rate a packet can arrive every 8 ns. To cater for this speed and latency requirements, a number of memory banks or chips are employed in parallel. Fig. 4 illustrates the architecture. It shows four SRAM chips, each with 16 data-pins, operating in parallel to present a 64-bit data bus to the network processor (via the controller) for increased throughput. As DRAMs are slower than SRAMs, by a factor of four, DRAM is accessed via a wider data bus – the figure shows DRAM organised as a 4×4 grid, with multiple chips within a row operating in parallel to increase data width, while each successive row adds to buffer depth. A packet stored in off-chip memory will therefore straddle all chips within one row of DRAM (or SRAM). In practice, each row (or each column, depending on the data-bus widths inside the router) of DRAM chips could be realised with a dual in-line memory module (DIMM). Note that the packet memory architecture presented here is generic and so the data-bus widths and number of memory chips in the figure suffice to illustrate the concept. These numbers would need to be adapted suitably when analysing a specific router. To be generic, we use the term N_R to denote the number of rows of DRAM chips in this memory organisation.

Custom ASICs on board a line-card typically house the memory controllers, and it is not uncommon to have several concurrent controllers. For ease of illustration we have shown a single aggregated controller for DRAM and SRAM, respectively. Our algorithm will put to sleep or activate an entire row of DRAM or SRAM memory chips, and operates as follows. As buffer occupancy falls, DRAM row-3 can be put to sleep, followed by row-2, and so on, till at some point the entire DRAM (and its controller) may be placed in the sleep state. Conversely, when buffer size needs to grow, row-0 is activated first (along with the controller), followed by row-1, and so on. We note that the FIFO nature of queues ensures that successive packets can be stored in successive memory locations – such compaction permits unused rows of memory chips to be put to sleep. The DRAM controller is put to sleep if and only if all DRAM memory chips are in the sleep state, and likewise for the SRAM. Based on data-sheets of SRAM and DRAM components, we estimate that it takes no more than 50μs to switch on (i.e. to bring from inactive to active state) the SRAM (controller and memory), and likewise about 500μs for the DRAM.

B. Energy Model

The power consumption of DRAM is tightly coupled with the frequency of read/write operations. To keep our energy model simple, we approximate the DRAM as being in one of three states: active (i.e. high frequency of read/write operations), idle (i.e. little or no read/write operations), and sleep (i.e. read/write disabled). Each row of DRAM (as shown in Fig. 4) of capacity 2 Gigabits consumes 2W when active, 200mW when idle, and 20mW when asleep – these numbers are derived from Micron [4] DDR2 specifications. As shown in the figure, larger DRAM buffer (1 Gigabyte in this case) is realised using multiple rows of DRAM chips (4 rows of 2 Gigabits each), and the power therefore scales linearly with buffer size. The DRAM controller, which controls the entire DRAM buffers, is assumed to consume half the power of the entire DRAM memory, namely 4W when active (i.e. when any row of DRAM is active), 400mW when idle (i.e. when no row of DRAM is active), and 40mW when asleep (i.e when all DRAM rows are in sleep state). We believe this model for DRAM power consumption is simple yet realistic, and can be customised to the architecture of the specific router whose buffer memory is being optimised for energy.

The power consumption of SRAM comprises two parts: a static component due to leakage current that increases with the number of transistors, and a dynamic component that is proportional to switching frequency (i.e. read/write operations). As the static power dominates [24], for simplicity our model assumes that the SRAM power is invariant to workload. We therefore consider SRAM to be in one of two states: active and sleep. For an SRAM of size 4 MB, we assume the active and sleep state powers to be 4W and 40mW, respectively. These are derived from Cypress [5] data-sheets. As before, the SRAM controller is assumed to require half the power of the SRAM (2W when active and 20mW when asleep).

In our evaluations further on in this paper, the baseline power (i.e. one which does not employ our energy management scheme for putting packet buffer elements to sleep) is estimated by assuming that SRAM is always active, and a row of DRAM is active or idle depending on whether the buffer occupancy spills over to that row or not. Our algorithm additionally puts both SRAM and DRAM rows into sleep state, and the resulting power savings are expressed as a percentage of the baseline. We only consider the power consumption of

off-chip buffer memory; the on-chip buffers internal to the network processor are assumed to be always-on, and their energy is therefore not explicitly modeled.

C. Algorithm for Dynamic Buffer Adjustment

In order to save maximum energy, active buffer capacity should track the actual queue occupancy, and so any off-chip buffer memory that is not needed can be put to sleep. However, the risk in following this approach is that if a sudden burst of traffic arrives, then packets from the burst may have to be dropped because there may not be sufficient time to activate the buffer memory. Thus, in order to control how aggressively or conservatively we want to track the buffer occupancy, we introduce a parameter $\alpha \in [0, 1)$ in our algorithm. The idea is to make the total active buffer capacity B at any time instant stay between the lower bound of the current queue occupancy Q and upper bound of the maximum available buffer space $B_I + B_S + B_D$. One simple way to do this is to use a linear combination of the two extremes, i.e. set $B = \alpha Q + (1 - \alpha)(B_I + B_S + B_D)$. Choosing $\alpha = 0$ (the extremely conservative setting) sets active buffers to maximum available buffers, essentially disabling the algorithm. Choosing $\alpha = 1$ (the aggressive setting), would make the active buffer capacity track the exact queue occupancy – this would be equivalent to saying that buffer space is created (by activating memory) as and when a packet arrives. Since memory takes non-zero time to become active, this would result in high loss. Choosing α in $[0, 1)$ allows the energy versus loss trade-off to be controlled. Our algorithm is presented formally below, taking into account that memory can only be activated/put to sleep in discrete quantities (i.e. capacity of the SRAM or DRAM row):

