Sloth: A Kernel-Bypass Scheduler Maximizing Energy Efficiency under Latency Constraints

Clément Delzotti
ICTEAM/INGI, UCLouvain
Louvain-La-Neuve, Belgium
clement.delzotti@uclouvain.be

Pol Maistriaux
ICTEAM/ELEN, UCLouvain
Louvain-La-Neuve, Belgium
pol.maistriaux@uclouvain.be

Tom Barbette
ICTEAM/INGI, UCLouvain
Louvain-La-Neuve, Belgium
tom.barbette@uclouvain.be

***Abstract*—In recent years, multi-hundred-gigabit networking applications such as Virtual Network Function (VNF) and Key Value Store (KVS) implementations have relied on *kernel-bypass* and *polling* to achieve maximum throughput. However, this performance improvement comes at the expense of high CPU usage and power consumption.**

This paper first analyses the trade-off between the power consumption, the latency and the throughput of VNF applications. We then present Sloth, an energy-aware scheduler that adapts the number of cores used by an application and their frequency. Sloth uses the information gathered in a training phase to maximize the energy reduction in real time while maintaining a user-provided service-level objective. Sloth manages to reduce CPU power consumption by up to 50% compared to the classical DPDK polling approach with only a 30 µs latency increase. Sloth also saves up to milliseconds of latency compared to state-of-the-art solutions at equivalent power consumption.

***Index Terms*—dvfs, nfv, dpdk, energy efficiency, networking**

I. Introduction

Over the last decade, Kernel Bypass Networking (KBN) has shown to be an efficient technique to enable network-intensive applications. For instance, many virtual network functions have integrated KBN stacks such as DPDK [1]. These KBN frameworks are used in key-value stores [2] deep packet inspection [3] and monitoring [4], virtual switches [5] and TCP stacks [6]. They are also integrated in packet processing frameworks such as FastClick [7] or VPP [8] to build network functions like routers and firewalls.

KBN handles packets directly in userspace to avoid both the convoluted operating system's kernel networking stack and crossing the user and privileged boundaries for packet reception and transmission [9]. As these applications receive raw packets in userspace they lose at the same time kernel facilities for networking (socket API, TCP stack, firewalling, QOS, ...), and the ability to unschedule a process when no packets are being transmitted, as the kernel is not aware of I/O completion. KBN applications, therefore, generally rely on polling, where the CPU cores spin-loop over the network interface, consequently pushing the CPU cores to full capacity and increasing the power consumption [10]. Therefore, as network traffic tends to be bursty [11], significant energy is lost due to useless polling operations during off periods. In addition, the excessive load generated by polling can decrease the lifespan of the CPU [12], leading to a higher replacement rate of the hardware, and an increased embodied carbon footprint of the service.

ISBN 978-3-903176-72-0

Previous work have investigated ways to prevent KBN applications from spin-looping between burst of packets, using short sleep periods [13], [14], Dynamic Voltage and Frequency Scaling (DVFS) [15], sometimes combined with core scaling [16]. However, they all rely on simple heuristics in practice. This prevents reaching the most optimal core count/frequency combination that can absorb the current input rate, while satisfying the Service Level Objective (SLO) defined by the user. That is, the maximum amount of time a packet should spend in the system either in the Network Interface Controller (NIC) queue or in processing.

In this work, we explore and evaluate the design space of possible leverages on energy consumption and latency and conclude that a combination of core scaling and dynamic voltage and frequency scaling (DVFS) can lead to an improved energy efficiency while enforcing a specified SLO. We propose Sloth, a controller running alongside an application, seamlessly tracking and optimizing the application's energy efficiency at run-time. Sloth relies on both a user-provided latency and the results of a training phase for a given application to calculate the optimal system configuration in the studied design space. In this approach, Sloth stands out from existing state-of-the-art solutions through this adjustable latency objective, allowing users to move throughout the Pareto front built on empirical data. Sloth can save 25 Watt of CPU power consumption while maintaining latency comparable to *polling*. With a more relaxed SLO constraint, Sloth can be configured to match the power consumption of state-of-the-art solutions with only a slight increase in latency, whereas other methods suffer from an order of magnitude higher tail latencies.

The remainder of the paper is organized as follows: Section II discusses the techniques used in the literature and the lessons learned to inform the design of Sloth. Sections III examines the trade-offs between the number of cores, their frequency, the latency and the CPU power consumption. Observations made from these measurements are used to determine the optimal configuration according to a given SLO in Section IV. Section V shows Sloth's architecture, which is followed by Sloth's evaluation in Section VI. Finally, the outcomes of this paper are summarized in Section VII.

