# Adapting Router Buffers for Energy Efficiency

Arun Vishwanath
Centre for Energy-Efficient Telecommunications
University of Melbourne
arun.v@unimelb.edu.au

Vijay Sivaraman
School of EE&T
University of New South Wales
Sydney, Australia
vijay@unsw.edu.au

Zhi Zhao
School of EE&T
University of New South Wales
Sydney, Australia
zhi.zhao@student.unsw.edu.au

Craig Russell
ICT Centre, CSIRO
Sydney, Australia
craig.russell@csiro.au

Marina Thottan
Bell-Labs Alcatel Lucent
Crawford Corner, NJ, USA
marinat@alcatel-lucent.com

## ABSTRACT

Reducing the power consumption of core Internet routers is important for both Internet Service Providers (ISPs) and router vendors. ISPs can reduce their Carbon footprint and operational costs, while router manufacturers can achieve higher switching capacity per rack space. In this work, we examine the impact of packet buffers on the power consumption of backbone router line-cards. We argue that Gigabytes of always-on SRAM and DRAM buffers account for around 10% of the power, but are actively used only during transient periods of congestion. We propose a simple and practical algorithm for activating buffers incrementally as needed and putting them to sleep when not in use. We evaluate our algorithm on traffic traces from carrier and enterprise networks, via simulations in ns2, and by implementing it on a programmable-router test-bed. Our study shows that much of the energy associated with off-chip packet buffers can be eliminated with negligible impact on traffic performance. Dynamic adjustment of active router buffer size provides a low-complexity low-risk mechanism of saving energy that is amenable for incremental deployment in networks today.

## 1. INTRODUCTION

The ICT sector consumed 156 GigaWatts, or about 8% of the world's total electricity consumption, in 2007. Of this, 14% is attributed to network equipment [1]. In addition to the large carbon footprint, the power density of modern core routers is becoming a serious concern for ISPs – a single telecommunications rack today consumes tens of KiloWatts of power, and requires complex cooling systems to manage heat dissipation. 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 [2], 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 [3] 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 proposed in the literature have the potential to achieve (individually or in conjunction) 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 network operators, 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 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


ACM CoNEXT 2011, December 6–9 2011, Tokyo, Japan.

and enterprise networks, and find 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 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. Specifically, we apply it off-line to traffic traces from operational networks, simulate it on-line in ns2 with several thousand TCP flows, and implement it to operate in real-time on an experimental platform comprising an FPGA-based programmable router and hardware-based traffic generators. Our evaluations show that dynamic buffer control typically saves most of the energy associated with off-chip buffering (around 10% of the total energy), with negligible impact on traffic performance. Our specific contributions are:

- We argue that the energy costs associated with always-on packet buffers in today's routers are nearly 10%. We then present empirical evidence from several carrier and enterprise networks that link loads (and by inference buffer occupancies) are low for a large proportion of the time, which presents an opportunity to adapt router buffer size dynamically to save energy.
- We propose a practical mechanism for dynamic adjustment of buffer size. Our aim is to track buffer usage while still having some capability to absorb sudden bursts of packet arrivals. This trade-off between energy savings and risk protection can be controlled explicitly in our algorithm.
- We quantify the impact of our scheme in terms of energy savings and loss/throughput. We apply our algorithm off-line to traffic traces derived from Internet link data. We then simulate it in ns2, and profile its impact on TCP performance. Finally, we implement dynamic buffer adaptation in the gateware of the NetFPGA-based Gigabit router platform, and demonstrate its feasibility in a test-bed with realistic traffic.

Our aim is to 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: §2 motivates our focus on packet buffers and the opportunities to reduce their power consumption. Relevant background research is summarised in §3, while in §4 we present our dynamic buffer adjustment algorithm. §5 evaluates the algorithm via analysis, simulation and experimentation. The paper is concluded in §6.

# 2. THE CASE FOR REDUCING ROUTER BUFFER ENERGY

In this section we argue why it is worthwhile to focus on reducing power consumption associated with buffers in today's core routers. Our argument is laid out in three stages: in §2.1, we argue that always-on buffers account for around 10% of the router line-card's power consumption; in §2.2, we present empirical evidence that link-loads seldom go high (by inference, congestion is rare) on most links in observed carrier and enterprise networks; and in §2.3, we derive explicit queue occupancy traces from analysis and simulation to demonstrate that there is ample scope to save energy related to packet buffers with low risk of affecting performance.

## 2.1 Energy Cost of Packet Buffers

Core Internet routers have memory capacity to buffer over a million packets during periods of congestion. Moreover, at 40 Gbps, a 40-byte packet can arrive every 8 ns. To achieve large storage capacity while meeting such stringent throughput and latency requirements, routers need to employ various (off-chip) memory components in hierarchical configurations [4]. A core router line-card today has a few Gigabytes of dynamic RAM (DRAM), providing the bulk of the packet storage, as well as several Megabytes of static RAM (SRAM), acting as the packet cache. Focusing first on DRAM, power calculators from Micron [5], 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. At lower workloads, i.e. when there are few read/write operations, the power consumed by DRAM is lower but still non-negligible. It should also be noted that router manufacturers typically 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.

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 [6] 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 [7, 8, 9] 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%) [10], and around 10% of the total power of the system is attributable to buffer memory.