Algorithm 1 Determine active buffer size B (in bytes)

Inputs: Constants: α, B_I, B_S, B_D, N_R
Variable: current queue occupancy Q
Output: Buffer capacity B that should be active

1: **if** $Q < \alpha B_I$ **then**
2: $B = B_I$ /* on-chip buffers only */
3: **else if** $Q < \alpha(B_I + B_S)$ **then**
4: $B = B_I + B_S$ /* on-chip and SRAM buffers */
5: **else**
6: $B_A = (1 - \alpha)B_D + \alpha \max\{0, Q - B_I - B_S\}$
7: $K_D = \lceil \frac{B_A}{B_D/N_R} \rceil$ /* number of DRAM rows */
8: $B = K_D.B_D/N_R + B_I + B_S$
9: **end if**
10: output B

The algorithm above takes as input the user parameter α and the current queue occupancy Q (in bytes). If the queue occupancy is found to be low, i.e. on-chip buffer occupancy is below fraction α (step 1), all off-chip buffers are put to sleep (step 2). Otherwise, if occupancy of the on-chip and off-chip SRAM is below fraction α (step 4), only on-chip and SRAM buffers are kept on. If it is deemed that DRAM needs to be on (step 5), the desired DRAM capacity B_A is computed as a linear combination of the total DRAM capacity (weighted by $1 - \alpha$) and the current DRAM occupancy (weighted by α) in step 6. The number of rows of DRAM chips that need to be active to achieve this desired DRAM capacity is deduced in step 7, and the corresponding buffer size in bytes (including on-chip, SRAM and DRAM buffers) is determined in step 8 and returned in step 10.

D. Discussion

Our algorithm is relatively easy to implement in hardware. It is executed upon a packet arrival or departure event, i.e. whenever the queue occupancy Q changes. Note that if $1 - \alpha$ is chosen to be a negative power of 2 (e.g. $\alpha = 0.75$ or 0.875), then all steps can be performed without any multiplication or division operations, since the product in steps 1 and 3 can be precomputed for given α, and steps 6-8 can be realised using shift and add operations. An additional memory row (i.e. SRAM or a row of DRAM) is activated whenever the algorithm returns an active buffer size B that is higher than what is currently active, and vice versa. However, to prevent memory components toggling between active and sleep states in quick succession, we introduce some hysteresis protection; specifically, our implementation (described in §VI) mandates that any memory component, once active, should not be put to sleep for at least 1ms.

We have intentionally chosen to keep our algorithm simple, and have strived to have only one control parameter α. It is easy to envisage any number of more complex algorithms for determining the best buffer size, such as by attempting to predict how queue occupancy will evolve, but we believe they will be too complex for real-time hardware implementation. There are some unavoidable risks in turning buffer memory elements on/asleep to save energy. In an exceedingly improbable case, on-chip buffers of size $B_I = 100$ KB can go from zero to full occupancy within 1.25μs at input rate of 640 Gbps (if each of the 16 CRS-1 line-cards sends traffic at 40 Gbps to the same egress line-card), which is much faster than the SRAM turn-on time of 50μs. Likewise, SRAM of capacity $B_S = 4$ MB can fill within 50μs at 640 Gbps, an order of magnitude quicker than the DRAM turn-on time of 500μs. However, such scenarios are worst-case, and were never observed in the traces, simulations, and experiments we describe in §VI. To protect against typical bursts of packets that need to be absorbed while buffer memory is being activated, we found that using $\alpha \in [0.8, 0.9]$ ensured sufficient vacant buffer space for such transients, while still saving significant energy. The router manufacturer may set α at a default value in this range, and network operators can tune it if they prefer a different trade-off point between the benefit (of energy savings) and risk (impact on traffic performance). The next two sections evaluate our scheme via analysis, simulation, and experimentation.

V. Analytical Model

We develop an analytical model to predict loss and power savings from our dynamic buffer adjustment scheme. Our model is developed for the case of on-chip memory overflowing into SRAM (since that is when the losses are maximum) - the same approach can be directly applied to model overflows from SRAM to DRAM or from one row DRAM to another, but is not discussed here due to lack of space.

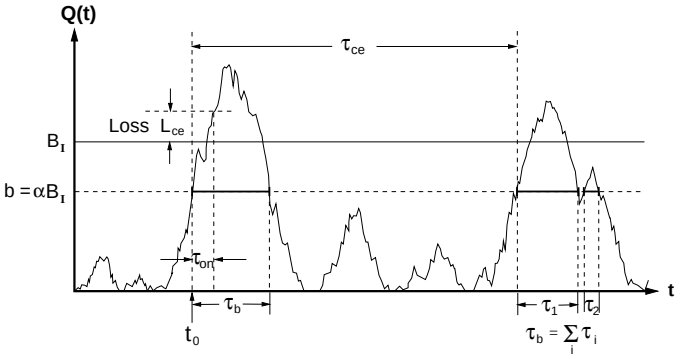


Fig. 5. Entities required by the analytical model to estimate loss.