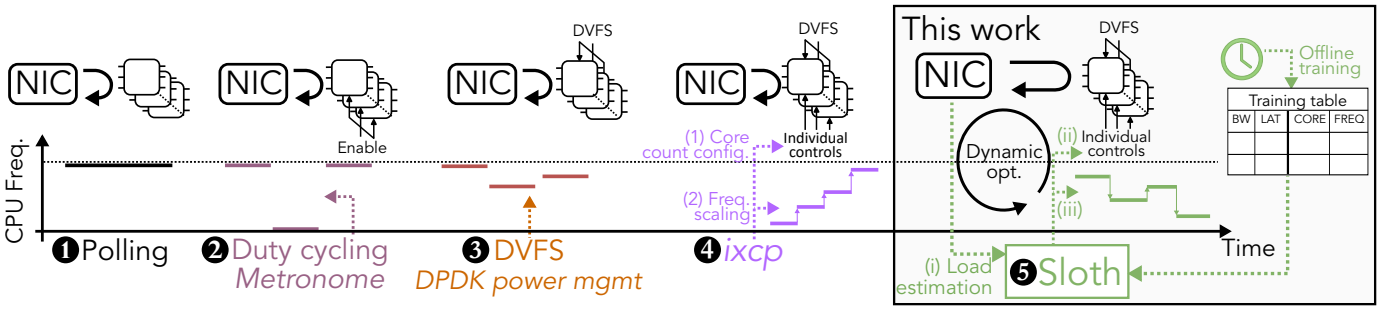


Fig. 1: State-of-the-art techniques to enable energy saving through different Network Interface Card (*NIC*) operations in polling applications or operating systems and our proposed system, Sloth.

II. Energy-saving Techniques

We start by presenting state-of-the-art energy savings techniques and their shortcomings, with a summary shown in Figure 1. Kernel-bypass I/O frameworks [1], [9] rely on polling (Fig. 1.❶) where all CPU cores used by the application spinloop over the NIC queues until some packets are ready to be processed. By relying on **duty cycling** (Fig. 1.❷), Metronome [13] alternates between periods of packet processing and sleeping. Note, the primary goal of Metronome is to relinquish processing time from the polling application for other applications, not especially targeting energy efficiency.

Traditional operating systems, such as the *ondemand* scheduler of Linux, enable the use of **DVFS** (Fig. 1.❸) that modulates the core frequency to achieve quadratic energy savings. The frequency linearly affects the system performance and power consumption, while the voltage has a quadratic relationship with the power consumption. Since a system running at a lower frequency can operate at a lower voltage, DVFS reduces the CPU frequency at the cost of an increased execution time, but provides an overall lower energy consumption for the same task. However, the benefits of DVFS diminish below a certain point, as static power consumption – independent of clock frequency – becomes the dominant factor in the system's total power usage.

As DPDK uses polling, the OS cannot detect a reduced load and, therefore, cannot benefit from DVFS's low-power state or CPU deep sleep. Therefore, DPDK proposes a power management library [1] that can dynamically tune the frequency of the CPU cores according to the current load of the application, measured through the amount of polling that returned no packets. A significant downside of this library lies in its lack of core scaling, forcing applications to run on a fixed set of cores. Li et al. [14], [17] build a model to formalize the packet processing pipeline of high-performance packet processing applications. From observations of their model, they build two heuristics (TUPE and TUKE) to scale up and down the frequency of the CPU, and propose a version combined with duty cycling (called NAP).

ixcp [16] (Fig. 1.❹) explores *energy proportionality*. After a study of the optimal DVFS configuration (also including hyper-threading as a parameter), they build a heuristic to grow resources according to the current load. In summary, heuristics use more cores as it provides more performance per Joules, then increases the frequency of the cores. Nevertheless, *ixcp* is limited by a lack of precise latency objective, relying on an estimate based on the queue occupancy. *ixcp* is co-designed with the IX prototype Operating System, limiting its adoption.

We claim all the aforementioned techniques miss a broad exploration of the design space, which would lead to a more optimal configuration. For instance, *ixcp* explores some optimal configurations, but for a single SLO. We explore the Pareto front of various SLOs and observe that significant energy savings can be made when sacrificing a small fraction of latency.

A. Sloth

We observe all these proposal use simplistic heuristics. We propose **Sloth**, a scheduler capable of adjusting in real time the active core count of an application and the running frequency to reach the best power efficiency trade-off based on a complete pre-exploration of the design space.

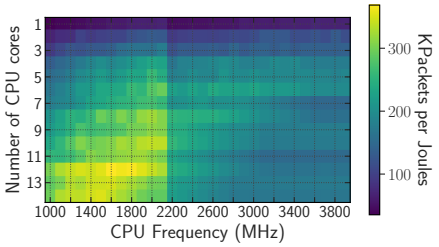
Figure 1.❺ shows Sloth's workflow. Sloth will periodically compute statistics on the current input of an application and adjust the number of cores and their operating frequency and voltage in consequence. As DVFS automatically scales the voltage with respect to the frequency to achieve maximum energy savings, we will only refer to the frequency as a design parameter.

Sloth relies on a pre-exploration of the parameter space (frequency, core) that allows building a table of the best configuration given a certain input rate. The learning phase also measures what latency objective a given configuration can satisfy.