In addition to powering the packet buffer memory, power is also required to drive the memory controller circuitry that implements the logic to move packets between main memory, cache memory, and on-chip memory. As shown in [4], this is particularly challenging in high-speed routers which typically have long pipelines, and cache misses (e.g. when a head-of-line packet ready for transmission is not available in cache) introduce non-determinism that can cause the system to lose throughput in unpredictable ways. Other complicating factors can include multiple output queues (e.g. for class-of-service support), static (e.g. circular queues) versus dynamic (e.g. linked-lists) buffer allocation, 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 typically integrated into custom ASICs on most routers, making it very difficult to estimate their energy footprint accurately. A reasonable estimate can be obtained by noting that the DRAM memory controller on the AMD Opteron 6100-series multi-core processor [11]) accounts for 15-20% of the chip's power consumption. For a network processor such as EZchip NP-4 [12] (which operates at 50 Gbps and consumes 35 Watts), DRAM controllers would conservatively account for 5-7 Watts. Bearing in mind that modern routers have complex memory pipelines across DRAM, SRAM, and on-chip buffers, and often have separate ingress and egress queueing ASICs (as in the CRS-1), it would be reasonable to expect that memory controllers consume at least half as much power as the memory chips themselves.

Based on the above arguments we conservatively estimate that the power consumed by packet buffers (i.e. memory chips and controllers) would amount to around 10% of the total power of a line-card in modern high-speed routers. This number is high enough to motivate the study in this paper to optimise the energy consumption of buffer memory, particularly because buffers are meant to absorb congestion, which is a relatively infrequent event in operational networks, as discussed next.

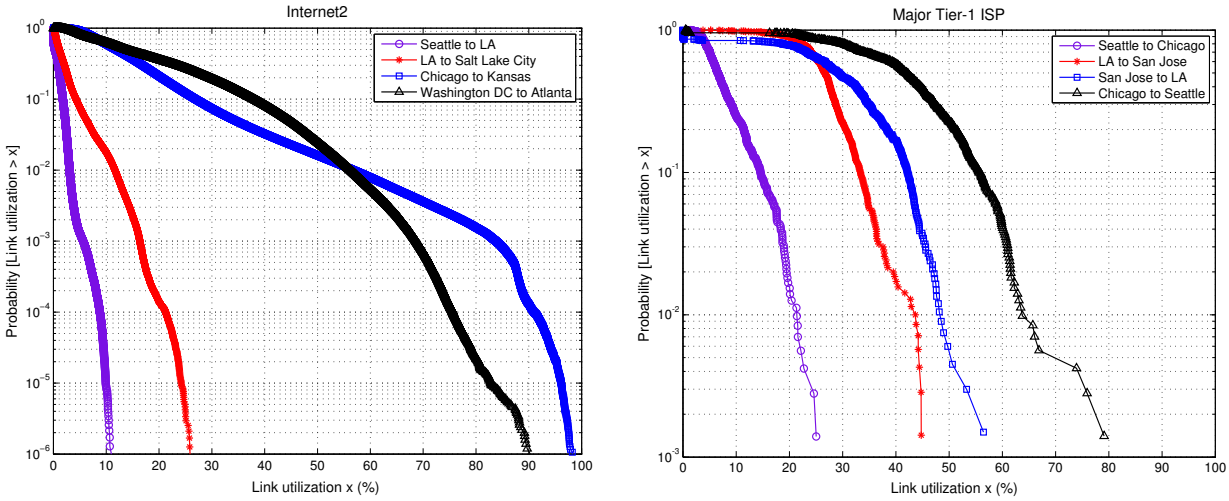
## 2.2 Link Congestion in Operational Networks

Having seen that buffers consume a non-negligible fraction of the energy in a core router, we now present empirical evidence from carrier and enterprise networks that link loads are quite low for a vast majority of the time, implying that the buffer capacity built into routers to deal with worst-case congestion situations are needed only rarely. We have 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. In this section, we briefly summarise our observations from two backbone networks (Internet2 and a major Tier-1 carrier ISP), and two enterprise networks (a large University of over 40,000 students and a large governmental organisation of over 6,000 employees).

The first backbone network we discuss is Internet2 [13], chosen for the comprehensive data it freely provides on its national long-distance network. We analyse load from over 50 links at 10-second granularity over the past three years (spanning Nov 2007 to Nov 2010). All links are of 10 Gbps capacity, and our general observation was that nearly all links had light loads ($< 30\%$) most of the time. To depict this, we show in Fig. 1(a) the complementary cumulative distribution function (CCDF) of the link utilisation, namely 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%.