Our analysis uses a fluid model of traffic, consistent with the Norros model for LRD traffic [25] (packetized traffic will be considered via simulation in the next section). It is well-known that Internet traffic exhibits LRD properties. Unlike traditional queueing models, which are useful for analysing Markovian traffic, fluid models have been shown to be tractable when analysing systems with LRD traffic. In our fluid model, time is discretized into small slots (of $\delta_t = 1\mu$s in our case), and the volume of traffic arriving in the i-th slot is given by:

$$A_{\delta_t}(i) = m + \sigma x_i \tag{1}$$

where m is a constant traffic rate (e.g. 800 bits/μs), x_i is the i-th sample of a standard (zero mean and unit variance) fractional Gaussian noise (fGn) random variable (with chosen Hurst parameter $H = 0.85$ in our study), and σ is used to scale the standard-deviation of the fGn (we use $\sigma = m/2$ to ensure that no more than 2.2% of samples are truncated when the quantity in the right side of (1) becoming negative). We use our filtering method of [26] to generate very long sequences (512 million samples) of high-precision normalised fGn values x_i that will be used for simulations to validate our model.

We define the following entities required by our analysis, and depicted in Fig. 5: A *busy period* is defined as a contiguous interval of time over which the queue is non-empty. We define a *congestion event* as the first instant t_0 within a busy period that the buffer occupancy reaches the threshold αB_I; at this time, off-chip SRAM turn-on is initiated, and the SRAM becomes available for use $\tau_{on} = 50\mu$s subsequently. A busy period which has a congestion event is termed a *congested busy period*. As shown in the figure, not all busy periods are congested, and losses happen within a congested busy period if and only if the on-chip buffer overflows during the $\tau_{on} = 50\mu$s during which the SRAM is being turned on; further, the SRAM is turned off at the end of each congested busy period.

Our approach to estimating loss probability proceeds as follows: we first consider an arbitrary congested busy-period, and estimate $\overline{L_{ce}}$, the mean volume of lost traffic (in bits) during the congestion event. We then compute the expected time $\overline{\tau_{ce}}$ (in μs) between congestion events. Dividing $\overline{L_{ce}}$ by $\overline{\tau_{ce}}$ gives us the mean loss rate in bits/μs, which is then divided by the mean traffic arrival rate m to obtain bit loss probability:

$$L = \overline{L_{ce}}/m\overline{\tau_{ce}} \tag{2}$$

A. Estimating loss per congestion event ($\overline{L_{ce}}$)

Let t_0 denote the time instant within the congested busy period at which the buffer occupancy reaches the threshold αB_I; this triggers the SRAM to be turned on, which is ready for use at $t_0 + \tau_{on}$. Loss can happen only if the arriving traffic in $[t_0, t_0+\tau_{on}]$ exceeds the sum of the available on-chip buffer $(1-\alpha)B_I$ and the service $c\tau_{on}$ that can be provided in that interval, where c is the service rate (1 Gbps or 1000 bits/μs in our example). Thus the volume of lost traffic is

$$\begin{aligned} L_{ce} &= [A[t_0, t_0+\tau_{on}] - c\tau_{on} - (1-\alpha)B_I]^+ \\ &= [\sum_{i=1}^{\tau_{on}} A_{\delta_t}(i) - c\tau_{on} - (1-\alpha)B_I]^+ \\ &= [\sigma X^{(\tau_{on})} - (c-m)\tau_{on} - (1-\alpha)B_I]^+ \end{aligned} \tag{3}$$

where the arriving traffic follows the Norros model in (1), and we use $X^{(\tau_{on})} = \sum_{i=1}^{\tau_{on}} x_i$ to denote the aggregation of fGn samples over τ_{on} slots. It is known that the aggregation of m standard fGn samples having Hurst parameter H is also Gaussian with variance m^H [27, pg 19], so we have $X^{(\tau_{on})} \sim \mathcal{N}(0, \tau_{on}^H)$.

Noting that L_{ce} is positive only when $X^{(\tau_{on})} \geq X_{min}^{(\tau_{on})} = ((c-m)\tau_{on} + (1-\alpha)B_I)/\sigma$, we can now compute the expectation of the volume of traffic lost during the congestion event by integrating over the distribution of $X^{(\tau_{on})}$:

$$\begin{aligned} \overline{L_{ce}} &= \int_{x=X_{min}^{(\tau_{on})}}^{\infty} L_{ce} \frac{1}{\tau_{on}^H\sqrt{2\pi}} e^{-x^2/2\tau_{on}^{2H}}\, dx \\ &= \frac{\sigma}{\tau_{on}^H\sqrt{2\pi}} \int_{x=X_{min}^{(\tau_{on})}}^{\infty} x e^{-x^2/2\tau_{on}^{2H}}\, dx \\ &\quad - \frac{(c-m)\tau_{on} + (1-\alpha)B_I}{\tau_{on}^H\sqrt{2\pi}} \int_{x=X_{min}^{(\tau_{on})}}^{\infty} e^{-x^2/2\tau_{on}^{2H}}\, dx \\ &= \frac{\sigma\tau_{on}^H}{\sqrt{2\pi}} e^{-((c-m)\tau_{on}+(1-\alpha)B_I)^2/2\sigma^2\tau_{on}^{2H}} \\ &\quad - \frac{(c-m)\tau_{on} + (1-\alpha)B_I}{\tau_{on}^H} \Phi\left(\frac{(c-m)\tau_{on} + (1-\alpha)B_I}{\sigma\tau_{on}^H}\right) \end{aligned}$$

where $\Phi(x) = P(X > x)$ is the tail probability of the standard Gaussian variable X. This expression allows us to estimate $\overline{L_{ce}}$ in closed form.