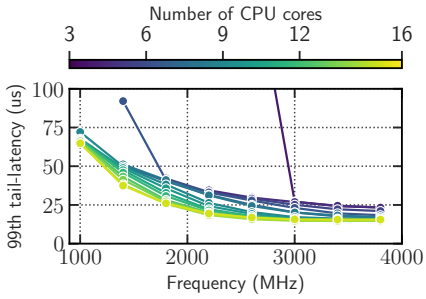
At runtime, the user can, therefore, act on the latency objective to dynamically trade it for energy savings. Sloth outperforms all state-of-the-art solutions as its exploration space is a superset of all previous work.

B. Related Work

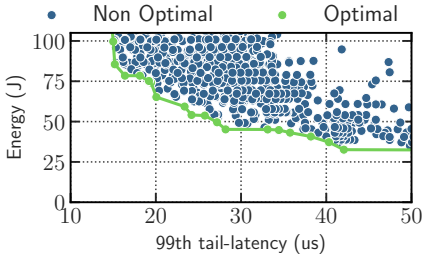
Most KBN schedulers focus on dispatching packets to spread the load equally between a minimal amount of cores. For instance, eRSS [18] offloads a load-balancer on the NIC to dynamically allocate CPU cores depending on the load while RSS++ [19] spreads the load more equally across



(a) Energy efficiency according to frequency and cores configurations under a constant 16MPPS throughput.



(b) Tail-latency observed depending on the frequency and the number of cores when processing a 6 Mpps throughput



(c) The energy consumption and the latency for configuration that reach at least 6Mpps. The Pareto frontier, in green, is the most power-efficient configurations acc. to the latency.

Fig. 2: Firewall behaviour under different frequencies. As the frequency diminishes, both latency and energy-efficiency increase, leading to a Pareto front showing the optimal values to exploit the latency/energy trade-off.

cores and tries to minimize the number of cores used by an application. Systems like ZygOS [16] or Shinjuku [20] focus on maintaining a good tail latency but not particularly looking at power consumption. Energy-aware scheduling and load-balancing are still understudied. Most efforts concentrate either on energy-efficient deadline scheduling [21], [22] or are built for large workloads such as AI training [23]. Limitations of such approaches make them unsuitable for network-intensive applications where packets do not rely on explicit deadlines but rather on guaranteed latency. Pegasus [24] proposes to achieve energy proportionality for datacenters by using load-balancing. Sloth offers a knob for such large-scale systems into NFV appliances.

III. Core, Frequency, Latency and Power Trade-off

In this section, we look at the trade-off between the number of cores and their operating frequency to influence both the energy consumption and the SLO of a KBN.

A. Experiments details

Measurements have been conducted on a stateless firewall built with FastClick [7]. FastClick integrates the DPDK [1] kernel-bypass framework in the Click Modular Router, among other techniques for high-speed packet processing. These experiments were run on Grid5000 [25] *gros* cluster. Machines are equipped with 96 GiB of main memory, 18 cores *Intel Xeon Gold 5220* CPUs (Cascade Lake architecture) and two 25 Gbps *Mellanox Technologies MT27710 ConnectX-4*. We use pair of machines with one packet generator and one DUT running the firewall.

The firewall consists of a thousand rules filtering packets based on their IP addresses. The input load is always comprised of 64-byte UDP packets at a variable rate, according to the need of the experiment. We use small packets as they represent a worst case scenario and ease the understanding of the trade-offs, but later evaluate dynamic variations of the load and diverse packet types and sizes with a real trace in Section VI. Packet inter-arrival time follows an exponential distribution. In this paper, we always report the **99th percentile packet Round-Trip-Time (RTT) tail latency** as computed by the generator.

We use *RAPL* [26] to measure the energy consumption of the CPU. The total consumption of the machine only variates by a constant of ~ 60 W.

B. Frequency/Core Ratio

Figure 2a shows the thousands of packets handled per Joules in every core/frequency cross-product on the left part. The highest efficiency is achieved by using nearly all cores below the nominal frequency (2200MHz). Overall, the experiment shows one should always try to use more cores at a lower frequency, but the sweet spot is not solely reducible to keeping cores at the minimal frequency and raising the frequency afterwards as proposed by *ixcp*.

Hossfeld *et al.* [27] highlight increasing the energy intensity does not necessarily mean increasing the energy efficiency if the workload is processed with a higher efficiency but takes, overall, a longer time. In the NFV context, the input load is an external parameter. The goal is therefore to find the most power-efficient configuration that can sustain the given input, therefore leading to a better energy efficiency as cycles spent on over-provisioned polling simply go to waste. However, processing packets at a lower frequency has the downside of augmenting the latency, as the time needed to process packets is longer too. Figure 2b shows the round-trip time latency of packets using a varying number of cores according to the frequency on the X-axis. Configurations that cannot sustain the load see the latency increase as queues build up. The figure, combined with Figure 2a, outlines a clear trade-off where one could exchange μs in latency to gain in energy efficiency and vice versa. Energy efficient configurations identified earlier, i.e., with multiple cores running at low frequencies, only suffer from a reduced latency penalty, as shown in the bottom lines of Figure 2b.