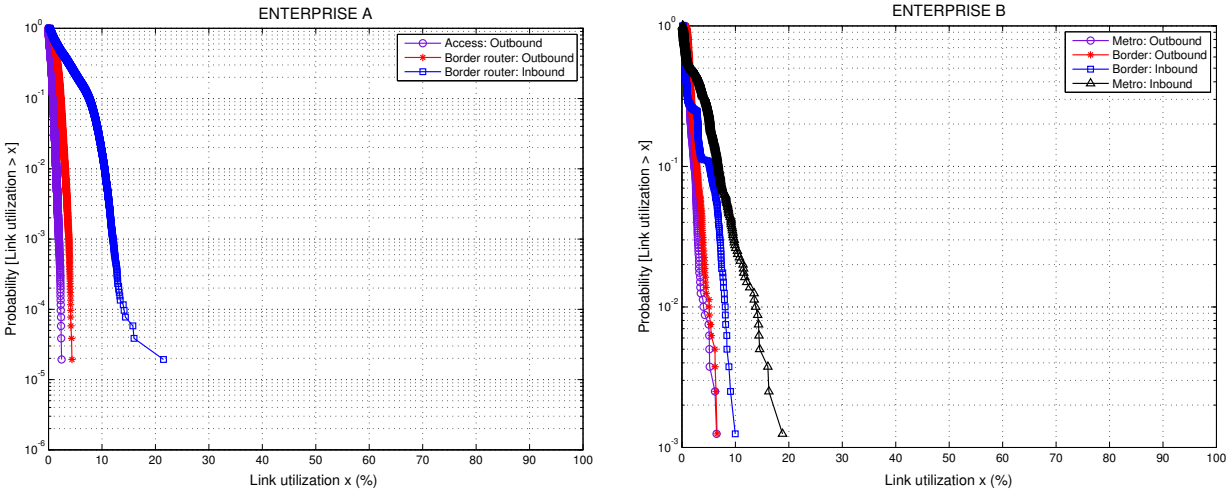
At this point one may wonder that even if the load over a 10-second interval is low, there might well be many spikes in the traffic at smaller time-scales (say over a millisecond), which use the large buffers in the router over that interval. While it is very difficult to get link-load data at time-scales finer than 10 seconds from operational networks over any sustained period of time, a simple argument can be made to show that there can only be a bounded number of such traffic spikes. For example, say the link load is 20% (or lower) over a 10-second interval (we found this to be the case most often), but we are interested in traffic load at a much finer time-scale, say a millisecond. We can use the Markov inequality to bound the number of 1 millisecond slots that have high loads: the Markov inequality states that $P(X > nE[X]) \leq 1/n$ for non-negative $X$, and hence at most one-third of the 1 millisecond slots within that 10 second interval can have a load greater than 60%, which means that at least two-thirds of the slots have a load lower than 60% (the bound is quite loose and in reality many more slots would have a low load). Thus no matter what the time granularity we pick, there will be sufficient intervals of low load, which is a strong indicator that the buffers for that link would be occupied minimally during most periods.

In Fig. 1(b) we plot the CCDF of load on four links belonging to a major Tier-1 ISP. The data for these plots was obtained from CAIDA, and spans a period of two years (2008-2010). For three of the links, the load never exceeds 60%, and it does so with a 5% probability for the fourth link (Chicago to Seattle). This should not




(a) Internet2 (b) Tier-1 ISP (from CAIDA)

**Figure 1: CCDF of link load from two backbone networks**




(a) Univ. of New South Wales (40,000+ students) (b) Govt. Organisation: CSIRO (6,000+ employees)

**Figure 2: CCDF of link load from two enterprise networks**

come as a surprise, since carriers in general know their traffic profiles quite well, and provision their links to ensure that loads are not consistently high. Nevertheless, the data 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.

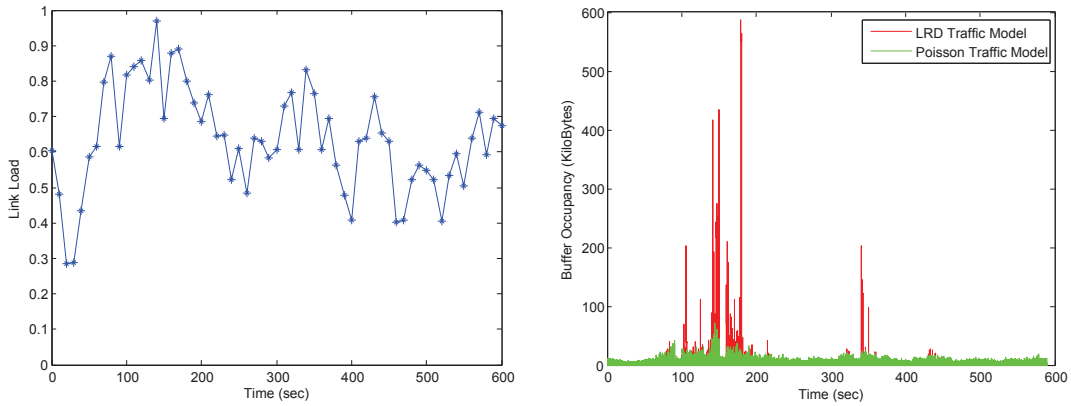
We also obtained comprehensive load data for several links in two enterprise networks: a large University (UNSW) of 40,000+ students, and a large government organisation (CSIRO) of 6000+ employees. Fig. 2(a) shows the CCDF of load (in each direction) on the University's external link (to the Internet) and on an internal link within the campus (the load in the two directions was mostly symmetrical and hence only one direction is shown in the figure), measured at 5-minute intervals over a six-month period (July-Dec 2010). All links had 1 Gbps capacity, and as the figure shows, the loads never exceeded 25%. Fig. 2(b) shows the CCDF of link loads on external (to Internet) and internal (between sites) links in the government organisation's network spanning several cities spread across Australia, and again shows that links (of 1 Gbps) had low loads ($< 20\%$) for much of the time.

## 2.3 Buffer Occupancy - Analysis / Simulation

The previous subsection only depicted link load, which is an indirect measure of congestion. In this subsection we explicitly depict router buffer usage, derived from analysis of the traffic load traces and ns2 simulations.