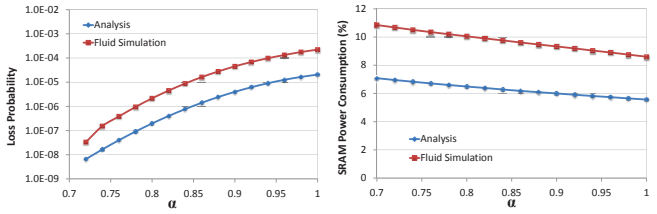
B. Estimating time between congestion events ($\overline{\tau_{ce}}$)

To estimate $\overline{\tau_{ce}}$, we deduce $\overline{\tau_b}$, the mean length of time that the queue exceeds a level b during a congestion event, and $P[Q(t) \geq b]$, the probability that the queue exceeds b, where $b = \alpha B_I$. In an arbitrary time interval T, the queue exceeds level b for a total time $T \times P[Q(t) \geq b]$ and so the mean number of congestion events with level b in T is $P[Q(t) \geq b]T/\overline{\tau_b}$. The average time between congestion events is then $\overline{\tau_{ce}} = \overline{\tau_b}/P[Q(t) \geq b]$ giving an average loss rate of $L_{ce}/\overline{\tau_{ce}}$.

In the case of Norros (fractional Brownian) traffic, no exact closed-form solutions are known for the required quantities $\overline{\tau_b}$ and $P(Q \geq b)$. However an approximation for $\overline{\tau_b}$ is developed in [28], based on the distribution of the maximum queue length for fractional Brownian traffic and a Poisson clumping approximation, as

$$\overline{\tau_b} \approx \int_0^{\infty} \bar{\Phi}\left(\frac{ct - \mu_{r_b,d_b}(t)}{\sqrt{\sigma_{r_b}^2(t)}}\right) dt \tag{4}$$

where $\Phi(x)$ is the tail probability of the standard normal distribution, c is the scaled surplus service rate and $\mu_{r_b,d_b}(t)$



(a) Loss (b) SRAM power usage

Fig. 6. Comparing analytical estimates of (a) loss probability and (b) SRAM power consumption against simulation.

and $\sigma^2_{r_b}(t)$ are the mean and variance of fBm traffic with Hurst parameter H and conditioned on the queue increasing from empty to a level b in the time interval $r_b = bH/(c(1-H))$ (the critical time scale of fBm traffic).

An approximation for the queue occupancy $P(Q \geq b)$ for fractional Brownian traffic with mean net input rate m and standard-deviation over a unit interval σ is given in [29] as

$$P(Q \geq b) \approx \int_b^\infty cx^{\frac{1-2H}{H}} e^{-\frac{x^{2-2H}(1-H)^{2H-2}|m|^{2H}}{2H^{2H}\sigma^2}}, \quad (5)$$

where

$$c = \nu\alpha^{\beta/\nu}/\Gamma(\beta/\nu),$$

$$\alpha = \frac{(1-H)^{2H-2}|\mu|^{2H}}{2H^{2H}\sigma^2}$$

and $\beta = (1-H)/H$, $\nu = 2-2H$, and Γ is the gamma function. The integral can be carried out in closed form in terms of the incomplete gamma function.

C. Estimating overall loss and power savings

Using the closed-form expressions for $\overline{L_{\text{ce}}}$ and $\overline{\tau_{ce}}$ above, we compute the loss probability L given in (2). Simultaneously, we also compute the SRAM power usage by making the approximation that it is on whenever the on-chip buffer occupancy is above the threshold, namely $P(Q > \alpha B_I)$, estimated from (5). To validate our model we wrote a slotted simulation (with slot size 1μs) in which fluid LRD traffic arrives as per the Norros model described in (1), for which we generate long fGn sample paths x_i (512 million samples) using our filtering method in [26]. This fluid traffic (in bits) is served by a link operating at 1 Gbps, with queueing occurring in the on-chip buffer of size $B_I = 16$ KB and (practically) unlimited off-chip SRAM. The SRAM is triggered to turn on whenever the on-chip buffer occupancy reaches αB_I for fixed parameter $\alpha \in [0,1]$, and is subsequently ready for use $\tau_{on} = 50\mu$s later.

Fig. 6(a) shows the bit-loss probability associated with waking up the SRAM, when load is 80% and α varies from 0.7 to 1 (lower values of α lead to negligible loss that cannot be measured accurately in simulation). As expected loss increases steadily with α, since a higher threshold leaves smaller margin of on-chip buffer capacity to absorb bursts while the SRAM is turning on. The analysis under-estimates the loss by about an order of magnitude; this is not surprising, since expressions (4) and (5) are asymptotic approximations that do not capture multiplicative factors. It should however be noted that the slopes of the analysis and simulation curves are very well matched, validating the analysis as a good approximation. Further, Fig. 6(b) shows the percentage of time

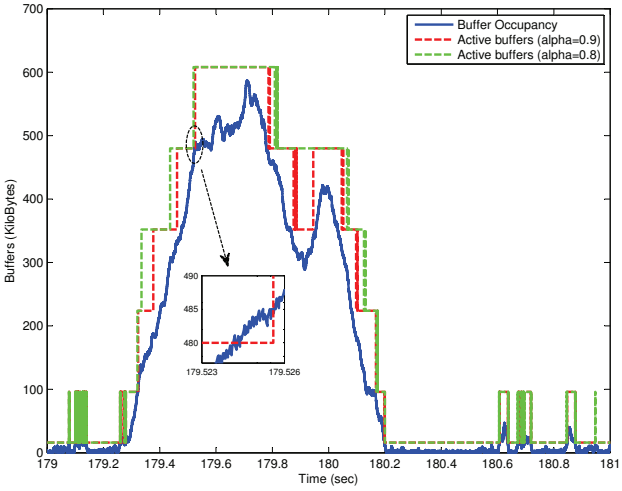


Fig. 7. Buffer occupancy and active buffers trace for $\alpha = 0.8$ and 0.9.

that the SRAM is on, which is seen to fall steadily with α. The analysis under-estimates SRAM usage, since it only counts periods when the occupancy is above the threshold, whereas the simulation keeps the SRAM on till the end of the busy period; nevertheless, the two curves match well in slope, again validating our analysis. Our subsequent evaluation will conduct simulations and experimentation with packetized input traffic and closed-loop TCP traffic for real-world validation.