Additionally, one can notice how sensitive the latency can be when tweaking the frequency. Especially at low frequencies, an error of a few hundred MHz can dramatically increase the

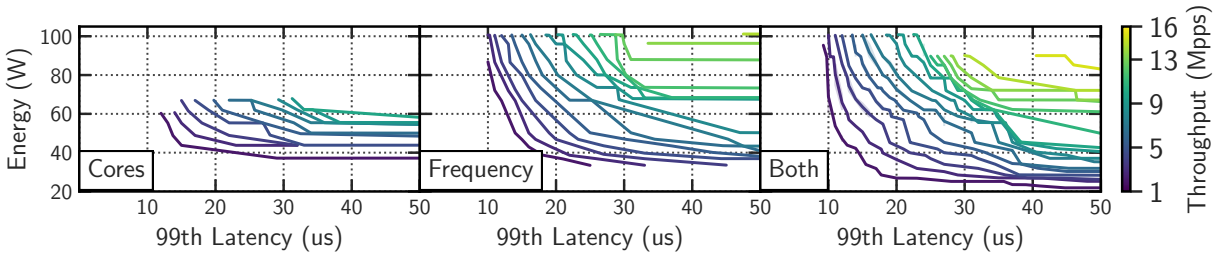


Fig. 3: Pareto frontiers with different scaling applied. The *core-only* graph shows CPU scaling with a fixed frequency of 2.2 GHz while *frequency-only* uses 11 cores. Varying both at the same time is necessary for optimal efficiency.

latency. This indicates saturation of the cores which become unable to process the entire load, causing high queueing times and ultimately packet drops.

IV. SLO Pareto Optimality

Given the analyzed trade-offs, we ought to find which core/frequency combination will allow processing all packets without any drop, while drawing minimal average power consumption. This optimal configuration depends on the service level objective desired by the user.

A. Pareto Frontier

Figure 2c shows what constitutes a *Pareto Frontier*. For each of the blue points consisting in a particular core count/frequency pair that can sustain 6Mpps, there is at least one green point which consumes the same amount of energy or less for a lower latency. Every blue point is deemed suboptimal. Such a Pareto Front can be built for multiple throughput.

B. Frequency and Core scaling

Figure 3 shows the achievable energy savings by changing, respectively, only the number of CPU cores used by the application, the frequency of the cores, and both at the same time. Running at nominal frequency and varying only the number of cores does not allow reaching high throughputs nor lowest latencies. However, the combination of both core and frequency scaling allows us to improve energy savings by up to $\sim 20\,\mathrm{W}$.

With a low throughput, the very steep slope visible in Figure 3 around $10\,\mu\mathrm{s}$ indicates a very profitable trade-off. Sacrificing only a handful of microseconds can already drive the application near the $\sim 40\,\mathrm{W}$ floor values obtainable with further latency concessions. On another hand, a higher throughput flattens the Pareto frontier, leading to a higher latency sacrifice for the same amount of energy.

C. Duty Cycling

We then investigate the use of duty cycling, i.e. putting the CPU in deep sleep between bursts of packets, as a way to reduce the power further.

One can put the CPU to sleep when no packets are received, as network traffic is naturally bursty. The longer the execution stays in such a state, the more energy is saved. But this comes at the risk of being asleep when packets arrive, with packets queuing leading to higher delays. We evaluate 2 modes, one where the sleeping time is constant (Constant Sleeping) and one where it is progressive (Progressive Sleeping). We evaluate additive and exponential backoffs for progressive sleeping, i.e. while there is no packet to process, increasing the sleep time by a constant term, or multiplying the sleep time exponentially up to a certain maximum.

While we hoped there would be a situation where the CPU could go to deep sleep between bursts of packets, even at low rates , there were never any situations where just running at the *right* core/frequency ratio would not be best. With long sleep delay, sleeping enables the CPU cores to go to a deep sleep state and save a handful of Watts. But this incurs ahigh tail latency for some packets. Therefore, in the remainder of this work, we omit duty cycling and Sloth only relies on finding the right frequency/core ratio.

V. Sloth Architecture

The workflow of Sloth has been presented in Section I. We present here how the findings of previous sections are used to train Sloth for a particular application to efficiently select the Frequency/Cores combination that minimizes the energy consumption while ensuring a service-level objective.

Sloth dynamic scheduling is applicable to any NFV application. We target DPDK as a wide-spread kernel-bypass framework but results can apply to other kernel-bypass techniques such as Netmap [9], or AF_XDP [28] or Shenango [29]. Sloth is available, as well as the training tools and the dataset of explored configurations, at https://github.com/UCLouvain-ENSG/Sloth.