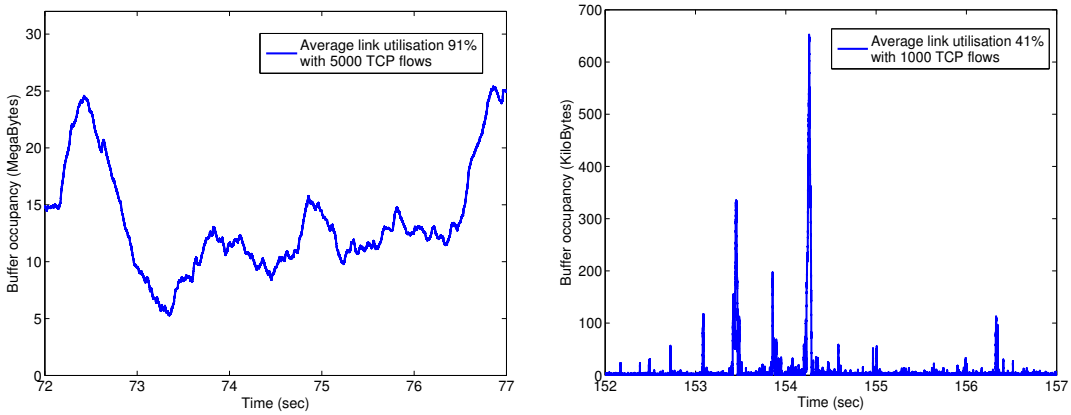
### *2.3.1 Trace Analysis*

Our first approach is to take link load data from operational networks, and generate synthetic traffic of




(a) Link load (b) Buffer occupancy

**Figure 3:** 10**-min link load and buffer occupancy traces (Poisson, LRD models) on an Internet2 link**




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

**Figure 4: Buffer occupancy from ns2 of** 1000 **and** 5000 **TCP flows sharing a** 1 **Gbps core link**

matching load, which is then fed into a FIFO queue simulation to generate the buffer occupancy trace. We used two models for generating traffic – 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 §5.1.1). To illustrate the outcome, we consider the Internet2 link from Chicago to Kansas, and show in Fig. 3(a) the load on that link over a 10-minute period of high load (reaching 96%) observed on 17 Sep, 2009. We used our Poisson and LRD models to generate traffic with matching rate (that changes every 10 sec), and show in Fig. 3(b) 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, under the Poisson model, buffer occupancy barely exceeds 40 KB, while with the LRD model, the buffer occupancy shows more burstiness, and is seen to spike occasionally to over 500 KB (which corresponds to about 4 ms of buffering at the link rate). The point being made is that the traffic loads we are observing on links in operational networks can lead to large excursions in buffer occupancy, but these are very rare, and it would seem wasteful in energy to have all buffers active at all times to deal with such rare events.

### *2.3.2 ns2 Simulation*

To corroborate that buffer occupancy fluctuations can be seen with closed-loop TCP traffic, we conducted tens of simulations in ns2. Space constraints limit the observations we can report from various topologies, link speed settings, number of flows and mixes of short- and long-lived flows. We consider one specific scenario with TCP flows from 1000 users multiplexing at a 1 Gbps core link. A vast majority of the TCP flows were short-lived, transferring files with Pareto distributed sizes and having exponential think times between transfers. The scenario is described in more detail in §5.2. In Fig. 4 we plot the queue occupancy for two link loads over a five second interval: Fig. 4(a) has 5 flows per user (5000 flows in total) creating a load of 91%, while Fig. 4(b) has 1 flow per user (total 1000 flows) creating a load of 41%. In both cases the buffer size was set to 31.25 MB, corresponding to the delay-bandwidth product. 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. This scenario presents ample scope to save energy by putting off-chip buffers to sleep for much of the time.

# 3. RELATED WORK

In recent years there have been many proposals to reduce the energy consumption of telecommunications networks. Some works recommend that energy efficiency be a major consideration in the network design stage [14, 15], such as in choosing appropriate combinations of grooming at the optical WDM layer and switching at the IP layer to reduce overall network energy consumption [14], and in choosing the configuration of interfaces and chassis to achieve the desired switching capacity [15]. Other approaches suggest selectively turning off or underclocking network elements such as interfaces and line-cards [16] to save energy during periods of low load, and yet others suggest new routing mechanisms [17, 18] to redirect traffic towards "greener" areas of the Internet (such as powered by renewable sources). We refer the reader to a recent survey paper [19] for a more comprehensive discussion of proposals for energy conservation in telecommunications networks. While all the above approaches hold promise for substantial energy savings, 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, though providing more modest energy savings (around 10% by our estimate), requires very minimal changes and is virtually transparent to operators and users of the network, and stands a much better chance of incrementally being deployed in the short term.

Our work also draws inspiration from recent debates on buffering capacity required at core Internet routers. Studies have suggested that buffers can be safely reduced by two to three orders of magnitude [20], and even made as low as a few tens of packets [21]. While the debate about the right amount of buffering required at a router continues (we refer the reader to our survey article on the topic [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 palatable 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 (rather than build-time), ISPs can gradually become comfortable with the idea of operating with reduced active buffers (which they can control in our algorithm), making them more likely to adopt routers built with smaller buffers in the future.

To the best of our knowledge our work is the first to propose dynamic adaptation of router buffer size with the primary aim of reducing energy consumption. Earlier works such as [23] have proposed to adapt buffer size primarily for meeting pre-specified loss criteria, delay or utilisation bounds. Our approach is also different: we do not use feedback-based schemes (which may warrant new protocols) or operator input (since they are quite likely to ask for zero loss and maximum throughput), but instead try to make the buffer size adaptation nearly invisible to the operator (i.e. the operator does not perceive any noticeable effect on traffic performance).

# 4. DYNAMIC BUFFER ADJUSTMENT

We now develop a simple and practical dynamic buffer adjustment algorithm. We begin by describing our buffer architecture (§4.1), its energy model (§4.2), and then present our algorithm (§4.3) and its analysis (§4.4).