VI. Evaluation

We now evaluate the efficacy of our algorithm using traces from real Internet data, on-line simulations using TCP and a real-time implementation on a testbed of NetFPGA routers.

A. Off-Line Trace Analysis

We used time-varying traffic load traces obtained from carrier and enterprise networks (as discussed in §II-B) and generated synthetic Poisson and long range dependent (LRD) traffic, which was in turn fed into our simulation of our algorithm. The performance metrics such as power savings and packet loss ratios were recorded.

1) Traffic Generation: We evaluated our algorithm using both Poisson and LRD traffic. The mean rate of the generated traffic for each link was varied as per the link load trace for that link. Consider the Internet2 link from Chicago to Kansas, shown in Fig. 2(a). The load on that link is measured every 10 sec, and so we varied the mean rate of the generated traffic over each 10-sec interval to match the measured load. As shown in Fig. 2(b), the Poisson traffic did not exhibit sufficient burstiness to cause high queue occupancy. The rest of this section therefore presents results from the LRD traffic model. The burstiness generated by this traffic model reflects closely the burstiness of Internet traffic. Our LRD traffic generator, described in Section V, accumulates the resulting fluid volume into packets of variable size with distribution derived from CAIDA's measurements over 87 million packets at the NASA Ames Internet Exchange (AIX) [30]. We note that to generate traffic traces for sufficiently long periods (> 10 min), we had to scale the 10 Gbps links down to 1 Gbps.

2) Dynamic Buffer Adaptation: We fed the packet trace discussed above into our algorithm for dynamic management of packet buffer memory. As mentioned previously, the loads

measured on the links were very low for the most part. Thus, to illustrate our algorithm we choose the load on the Chicago to Kansas Internet2 link corresponding to a 10-min window observed on 17 Sep, 2009. As depicted in Fig. 2, despite this 10-min interval having relatively high load, the queue occupancy did not exceed 600 KB. Clearly, these packets can be easily accommodated in on-chip and off-chip SRAM memory, and there would not be any need for storing packets in off-chip DRAM buffers. Nevertheless, to illustrate the operation of our algorithm, we assume that the router has capacity to buffer 16 KB on-chip, 80 KB in off-chip SRAM, and 512 KB in off-chip DRAM (organised as shown in Fig. 4).

The buffer occupancy trace and the buffer size determined by our algorithm over a chosen 2-sec interval are shown in Fig. 7. It can be seen from the figure that the algorithm initially places all off-chip buffers in the sleep state, and sets the active buffer size to 16 KB, corresponding to the on-chip buffer size. Then, as the buffer occupancy increases (at around 179.2 sec), the algorithm first activates SRAM, and subsequently each row of the DRAM. Finally, the off-chip memory is again placed in the sleep state by the algorithm when the buffer occupancy falls beyond 180.2 sec. The impact of parameter α on how aggressively the algorithm tries to save energy can also be seen: there are several instances where the $\alpha = 0.9$ curve is seen to track the actual occupancy curve more closely than the $\alpha = 0.8$ curve. Two things to note here are: there are instants where the buffer occupancy overshoots the buffer size (e.g. at around 179.5 sec, see inset) since it takes time to activate each memory element, and this causes loss that could have been avoided if buffers were always-on. Second, while we have chosen a narrow period of particularly high load, in general loads are quite low and much of the off-chip memory can safely be put to sleep, saving most of the off-chip buffering energy. This trade-off between energy-savings and loss is quantified next.

3) Power vs. Loss Trade-Off: To evaluate the power vs. loss trade-off performance of our algorithm, we applied it off-line to the traffic trace considered above, namely the Internet2's Chicago to Kansas link, and studied the impact of different values of the parameter α. When $\alpha = 0$, the algorithm is effectively turned off. As α was increased and until it approached 0.7, the power savings were not very significant (typically $< 50\%$ because the SRAM was active for over 80% of the time), and alongside there were no packet losses induced by the algorithm. This is because at low values of α the algorithm is very conservative, and activates off-chip memory well in advance of the on-chip buffers overflowing. The SRAM state transition frequency (i.e. from active to sleep and vice-versa) varied between 5 and 10 times/sec, while it was < 1 for the DRAM rows. The performance of the algorithm when α is in the range $(0.7, 1)$ is depicted in Fig. 8. When $\alpha = 0.8$, the algorithm activated SRAM for only 2.66% of the time, which is substantially lower than when $\alpha = 0.7$. The four DRAM rows were activated for only 0.83%, 0.61%, 0.20% and 0.05% of the time. The SRAM toggled between states 62.4 times/sec, while the DRAM rows toggled states between 0.1-2.7 times/sec. Note that the baseline power consumption was 6.66W, and the average power consumed by running