A. Training

We use essentially the measurements that lead to the Pareto fronts to build a table that, given a throughput and latency objectives from the user, gives a tuple of the number of cores and frequency to apply, as well as the CPU load observed during training.

To train a new system, we explore the combinations of target latencies, input load/throughput, number of enabled cores and their operating frequency. We build a few shortcuts to avoid exploring the whole design space. We start with the highest input rate, number of cores and highest frequency. We reduce the rate until we find the maximal zero-loss-throughput using a binary search. Then, we know any configuration with either fewer cores or lower frequency cannot achieve a higher rate,

given a relatively parallelizable application, which is generally possible in networking with RSS [30].

From this exploration, we can define for each latency objective and for each throughput, which number of cores/frequency pair is the least energy-consuming. This information is loaded by Sloth to perform lookups at runtime.

B. Sloth Implementation

Sloth's main loop consists of a fast routine that is executed at a default frequency of 10 times per second. It only needs to gather statistics and lookup in the table, and is therefore negligible in term of CPU cost. Once Sloth decides the number of cores should change, the *RSS* indirection table of the NIC is modified to steer traffic away from that particular core. We propose the use of RSS++ [19] technique to migrate flow state, although, for small network functions such as a load-balancer, we observe the contention of the single lock-less hashtable is minimal as in prior work [31]. Cores are put to sleep using a relatively large sleep period so it can start before the next tick of Sloth. To ensure traffic is not steered before cores are waken up, we use a semaphore to ensure the NIC is reconfigured only when the last core is waken up.

C. Latency Knob

Sloth requires two essential pieces of information to look up in the hashtable: current throughput and the target latency. The first is retrieved by gathering data from the Ethernet device by Sloth. The latency is, however, dependent on the measurement method used during the training phase. In the measurement phase, the RTT latency is given by the packet generator. At runtime, the same measurement technique is not always possible.

We observe the Pareto front has a minimal latency, even at very high power, and a maximal latency objective where there is no power gain anymore, even allowing a higher latency (without dropping packets). Applications where the latency is not measurable in the same way as the training phase can, therefore, use the latency objective as a knob to move along the Pareto front. Sloth has a REST API that allows to dynamically trade latency for energy and vice-versa.

D. Dynamic Adjustment

As Sloth cannot rely on direct runtime-latency feedback that would match what was measured at runtime, we devise a mechanism to adapt to variation in the workload. Although Sloth constantly applies the best configuration for the given input rate, if the workload per packet is different than the one used at training, the actual configuration could not be the most efficient anymore. Therefore, we ensure Sloth stays around the load observed for the configuration currently applied. We observe that if the application is put in a given core/frequency/input rate configuration, and if the actual CPU load is higher than the one observed during the training phase, then the latency will be higher as the queue is filled faster. Following this observation, when Sloth runs its main loop, we apply a penalty factor on the input load, forcing Sloth to use a configuration for a higher (or lower) throughput until the load stays around an experimentally driven threshold of 20 % of the original observed load. While this mechanism is sufficient to sustain the variations in the workload, such as a change in the number of firewall rules, the configuration selected might not be the most efficient for that particular workload. One should re-train on this different workload. We leave potential online approaches to future work.

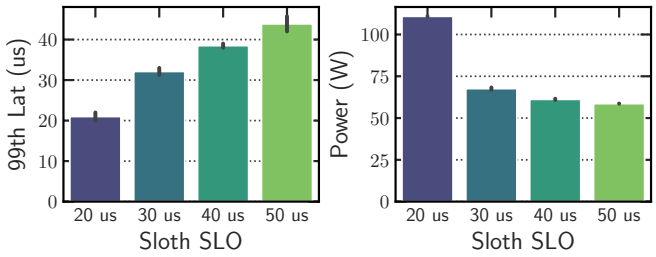


Fig. 4: Running Sloth in the training conditions (firewall, small packets) at 8 Mpps. Error bars show the standard deviation among 3 runs.

VI. Evaluation

This section evaluates the energy gains of Sloth while maintaining the user's designated service level objective. First, we investigate Sloth's ability to meet the user-provided SLO while reducing the power consumption under the training condition. We then explore how Sloth is fit for realistic traffic with a real trace. We also compare Sloth against other state-of-the-art solutions.

A. Training Conditions

Figure 4 shows how Sloth behaves under a constant throughput with minimal-sized packets, as used in the training phase with the same setup described in Section III-A. We repeat the experiment 3 times and draw the mean and standard deviation. We observe that Sloth selects an appropriate frequency/core configuration using the training table, maintaining the target tail latency while leveraging the available margin to save energy.