## 4.1 Buffer Architecture

The focus in this work is on egress packet buffers, which absorb output link congestion, and ingress buffers, that can also absorb (link or fabric) congestion in architectures supporting back-pressure. We do not consider buffers that internally segment and reassemble packets for transmission across the switching fabric(s). The architecture of packet buffer memory varies from one platform to another, and in this paper we consider a fairly generic three-level hierarchical model (taken from [4]). As shown in Fig. 5, it consists of on-chip (within the Network Processor (NP) or ASIC) buffers, off-chip cache (SRAM), and off-chip bulk memory (DRAM). We assume that the on-chip buffer memory has capacity $B_I$, typically a few tens or hundreds of Kilobytes (for example 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 $B_S$, of the order of a few Megabytes, while the bulk DRAM has capacity $B_D$, of the order of several Gigabytes. 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.

To meet the speed and latency requirements (for example at 40 Gbps a packet can arrive every 8 ns), a number of memory banks or chips are employed in parallel. To illustrate this in the architecture, Fig. 5 shows four SRAM chips, each with 16 data-pins, that operate in parallel to present a 64-bit data bus to the network processor (via the controller) for increased throughput. Since DRAM is slower than SRAM (by a factor of four), DRAM is accessed via a wider data bus – the figure shows DRAM organised as a $4 \times 4$ grid, with multiple

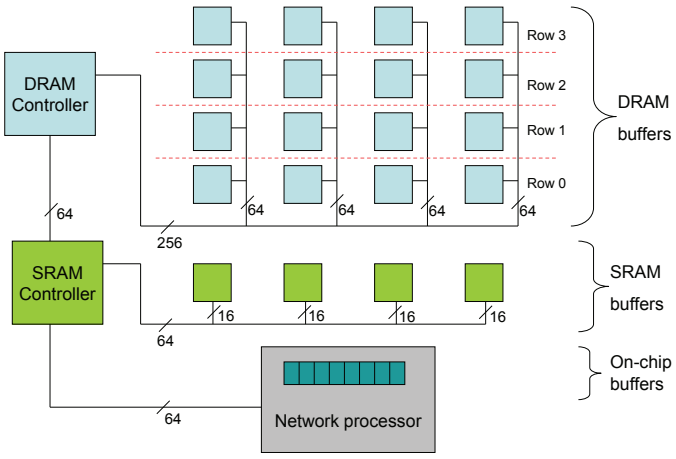



**Figure 5: Generic model of buffer architecture**

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). The data-bus widths and number of memory chips in the figure are chosen merely to illustrate the concept, and would need to be adapted when analysing a specific routing platform. To be generic, we use $N_R$ to denote the number of rows of DRAM chips in this organisation.

Memory controllers are typically integrated into custom ASICs, and there are often several concurrent controllers; however, for ease of depiction we have shown a single aggregated controller for DRAM and similarly for SRAM. Our algorithm will put to sleep or activate an entire row of memory chips (DRAM or SRAM). 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 $50\mu$s to switch on (i.e. to bring from inactive to active state) the SRAM (controller and memory), and likewise about $500\mu$s for the DRAM.

## 4.2 Energy Model

The power consumed by DRAM is highly dependent on 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. 5) of capacity 2 Gigabits consumes 2W when active, 200mW when idle, and 20mW when asleep – these numbers are derived from Micron [5] 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 [6] 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.

## 4.3 Algorithm for Dynamic Buffer Adjustment

To save maximum energy, it is desirable to have the active buffer capacity track the actual queue occupancy, and to put to sleep any off-chip buffer memory that is not needed. However, the risk in doing so is that if a sudden burst of traffic arrives, there may not be sufficient time to activate buffer memory without dropping packets from this burst. 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 broad idea is to make the total active buffer capacity $B$ at any time instant stay between the lower bound of $Q$ (the current queue occupancy) and upper bound of $B_I + B_S + B_D$ (the maximum available buffer space). 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 the extremely conservative setting of $\alpha = 0$ sets active buffers to maximum available buffers, essentially disabling power control. On the other (aggressive) extreme, choosing $\alpha = 1$ 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 $\alpha$ 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: $\alpha$, $B_I$, $B_S$, $B_D$, $N_R$
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 $\alpha$ 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 $\alpha$ (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 $\alpha$ (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 $\alpha$) 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.

## 4.4 Discussion

Our algorithm is relatively easy to implement in hardware. It is executed whenever the queue occupancy $Q$ changes, either due to packet arrivals or departures. If $1-\alpha$ is chosen to be a negative power of 2 (e.g. $\alpha = 0.75$ or 0.875), all steps can be performed without any multiplication or division operations, since the product in steps 1 and 3 can be precomputed for given $\alpha$, and steps 6-8 can be realised using shift and add operations. When the algorithm returns an active buffer size $B$ higher than is currently active, an additional memory row (i.e. SRAM or a row of DRAM) is activated. Likewise, when the required buffer size computed by the algorithm is lower than the current active buffers, the corresponding row of memory chips (i.e. SRAM or of DRAM) is put to sleep. However, to prevent memory components toggling between active and sleep states in quick succession, it is wise to have some hysteresis protection; specifically, our implementation (described in §5) mandates that any memory component, once active, should not be put to sleep for at least 1ms.

Though it is easy to envisage more complex algorithms for determining the best buffer size, such as by attempting to predict how queue occupancy will evolve, we believe they will be too complex for real-time hardware implementation. We have intentionally chosen to keep it simple, and have strived to have only one user input parameter $\alpha$. There are however some unavoidable risks in turning buffer memory elements on/asleep to save energy. In the worst-case, on-chip buffers of size $B_I = 100$ KB can go from zero to full occupancy within $1.25\mu$s 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 $50\mu$s. Likewise, SRAM of capacity $B_S = 4$ MB can fill within $50\mu$s at 640 Gbps, an order of magnitude quicker than the DRAM turn-on time of $500\mu$s. However, such worst-case scenarios are exceedingly improbable, and were never observed in the traces, simulations, and experiments we describe in §5. 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 $\alpha$ 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 section evaluates our algorithm via trace analysis, simulation, and experimentation.