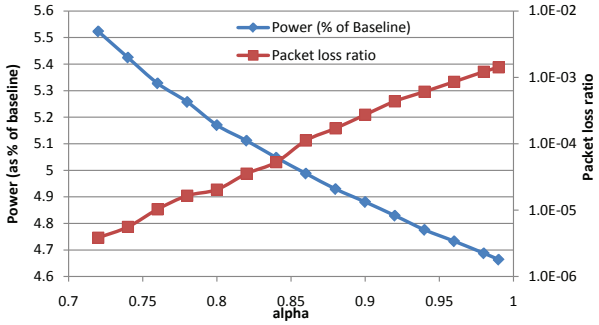


Fig. 8. Power consumption (% of baseline) and loss trade-off for various α.

the algorithm was 0.34W, which is only about 5.17% of the baseline power. However, this comes at the cost of increased packet loss due to sudden bursts arriving when the buffer memory is in the sleep state: for $\alpha = 0.8$, there is loss for about 2×10^{-5} packets (which is within the tolerance of 10^{-3} typical of many Service Level Agreements (SLAs), e.g. [31]). The figure shows that as α increases to 1, power consumption (left axis) falls, while loss (right axis) increases. The operating point on this trade-off curve can be chosen by the operator, and will depend on the memory configuration and size, as well as traffic characteristics, cost of power, and criticality of traffic.

B. On-Line ns2 Simulations with TCP

In this section, we evaluate the performance of the algorithm in the presence of TCP traffic, which is known to carry over 90% of Internet traffic. We implemented the algorithm in the ns2 simulator and carried out numerous simulations comprising different topologies, link speeds, number of flows, and mixes of long-/short-lived TCP flows. In what follows, we present results from a small subset of scenarios to illustrate the efficacy of dynamic buffer adaptation with a realistic traffic mix under different loading conditions.

We consider a three-level hierarchical topology comprising one core bottleneck link fed by 50 edge links, with each edge link aggregating traffic from 20 access links. In all there were 1000 source hosts generating TCP traffic. The capacity of the core and edge links was set to 1 Gbps, while the access links had capacity uniformly distributed in $[100, 300]$ Mbps. The mean round-trip time (RTT) for the flows was 250 ms. We used the TCP Reno version and varied the number of flows from 1000 to 5000 by varying the number of flows per user, to simulate different loading conditions. Our simulations comprised both short- and long-lived flows. The former models HTTP transfers with Pareto distributed file-sizes (mean 100 KB and shape parameter 1.2) and exponentially distributed think-times of mean 1 sec, while the latter represents persistent FTP transfers. The number of long-lived flows (50) accounted for only a small fraction of the total number of flows. These parameter settings are consistent with prior literature and based on measurement studies of Internet traffic. The maximum window size was set to a very large value so that transfers are never limited by the window size. Our simulations ran for over 180 sec, and all links were equipped with delay-bandwidth buffers. The simulation settings (link speeds, number of flows) are at the limit of the memory and CPU constraints available on our ns2 environment.

TABLE I
POWER SAVINGS, AVERAGE FLOW COMPLETION TIMES (AFCT) AND PACKET LOSS FROM NS2 SIMULATIONS

Workload	Load	AFCT	Power saved	Packet loss
Low	21.5%	2.233 sec	97.4%	0
Medium	41.1%	2.244 sec	97.2%	0
High	59.8%	2.250 sec	83.4%	10^{-7}
Heavy	78.6%	2.295 sec	52.9%	10^{-6}
Very heavy	90.9%	2.757 sec	11.6%	10^{-6}

We ran the simulations with our algorithm implemented at the core link, and evaluated the performance for various values of the parameter α in $[0.75, 0.95]$. The results are summarised in Table I. When the workloads were low to medium (up to 41%), the off-chip SRAM and DRAM buffers were used for only $\approx 0.25\%$ of the time, thus saving over 97% of the off-chip buffering energy. There were no packet losses as a result of the buffers turning active/asleep, and so the average flow completion time (AFCT) for HTTP flows was identical to the case when all buffers were always-on. In addition, all values of α in $[0.75, 0.95]$ gave identical results.

Next, under high load (i.e. 59.8%), the off-chip buffers were used about 12.3% of the time, and our algorithm saved in excess of 83% of energy. A very small fraction of packets were lost (of the order of 10^{-7}), which barely increased AFCT by a few ms. Even under heavy workload regime (78.6% load), we found that our algorithm could save over 50% of the off-chip buffering energy as the SRAM/DRAM buffers were used for only 40% of the time. Increase in packet loss (of 10^{-6}) and AFCT (< 4 ms) were also negligible.

Finally, when the load was very heavy ($> 90\%$), it is not surprising to see that off-chip buffers were used nearly 82% of the time, resulting in about 11% power savings. The loss induced our algorithm even under this setting was very small loss (i.e. $< 10^{-6}$, which is again within the tolerance of typical SLAs [31]), and AFCT was barely affected (by no more than 6 ms compared to low load). These results show that our algorithm performs well across a wide range of workloads with negligible impact on TCP traffic performance.

C. Real-Time Implementation in a Router

We use the programmable NetFPGA platform, hardware-based traffic generators and delay emulators to demonstrate the feasibility of deploying our scheme in hardware.