B. Variable Input

To show the approach works on multiple systems, for the second experiment, we run measurements on a local testbed equipped with a *Intel®Xeon®Gold 5317 CPU @ 3.00GHz* interconnected by 100GbE *NVIDIA Mellanox ConnectX-6 Dx NICs*. We run a load-balancer FastClick configuration instead of the firewall one. We replay packets using the CAIDA [32] trace. We accelerate the trace using the technique proposed in [33] to create an artificial variation that can be seen in Figure 5. In this experiment, Sloth is trained using the CAIDA trace, replayed at various acceleration rate following the algorithm described in Section V-A. Sloth's power consumption with an SLO of 70 µs remains much lower than *Polling*, saving more than 30 W at the cost of a $\sim$ 30 µs latency increase.

With a SLO of $\sim 20\,\mu s$ Sloth achieves a tail latency similar to Polling, while saving 25 Watts in average.

C. Comparison with State-Of-The-Art Methods

Figure 5 also shows how Sloth performs compared to multiple alternatives, namely TUPE [17], *ixcp* [16] and DPDK Power Management API [15]. We implement all these techniques in FastClick to enable a fair comparison.

We first compare to TUPE, one of the two proposed algorithms from [17][1]. TUPE uses duty cycling and does not target the tail latency, nor a specific latency objective. Therefore, the tail latency grow very high because of packets arriving when a core is sleeping.

As *ixcp* is tied to the IX Operating System, we re-implement the scaling algorithm of *ixcp* as in the paper [16] and its associated code. *ixcp* uses queue occupancy as an indication for latency. It starts with a single core at the lowest frequency. If the occupancy grows above a threshold, *ixcp* will use more cores one at a time, and after using all the cores, will increase the frequency by steps. However, we observe the queue occupancy has mostly a binary state, either being quite low (a few packets) or nearly completely full. Therefore *ixcp* starts oscillating between two configurations, one even dropping packets, the next one leading to nearly no queuing. Moreover, running at the lowest frequency induces a baseline latency of $60\,\mu s$ as shown in Section IV-B. On the other hand, Sloth uses configuration with a bit more frequency but fewer cores, reaching low latency at a similar power cost.

Both TUPE and *ixcp* [16] provide a low power consumption with approximately 60 W but exhibit extremely high latencies at the scale of multiple milliseconds. Meanwhile, *DPDK Power API* [1] is capable of saving a handful of Watts while providing lower latencies on a scale of hundreds of microseconds. When allowing higher latencies, Sloth is capable of reaching similar power consumptions to other techniques while sacrificing only a few microseconds. In such configuration, Sloth is capable of saving approximately 50 % of the CPU energy consumption needed by classical *Polling* methods.

VII. Conclusion

After analyzing the trade-off between latency and energy consumption, we presented Sloth, which dynamically tunes both the frequency of the CPU and the number of allocated cores to satisfy a certain level objective while lowering the CPU consumption. Sloth uses a training phase to observe the throughput and latency of the application over various CPU configurations. Combined with a latency objective provided by the user, Sloth is capable of saving up to 50 % compared to regular polling and outperforms existing energy-efficient KBN solutions.

Future work will focus on removing this training phase using online learning techniques, but also on widening the scope of current applications to include other KBN solutions such as Key/Values stores.

[1]The paper mentions TUKE does not offer much gain compared to TUPE. Moreover, TUKE relies on a perfect synchronization of day-to-day traffic throughput, which is not necessarily observable in practice.

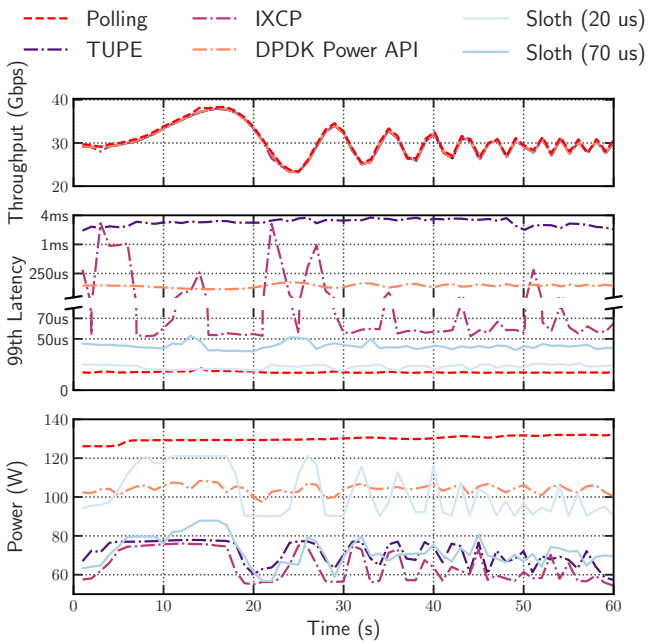


Fig. 5: Scaling of a load-balancer with Sloth and state-of-the-art techniques under dynamic workload.