# 5. EVALUATION

We use three methods to evaluate our algorithm: off-line application to traffic traces generated from real Internet data (§5.1), on-line simulation of TCP flows in ns2 (§5.2), and real-time implementation on an experimental testbed of NetFPGA routers (§5.3).

## 5.1 Off-Line Trace Analysis

For our off-line study we generated synthetic Poisson and long range dependent (LRD) traffic traces using time-varying load obtained from empirical data in carrier and enterprise networks (as discussed in §2.2). The packet trace was fed into a simulation of our algorithm, and the resulting performance metrics such as power savings and packet loss ratios were obtained.

### *5.1.1 Traffic Generation*

For each link considered, we generated Poisson and LRD traffic with mean rate varying as per the link load trace for that link. For example, the load on the Internet2 link from Chicago to Kansas (depicted in Fig. 3(a)) is measured every 10 sec, and we therefore changed the mean rate of the generated traffic over each 10-sec interval to match the measured load. Traffic from the Poisson model did not exhibit sufficient burstiness to cause queue occupancy to go high (as confirmed in Fig. 3(b)), and henceforth we concentrate on the LRD model, which generates burstiness more reflective of Internet traffic. Our LRD traffic generator (derived from Norros' self-similar traffic model [25]) combines a mean arrival rate that is constant over each 10-sec interval with fractional Gaussian noise (fGn) having Hurst parameter $H \in [1/2, 1)$. We use the filtering method from [26] to generate long sequences of normalised fGn (zero mean and unit variance) samples with $H = 0.85$. These samples are scaled and added to the desired mean volume of traffic in each discretization interval, and the resulting fluid volume is accumulated into packets of variable size with distribution derived from CAIDA's measurements over 87 million packets at the NASA Ames Internet Exchange (AIX) [27]. 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.

### *5.1.2 Dynamic Buffer Adaptation*

The packet trace derived as described above from the measured link load was fed to our algorithm for dynamic buffer size adjustment. The measured link loads were very low for the most part, and to illustrate our algorithm we pick here a 10-min period, observed on 17 Sep, 2009, of relatively heavy load on the Chicago to Kansas Inernet2 link. The load and corresponding queue occupancy have already been shown in Fig. 3, and it was seen that even in this period of relatively heavy load, queue occupancy does not exceed 600 KB. This would easily fit in on-chip and off-chip SRAM, 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. 5). In Fig. 6 we show a trace of the buffer occupancy and buffer size set by our algorithm over a

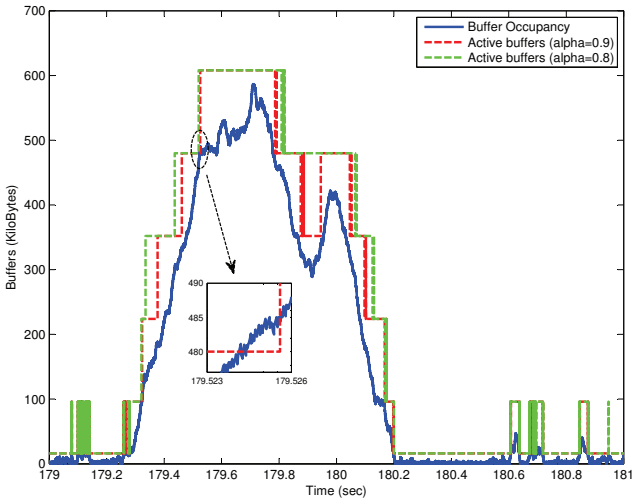



**Figure 6: Trace of buffer occupancy and active buffers for $\alpha = 0.8$ and $0.9$ from algorithm**

chosen 2-sec interval. As the figure shows, the buffer size is initially set at 16 KB (i.e. all off-chip buffers are in the sleep state). As the buffer occupancy increases (at around 179.2 sec), the algorithm reacts by first activating SRAM and then successively each of the rows of DRAM, which subsequently become inactive when the buffer occupancy falls beyond 180.2 sec. The impact of parameter $\alpha$ 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.

### *5.1.3 Power vs. Loss Trade-Off*

Continuing with our above example of traffic on Internet2's Chicago to Kansas link, we applied our algorithm off-line to the trace with various values of the parameter $\alpha$ to measure power savings and impact on packet loss. When $\alpha = 0$, the algorithm is effectively turned off. We progressively increased $\alpha$, and found that for $\alpha < 0.7$, 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 $\alpha$ 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