1) Implementation and Set-Up: Using Verilog, we implemented the dynamic buffer adaptation algorithm in the hardware data-plane, thus extending the gateware available at the NetFPGA website for router buffer sizing studies [32]. Since the gateware provides 512 KB of output queue capacity (internally implemented on off-chip SRAM), we evaluated our algorithm by partitioning this buffer capacity (virtually) into 16 KB of on-chip, 48 KB of SRAM, and 448 KB of DRAM buffers respectively, organised as four rows (as shown in Fig. 4). The algorithm is executed at every packet arrival or departure instant to determine the buffer size that should be active. The queue size register *oq_queue_full_thresh*, whose value determines the capacity of the output queue, is then updated by the algorithm which takes effect after a few clock cycles. Our implementation does not explicitly put memory elements to sleep nor does it introduce delays to model memory state transition latencies. We use the energy model developed earlier to estimate the energy savings. The objective of our experiments is to demonstrate the feasibility of implementing our algorithm in hardware. We tried various settings of α, and the results described are for $\alpha = 0.8$, which was found to yield a good balance between power and loss performance. We extended the software on the NetFPGA to extract the queue size information and also log all queuing/dequeuing events, so we can plot and analyse them. For our tests the focus was on a single output link at the NetFPGA router. Since the NetFPGA has only four ports, to emulate a large fan-in we rate-limited the output port; this also lets us make that port a bottleneck link for some tests.

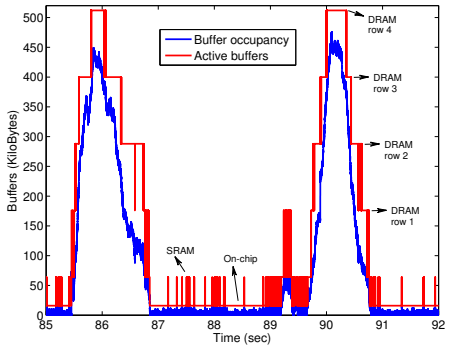


Fig. 9. Buffer adjustment with 150 TCP flows.

2) Power Savings with TCP Flows: We generated 150 concurrent TCP flows using the Iperf traffic generator. These flows share a 123 Mbps link under observation. The duration of the experiment was 180 sec. We used a hardware-based delay emulator from Anue Systems [34] to set the RTT to 35 ms so that the 512 KB of available buffers corresponds to the delay-bandwidth product. To emulate network conditions where this link may or may not be the bottleneck at all times, we introduced on-off UDP traffic in another downstream link so that the link under observation toggled between being and not being a bottleneck every few sec.

With our algorithm running on the NetFPGA, the output queue occupancy trace and the dynamically adjusted buffer size over a 7-sec interval for a chosen run are shown in Fig. 9. The link is a bottleneck at around 86 and 90.1 sec, when the queue occupancy rises above 400 KB (all 4 DRAM rows are active), while the link is clearly not a bottleneck between 87-89 sec, when the queue occupancy is just a few KB (only SRAM gets activated).

The outcome of the experiment are as follows: The algorithm used SRAM buffers 44.3% of the time, while DRAM rows 1 to 4 were used 38.7%, 27.6%, 17.5% and 7.6%, respectively. This corresponds to a saving of nearly 40% of the off-chip buffering energy. The reader may note that the 40% energy savings for this scenario (88.7% load) are larger than the 11% savings for the similar simulation scenario (90.9% load, see Table I) of the previous section – this is explained by the fact that in the current scenario the link toggles periodically between being and not being a bottleneck link (unlike the simulation scenario in which the link stays a bottleneck), and this fluctuation in load (for the same mean) presents increased opportunity for energy savings.

VII. Conclusions

This paper posits that much of the off-chip packet buffer energy, which by our estimates is around 10% of the total line-card energy, in backbone routers can be saved by selectively putting to sleep memory components when not needed. Although the savings are limited, the reduction comes with minimal hardware/software changes needed to effect the algorithm. We have strived for achieving this outcome by providing only a single control parameter, i.e. α. The scheme can be easily deployed incrementally today without requiring any new network protocols or architectures since the changes that need to be made for the algorithm to function are contained within a router. Our detailed evaluations using real traffic traces and TCP simulations show that the risk of affecting traffic performance is very minimal, since losses will only happen during transients when the memory components are transitioning from sleep to active state, and these can be mitigated to some extent by adjusting the threshold parameter α in the algorithm. We hope our work can persuade router manufacturers and operators to consider dynamic buffer size adjustment as a relatively safe and easy way of reducing the energy consumption of router line-cards.

References

[1] R. Tucker et al. Energy Consumption of IP Networks. In *Proc. Europ. Conf. on Optical Comm.*, Belgium, Sep 2008.

[2] GreenTouch. www.greentouch.org.

[3] S. Iyer, R. R. Kompella, and N. McKeown. Designing Packet Buffer for Router Linecards. *IEEE/ACM Trans. Netw.*, 16(3):705–717, Jun 2008.

[4] Micron-Technology-Inc. System power Calculators. www.micron.com/support/dram/power_calc.html.

[5] Cypress-Semiconductor. QDR-II SRAM CY7C1515KV18. http://www.cypress.com/?docID=24145.

[6] R. Tucker et al. Evolution of WDM Optical IP Networks: A Cost and Energy Perspective. *J. Lightw. Tech.*, 27(3):243–252, Feb 2009.

[7] S. Aleksic. Analysis of Power Consumption in Future High-Capacity Network Nodes. *J. Opt. Comm. and Netw.*, 1(3):245–258, Aug 2009.

[8] O Tamm. Scaling and Energy Efficiency in Next Generation Core Networks and Switches. In *ECOC Workshop.*, Austria, Sep 2009.

[9] G. Epps et al. System Power Challenges. Cisco Research Seminar, Aug 2006. http://www.cisco.com/web/about/ac50/ac207/proceedings/POWER_GEPPS_rev3.ppt.

[10] AMD Opteron processors. http://www.amd.com/acp.