Acknowledgement

We would like to acknowledge the anonymous reviewers for their valuable feedback. Experiments presented in this paper were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities as well as other organizations (see https://www.grid5000.fr). Clément Delzotti is a FRIA grantee of the Fonds de la Recherche Scientifique – FNRS. Local equipments were funded by the UCLouvain FSR. We also acknowledge Basile Leretaille and his internship co-supervisor, David Bol, who helped develop an early version of this work.

References

[1] Linux Foundation. (2023) DPDK: Data Plane Development Kit. [Online]. Available: https://www.dpdk.org/

[2] H. Lim, D. Han, D. G. Andersen, and M. Kaminsky, "{MICA}: A holistic approach to fast {In-Memory}{Key-Value} storage," in *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, 2014, pp. 429–444.

[3] D. Day, "A performance analysis of Snort and Suricata Network Intrusion Detection and Prevention Engines," in *ICDS 2011 : The Fifth International Conference on Digital Society*. IARIA, Feb. 2011, pp. 187–192.

[4] G. Wan, F. Gong, T. Barbette, and Z. Durumeric, "Retina: analyzing 100gbe traffic on commodity hardware," in *Proceedings of the ACM SIGCOMM 2022 Conference*, 2022, pp. 530–544.

[5] N. Pitaev, M. Falkner, A. Leivadeas, and I. Lambadaris, "Characterizing the performance of concurrent virtualized network functions with ovs-dpdk, fd. io vpp and sr-iov," in *Proceedings of the 2018 ACM/SPEC International Conference on Performance Engineering*, 2018, pp. 285–292.

[6] K. Yasukata, M. Honda, D. Santry, and L. Eggert, "{StackMap}:{Low-Latency} networking with the {OS} stack and dedicated {NICs}," in *2016 USENIX Annual Technical Conference (USENIX ATC 16)*, 2016, pp. 43–56.

[7] T. Barbette, C. Soldani, and L. Mathy, "Fast userspace packet processing," in *2015 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*. Oakland, CA, USA: IEEE, May 2015, pp. 5–16. [Online]. Available: http://ieeexplore.ieee.org/document/7110116/

[8] D. Barach, L. Linguaglossa, D. Marion, P. Pfister, S. Pontarelli, and D. Rossi, "High-Speed Software Data Plane via Vectorized Packet Processing," *IEEE Communications Magazine*, vol. 56, no. 12, pp. 97–103, Dec. 2018, conference Name: IEEE Communications Magazine. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8535089

[9] L. Rizzo, "Netmap: a novel framework for fast packet I/O," in *Proceedings of the 2012 USENIX conference on Annual Technical Conference*, ser. USENIX ATC'12. USA: USENIX Association, Jun. 2012, p. 9.

[10] M. Shah, M. Yunus, P. Vachhani, L. Monis, M. P. Tahiliani, and B. Talawar, "Powerdpdk: Software-based real-time power measurement for dpdk applications," in *2020 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*. IEEE, 2020, pp. 13–18.

[11] K. Thompson, G. Miller, and R. Wilder, "Wide-area internet traffic patterns and characteristics," *IEEE Network*, vol. 11, no. 6, pp. 10–23, 1997.

[12] A. Tomlinson and G. Porter, "Something old, something new: Extending the life of cpus in datacenters," *SIGENERGY Energy Inform. Rev.*, vol. 3, no. 3, p. 59–63, oct 2023. [Online]. Available: https://doi.org/10.1145/3630614.3630625

[13] M. Faltelli, G. Belocchi, F. Quaglia, S. Pontarelli, and G. Bianchi, "Metronome: adaptive and precise intermittent packet retrieval in DPDK," in *Proceedings of the 16th International Conference on emerging Networking EXperiments and Technologies*, ser. CoNEXT '20. New York, NY, USA: Association for Computing Machinery, Nov. 2020, pp. 406–420. [Online]. Available: https://dl.acm.org/doi/10.1145/3386367.3432730

[14] X. Li, W. Cheng, T. Zhang, F. Ren, and B. Yang, "Towards power efficient high performance packet i/o," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 4, pp. 981–996, 2019.

[15] Linux Foundation. (2025) DPDK Power PMD Management Library. [Online]. Available: https://doc.dpdk.org/api/rte__power__pmd__mgmt_8h.html

[16] G. Prekas, M. Primorac, A. Belay, C. Kozyrakis, and E. Bugnion, "Energy proportionality and workload consolidation for latency-critical applications," in *Proceedings of the Sixth ACM Symposium on Cloud Computing*, ser. SoCC '15. New York, NY, USA: Association for Computing Machinery, Aug. 2015, pp. 342–355. [Online]. Available: https://doi.org/10.1145/2806777.2806848

[17] X. Li, W. Cheng, T. Zhang, J. Xie, F. Ren, and B. Yang, "Power efficient high performance packet i/o," in *Proceedings of the 47th International Conference on Parallel Processing*, 2018, pp. 1–10.