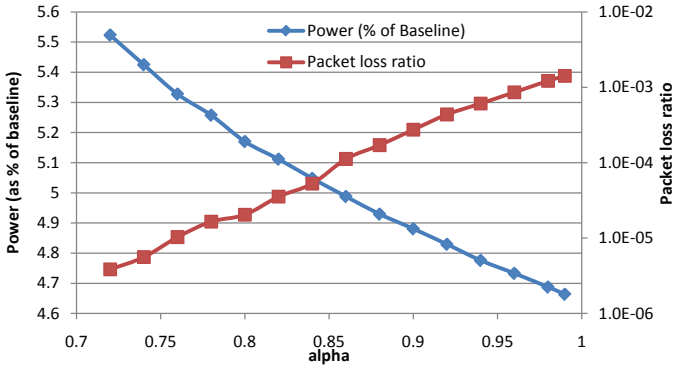



**Figure 7: Power-savings versus loss trade-off**

and 10 times/sec, while it was $< 1$ for the DRAM rows. For the range $\alpha \in (0.7, 1)$, the performance of the algorithm is depicted in Fig. 7. Taking the point corresponding to $\alpha = 0.8$, we found that our algorithm activated SRAM for only 2.66% of the time (significantly lower than when $\alpha = 0.7$), and the four DRAM rows for 0.83%, 0.61%, 0.20% and 0.05% of the time, respectively. We found that SRAM toggled between states 62.4 times/sec, while the DRAM rows toggled states between 0.1-2.7 times/sec. While the baseline power consumption was 6.66W, our algorithm reduced the average power consumption to 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 $2 \times 10^{-5}$ packets (which is within the tolerance of $10^{-3}$ typical of many SLAs, e.g. [28]). The figure shows that as $\alpha$ 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.

## 5.2 On-Line ns2 Simulations with TCP

Though the previous study demonstrated tangible benefits when dynamic buffer adaptation was applied off-line to traces from operational networks, it did not tell us how losses would affect TCP performance, which carries over 90% of Internet traffic. We therefore incorporated our algorithm in ns2 and ran several tens of simulations with different topologies, link speeds, number of flows, and mixes of long-/short-lived TCP flows. Here we present results for a small subset of scenarios to illustrate the efficacy of dynamic buffer adaptation with a realistic traffic mix under different loading conditions.

The topology we used comprised of a three-level hierarchy: one core bottleneck link fed by 50 edge links, with each edge link aggregating traffic from 20 access links. There were thus 1000 source hosts generating TCP traffic. The core and edge links had capacity 1 Gbps, while the access links had capacity uniformly distributed in $[100, 300]$ Mbps. The mean RTT for the flows was 250 ms. The total number of TCP (Reno) flows was varied from 1000 to 5000 (by varying the number of flows per user) to simulate different loading conditions. Our simulations comprised of 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.

| Workload | Load | AFCT | Power saved |
|---|---|---|---|
| Low | 21.5% | 2.233 sec | 97.4% |
| Medium | 41.1% | 2.244 sec | 97.2% |
| High | 59.8% | 2.250 sec | 83.4% |
| Heavy | 78.6% | 2.295 sec | 52.9% |
| Very heavy | 90.9% | 2.757 sec | 11.6% |

**Table 1: Power savings and average flow completion times (AFCT) from ns2 simulations**

A sample trace of queue occupancy obtained via simulation from two of the above representative scenarios have already been shown in Fig. 4. We repeated the simulations with our algorithm for dynamic buffer adaptation (implemented at the core link), and tried parameter settings $\alpha$ in $[0.75, 0.95]$. The results are summarised in Table 1. At low to medium workloads (up to 41%) the off-chip SRAM and DRAM buffers were used very sparingly ($\approx 0.25\%$), thus saving over 97% of the off-chip buffering energy. There were no packet losses induced by 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 $\alpha$ in $[0.75, 0.95]$ gave identical results.

Next, when the load went relatively high (i.e. 59.8%), we observed that the off-chip buffers were used about 12.3% of the time. In spite of this, our algorithm resulted in over 83% energy savings. It however induced a very small fraction of packet loss (of the order of $10^{-7}$), which barely increased AFCT by a few ms. Even under heavy workload regime (corresponding to 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, under very heavy load ($> 90\%$), off-chip buffers

were (unsurprisingly) used nearly 82% of the time, and power savings are limited to about 11%. Even for this scenario, our algorithm induced very small loss (i.e. $< 10^{-6}$, which is again within the tolerance of typical SLAs [28]), and AFCT was barely affected (by no more than 6 ms). These results clearly show that our algorithm performs well across a wide range of workloads with negligible impact on TCP traffic performance.

## 5.3 Real-Time Implementation in a Router

The aim of this section is to demonstrate the feasibility of deploying our scheme in hardware, and to quantify the power savings in the presence of real TCP traffic. To do so, we consider the programmable NetFPGA platform in conjunction with hardware-based traffic generators and delay emulators, as described next.

### 5.3.1 Implementation and Set-Up

Our algorithm is implemented in the hardware dataplane using Verilog and extends the gateware available at the NetFPGA website for router buffer sizing studies [29]. Specifically, we incorporated our algorithm in the *evt_capture_oq_plugin.v* file located in the output queues module. 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. 5). The algorithm is executed at every packet arrival or departure instant to determine the size of buffers 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. Note that our implementation does not explicitly put memory elements to sleep nor does it artificially introduce delays to model memory state transition latencies. The objective of our experimental work is to demonstrate the feasibility of implementing our scheme in hardware and quantifying the potential power gains. We tried various settings of $\alpha$, and the results described are for $\alpha = 0.8$, which was found to yield a good balance between power and loss performance.

The hardware packages several (typically 340) events at the output queue into a special "event packet" that is sent to the host software for display on the GUI. We extended the software 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.

### 5.3.2 Validation with UDP Traffic Burst

The first objective is to validate our hardware implementation. To this end we rate-limited the output port to 62 Mbps and fed in a burst of UDP traffic at four times the output link rate. The burst was generated using a high-precision hardware-based IXIA [30] traffic generator. Fig. 8 shows (lower blue curve) how the queue occupancy rises from being empty to nearly full (about 500 KB) in just a couple of ms. Our algorithm is able to detect and respond to these changes in real-time. The figure denotes (upper red curve) the active buffer size triggered by the algorithm during various stages of the UDP traffic burst. At its peak, the active buffers hits the maximum available buffer size of 512 KB. Once the input burst stops at 2782 ms, the output queue begins to drain and goes empty after roughly 62 ms. The algorithm keeps track of the changes in queue occupancy and adjusts the active buffer size accordingly (the red step lines), finally setting it to 16 KB (the on-chip buffer size) when the queue occupancy hits zero.