[11] EZchip 50 Gbps NP-4 Network Processor. http://www.ezchip.com/Images/pdf/NP-4_Short_Brief_online.pdf

[12] Internet2 Network. http://www.internet2.edu/network/.

[13] A. Vishwanath et al. Adapting Router Buffers for Energy Efficiency. In *Proc. ACM SIGCOMM CoNEXT*, Japan, Dec 2011.

[14] G. Shen and R. Tucker. Energy-Minimized Design for IP over WDM Networks. *J. Opt. Comm. and Netw.*, 1(1):176–186, Jun 2009.

[15] J. Chabarek et al. Power Awareness in Network Design and Routing. In *Proc. IEEE INFOCOM*, USA, Mar 2008.

[16] S. Nedevschi et al. Reducing Network Energy Consumption via Rate-Adaptation and Sleeping. In *Proc. USENIX NSDI*, USA, Apr 2008.

[17] Bill St.Arnaud. CANARIE: Research Networks to Help Reduce Global Warming. In *OFC Workshop on Energy Footprint of ICT: Forecast and Netw. Sol.*, USA, Mar 2009.

[18] A. Cianfrani et al. An Energy Saving Routing Algorithm for a Green OSPF Protocol. In *IEEE INFOCOM Traffic Monitoring and Management Workshop*, USA, Mar 2010.

[19] Y. Zhang et al., Energy Efficiency in Telecom Optical Networks. *IEEE Comm. Surveys and Tutorials*, 12(4):441–458, Q4 2010.

[20] G. Appenzeller, I. Keslassy, and N. McKeown. Sizing Router Buffers. In *Proc. ACM SIGCOMM*, USA, Aug-Sep 2004.

[21] M. Enachescu et al. Routers With Very Small Buffers. In *Proc. IEEE INFOCOM*, Spain, Apr 2006.

[22] A. Vishwanath, V. Sivaraman, and M. Thottan. Perspectives on Router Buffer Sizing: Recent Results and Open Problems. *ACM CCR*, 39(2):34–39, Apr 2009.

[23] Y. Zhang and D. Loguinov. ABS: Adaptive Buffer Sizing for Heterogeneous Networks. *Computer Networks*, 54(14):2562–2574, Oct 2010.

[24] X. Wu et al. Power and Performance of Read-Write Aware Hybrid Caches with Non-Volatile Memories. In *Design, Automation and Test in Europe*, France, Apr 2009.

[25] I. Norros. On the use of Fractional Brownian Motion in the Theory of Connectionless Traffic. *IEEE J. Sel. Areas Comm.*, 13(6):953–962, Aug 1995.

[26] D. Ostry. Synthesis of Accurate Fractional Gaussian Noise by Filtering. *IEEE Trans. Information Therory*, 52(4):1609–1623, Apr 2006.

[27] K. Park and W. Willinger, *Self-Similar Network Traffic and Performance Evaluation*. Wiley, 2000.

[28] Y. Jin, S. Bali, T. E. Duncan and V. S. Frost. Predicting Properties of Congestion Events for a Queueing System with fBm Traffic. *IEEE/ACM Trans. Netw.*, 15(5):1098-1108, Oct 2007.

[29] J. Chen, R. G. Addie and M. J. Zukerman. Performance Evaluation and Service Rate Provisioning for a Queue with Fractional Brownian Input. *Performance Evaluation*, 70(11):1028-1045, Nov 2013.

[30] CAIDA Packet Length Distributions. http://www.caida.org/research/traffic-analysis/AIX/plen_hist/.

[31] Voxel SLAs. https://www.voxel.net/sla.

[32] NetFPGA 1G and 10G Routers. www.netfpga.org.

[33] Ixia Traffic Generator. www.ixiacom.com.

[34] Anue Systems Network Emulator. www.anuesystems.com.

This submission is an extended and improved version of our paper presented at the 2011 ACM SIGCOMM CoNEXT conference [13].

Vijay Sivaraman (M '94) received his B. Tech. degree from IIT in Delhi, India, in 1994, his M.S. from North Carolina State University in 1996, and his Ph.D. from the University of California at Los Angeles in 2000, all in Computer Science. He has worked at Bell-Labs and a silicon valley startup. He is now an Associate Professor at the University of New South Wales in Sydney, Australia. His research interests include software-defined networking, and sensor networks for health applications.

Arun Vishwanath (SM '15, M '11) is a Research Scientist at IBM Research - Australia. He received the Ph.D. degree in Electrical Engineering from the University of New South Wales in Sydney, Australia, in 2011. He was a visiting Ph.D. scholar in the Department of Computer Science at North Carolina State University, USA in 2008. His research interests include software defined networking and energy-efficient networking. Arun is a Senior Member of IEEE.

Diet Ostry received the B.Sc. degree in Physics from the Australian National University, Canberra, in 1968 and the M.Comp.Sc. degree from the University of Newcastle, Australia, in 1996. He is currently a research scientist at the CSIRO ICT Centre in Sydney, Australia. His research interests include wireless communication, network traffic modeling, and security in wireless sensor networks.

Marina Thottan (M '00) received the Ph.D. degree in electrical and computer systems engineering from Rensselaer Polytechnic Institute in Troy, NY in, 2000. She is Director of the Mission-Critical Communications and Networking Group, Bell Laboratories, Murray Hill, NJ. Most recently, she has been leading work on smart grid communication networks. She has published over 40 papers in scientific journals, book chapters, and refereed conferences. Dr. Thottan is a Member of the Association for Computing Machinery.