[18] A. Rucker, M. Shahbaz, T. Swamy, and K. Olukotun, "Elastic RSS: Co-Scheduling Packets and Cores Using Programmable NICs," in *Proceedings of the 3rd Asia-Pacific Workshop on Networking*, ser. APNet '19. New York, NY, USA: Association for Computing Machinery, Aug. 2019, pp. 71–77. [Online]. Available: https://dl.acm.org/doi/10.1145/3343180.3343184

[19] T. Barbette, G. P. Katsikas, G. Q. Maguire, and D. Kostić, "RSS++: load and state-aware receive side scaling," in *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies*, ser. CoNEXT '19. New York, NY, USA: Association for Computing Machinery, Dec. 2019, pp. 318–333. [Online]. Available: https://doi.org/10.1145/3359989.3365412

[20] K. Kaffes, T. Chong, J. T. Humphries, A. Belay, D. Mazières, and C. Kozyrakis, "Shinjuku: Preemptive scheduling for {μsecond-scale} tail latency," in *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, 2019, pp. 345–360.

[21] N. E. K. Chadi Assi, Sara Ayoubi and L. Qu, "Energy-Aware Mapping and Scheduling of Network Flows With Deadlines on VNFs | IEEE Journals & Magazine | IEEE Xplore," 2017. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8588354

[22] S. H. Rastegar, H. Shafiei, and A. Khonsari, "EneX: An Energy-Aware Execution Scheduler for Serverless Computing," *IEEE Transactions on Industrial Informatics*, pp. 1–13, 2023, conference Name: IEEE Transactions on Industrial Informatics.

[23] H. Liu, B. Liu, L. T. Yang, M. Lin, Y. Deng, K. Bilal, and S. U. Khan, "Thermal-Aware and DVFS-Enabled Big Data Task Scheduling for Data Centers," *IEEE Transactions on Big Data*, vol. 4, no. 2, pp. 177–190, Jun. 2018, conference Name: IEEE Transactions on Big Data. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8070316

[24] D. Lo, L. Cheng, R. Govindaraju, L. A. Barroso, and C. Kozyrakis, "Towards energy proportionality for large-scale latency-critical workloads," in *Proceeding of the 41st Annual International Symposium on Computer Architecuture*, ser. ISCA '14. IEEE Press, 2014, p. 301–312.

[25] R. Bolze, F. Cappello, E. Caron, M. Daydé, F. Desprez, E. Jeannot, Y. Jégou, S. Lanteri, J. Leduc, N. Melab, G. Mornet, R. Namyst, P. Primet, B. Quetier, O. Richard, E.-G. Talbi, and I. Touche, "Grid'5000: A Large Scale And Highly Reconfigurable Experimental Grid Testbed," *The International Journal of High Performance Computing Applications*, vol. 20, no. 4, pp. 481–494, Nov. 2006, publisher: SAGE Publications Ltd STM. [Online]. Available: https://doi.org/10.1177/1094342006070078

[26] K. N. Khan, M. Hirki, T. Niemi, J. K. Nurminen, and Z. Ou, "RAPL in Action: Experiences in Using RAPL for Power Measurements," *ACM Transactions on Modeling and Performance Evaluation of Computing Systems*, vol. 3, no. 2, pp. 9:1–9:26, Mar. 2018. [Online]. Available: https://doi.org/10.1145/3177754

[27] T. Hossfeld, S. Wunderer, F. Loh, and D. Schien, "Analysis of energy intensity and generic energy efficiency metrics in communication networks: Limits, practical applications and case studies," *IEEE Access*, 2024.

[28] M. Karlsson and B. Topel. (2018) The Path to DPDK Speeds for AF XDP.

[29] A. Ousterhout, J. Fried, J. Behrens, A. Belay, and H. Balakrishnan, "Shenango: Achieving High {CPU} Efficiency for Latency-sensitive Datacenter Workloads," 2019, pp. 361–378. [Online]. Available: https://www.usenix.org/conference/nsdi19/presentation/ousterhout

[30] Intel. (2016) Rss: Receive-side scaling. Accessed on 16th of March, 2025. [Online]. Available: https://www.oasis-open.org/2023/01/11/creating-a-leading-peripheral-component-interconnect-express-pcie-based-ethernet-host-interface-standard-infrastructure-data-plane-function-idpf.html

[31] M. Girondi, M. Chiesa, and T. Barbette, "High-speed Connection Tracking in Modern Servers," in *2021 IEEE 22nd International Conference on High Performance Switching and Routing (HPSR)*, Jun. 2021, pp. 1–8, iSSN: 2325-5609. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9481841

[32] The CAIDA UCSD. (2018) Caida anonymized internet traces dataset, 2018.

[33] G. P. Katsikas, T. Barbette, D. Kostić, J. G. Q. Maguire, and R. Steinert, "Metron: High-performance nfv service chaining even in the presence of blackboxes," *ACM Trans. Comput. Syst.*, vol. 38, no. 1–2, Jul. 2021. [Online]. Available: https://doi.org/10.1145/3465628