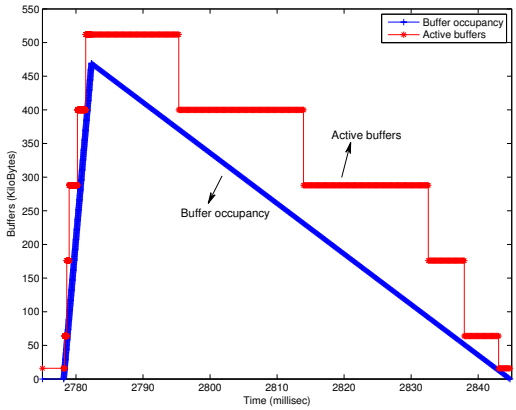



**Figure 8: Buffer adjustment for UDP burst**

### 5.3.3 Power Savings with TCP Flows

In this experiment we had 150 concurrent TCP flows (generated using Iperf) sharing the 123 Mbps link under observation for 180 sec. The RTT was set, using a hardware-based delay emulator from Anue Systems [31], 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 (using IXIA) in another downstream link so that the link under observation toggles 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). It is seen that our algorithm adapts dynamically to buffer occupancy.

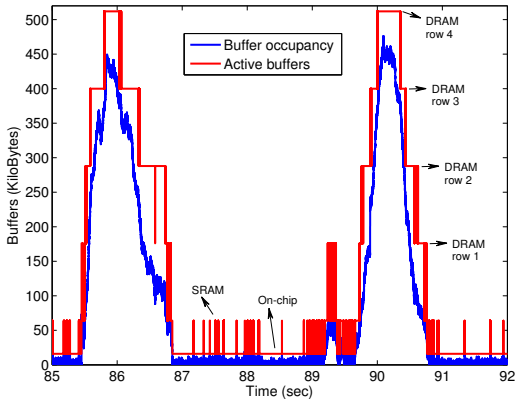



**Figure 9: Buffer adjustment with 150 TCP flows**

The algorithm used SRAM buffers 44.3% of the time, while DRAM rows 1-4 are 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 1) 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.

## 6. CONCLUSIONS

We believe that much of the off-chip packet buffer energy (typically 10% of the total line-card energy) in backbone routers can be saved by selectively putting to sleep memory components when not needed. While packet buffers are just one component of a router, and the energy savings from our proposal are thus limited to around 10%, this reduction comes at virtually no cost: the hardware/software changes required for dynamic buffer size adaptation are minimal. Since these changes are contained within a router, the scheme can be deployed incrementally today without requiring any new network protocols or architectures. The risk of affecting traffic performance is also 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 $\alpha$ in the algorithm. The scheme is therefore almost fully transparent to the operator and the user of the network. 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 power consumption.

**Acknowledgments:** We sincerely thank our shepherd Andrew Moore and the anonymous reviewers for their insightful comments and suggestions, which vastly improved the quality and presentation of the paper.

## 7. REFERENCES

[1] M. Pickavet. Network Solutions to Reduce the Energy Footprint of ICT. In *European Conference on Optical Communication (ECOC) Symposium*, Belgium, Sep 2008.
[2] R. Tucker et al. Energy Consumption of IP Networks. In *Proc. Europ. Conf. on Optical Comm.*, Belgium, Sep 2008.
[3] GreenTouch. `www.greentouch.org`.
[4] S. Iyer, R. R. Kompella, and N. McKeown. Designing packet buffer for router linecards. *IEEE/ACM Trans. on Networking*, 16(3):705–717, Jun 2008.
[5] Micron-Technology-Inc. System power Calculators. `www.micron.com/support/dram/power_calc.html`.
[6] Cypress-Semiconductor. QDR-II SRAM CY7C1515KV18. `http://www.cypress.com/?docID=24145`.
[7] R. Tucker et al. Evolution of WDM Optical IP Networks: A Cost and Energy Perspective. *IEEE/OSA J. of Lightwave Tech.*, 27(3):243–252, Feb 2009.
[8] S. Aleksic. Analysis of Power Consumption in Future High-Capacity Network Nodes. *IEEE/OSA J. Optical Comm. and Netw.*, 1(3):245–258, Aug 2009.
[9] O Tamm. Scaling and Energy Efficiency in Next Generation Core Networks and Switches. In *ECOC Sunday Workshop.*, Austria, Sep 2009.
[10] 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`.
[11] AMD Opteron processors. `http://www.amd.com/acp`.
[12] EZchip 50 Gbps NP-4 Network Processor. `http://www.ezchip.com/Images/pdf/NP-4_Short_Brief_online.pdf`
[13] Internet2 Network. `http://www.internet2.edu/network/`.
[14] G. Shen and R. Tucker. Energy-Minimized Design for IP over WDM Networks. *IEEE/OSA J. of Optical 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, P. Chowdhury, M. Tornatore, and B. Mukherjee. 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. Selected Areas in 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] CAIDA packet length distributions. `http://www.caida.org/analysis/AIX/plen\_hist/`.
[28] Voxel SLAs. `https://www.voxel.net/sla`.
[29] NetFPGA 1G and 10G Routers. `www.netfpga.org`.
[30] Ixia traffic generator. `www.ixiacom.com`.
[31] Anue Systems Network Emulator. `www.anuesystems.com`.