Improving Energy Efficiency in NFV Clouds with Machine Learning

Ligia M.M. Zorello*, Migyael G.T. Vieira*, Rodrigo A.G. Tejos*, Marco A.T. Rojas†,
Catalin Meirosu‡ and Tereza C.M.B. Carvalho*
*Escola Politécnica da Universidade de São Paulo, Sao Paulo, Brazil
Email: {ligia.zorello, migyael, terezacarvalho}@usp.br, rtejos@larc.usp.br
†Instituto Federal de Santa Catarina, Caçador, Brazil
Email: marco.rojas@ifsc.edu.br
‡Ericsson, Stockholm, Sweden
Email: catalin.meirosu@ericsson.com

***Abstract*—Widespread deployments of Network Function Virtualization (NFV) technology will replace many physical appliances in telecommunication networks with software executed on cloud platforms. Setting compute servers continuously to high-performance operating modes is a common NFV approach for achieving predictable operations. However, this has the effect that large amounts of energy are consumed even when little traffic needs to be forwarded. The Dynamic Voltage-Frequency Scaling (DVFS) technology available in Intel processors is a known option for adapting the power consumption to the workload, but it is not optimized for network traffic processing workloads. We developed a novel control method for DVFS, based observing the ongoing traffic and online predictions using machine learning. Our results show that we can save up to 27% compared to commodity DVFS, even when including the computational overhead of machine learning.**

***Keywords*-energy efficiency; NFV; machine learning; Dynamic Voltage and Frequency Scaling**

I. Introduction

With overall traffic expected to increase to about 77 Exabytes by 2022 [1], operators are looking for ways to benefit from economies of scale and introduce further agility at the edge of the network [2]. Network Function Virtualization (NFV) will enable economies of scale by replacing countless physical appliances with software running on distributed cloud platforms.

Central Office locations [3] and base station points-of-presence in telecommunication networks are in the same class of sites that, when housing small distributed data centers, were found to consume 95% of the total data center power consumption in the USA [4]. As they are equipped with commodity servers for NFV edge clouds, every Watt that could be saved matters tremendously.

Central Processing Units (CPUs) are the major source of power consumption in servers, in particular for telecommunications-oriented equipment. Recent high-end server-grade CPUs have as many as 24 cores and a data sheet nominal power consumption of over 450 Watts. The Dynamic Voltage and Frequency Scaling (DVFS) capability of the CPU [5] is widely used in enterprise environments for meeting performance requirements while reducing the power consumption. The CPU capabilities are complemented at the operating system level, for example through software in the Linux operating system known as DVFS governors. However, setting compute servers continuously to high-performance operating modes is a common NFV approach for achieving predictable operations. Such an approach disables DVFS and thus negates potential energy savings.

In this paper, we examine the applicability of DVFS to processing workloads that represent packet forwarding tasks in NFV. We make use of a novel network-driven prediction engine to optimize the power consumption over and above what a regular DVFS governor implementation offered. Furthermore, the use of the prediction engine provides additional visibility in the operating mode of the server, thus creating an environment that makes it easier for operators to trust the technique in deployments. This paper is organized as follows: Section II presents the related work, Section III describes the approach and the solution, Section IV presents a discussion about the results, and Section V concludes the paper.

II. Related work

Dynamic Voltage and Frequency Scaling (DVFS) is a generic name for techniques that dynamically change the voltage and operating frequency of a processor in order to adapt the energy consumption to the momentary workload. Several authors proved it to be efficient when considering energy savings [6]–[8].

DVFS is supported at the operating system-level [9] to modify processors frequency and voltage to reduce power consumption of the system [6], [8]–[10]. Lower clock frequency and voltage of the processor naturally reduces system performance; therefore, a trade-off between performance and energy consumption needs to be taken into consideration when using this framework [7].

As one example, Intel processors implemented increasingly advanced CPU energy efficiency capabilities for several generations [11]–[14]. At the processor level, the energy efficiency capabilities are exposed by the Running Average

Power Limit (RAPL) interface, which included both the energy measurement and the control system [15].

The significance of the values provided by RAPL differs between Intel CPU generations as the estimate models of earlier generations were replaced by the actual measurements in newer ones. Nevertheless, RAPL was found the be accurate in providing power measurement values for the processor package [15], [16], while the memory-related values were found to be less accurate [16].

In the Linux operating system, these capabilities are manageable through the CPUFreq framework. In CPUFreq, the software component that evaluates the workload demands and acts to set the appropriate voltage and frequency is known as a governor. Several types of governors are predefined. User-defined policies allow specifying limit values for the parameters that could be set by the governors. The governors are: *performance* (sets frequency statically to the highest within the limits), *powersave* (sets frequency statically to the lowest within the limits), *userspace* (allows the user to set the frequency), and *ondemand* (sets the frequency within the limits according to CPU load) [17].

Xu *et al.* present in [18] a study on NFV, analyzing the energy efficiency of different NFV implementations based on packets arriving in the network. Significant energy usage differences were measured for data plane implementations using different libraries, as well as for different software implementations of the same virtual network function (an intrusion detection system). Small packets were found to consume more power than larger packets, because they required higher CPU utilization to be processed. This work illustrates the complexity involved in characterizing NFV workloads.

Controlling DVFS in response to network traffic was presented by Kuan *et al.* in [5]. They proposed a system that obtained packet inter-arrival rates and used the information in a DVFS governor. A similar solution was proposed and characterized in a virtual router environment by Yu *et al.* [19]. A model of the network congestion was used for predicting whether it would be possible to reduce the CPU frequency while maintaining an acceptable delay.

Elastic resource adaptation for NFV (ENVI) was proposed by Cao *et al.* in [20] to allow provisioning of resources according to workload dynamics. ENVI detected the need for scaling by using machine learning models trained with NFV information and resource utilization, but it did not optimize for energy consumption.

Marotta and Kassler [21] developed a solution for power-efficient robust NFV placement, optimizing both server and network power consumption by applying workload consolidation techniques. They used a relatively simple model where a NFV power consumption is determined by a quantity of generic resources requested by the NFV and the maximum variation of the resource utilization. The power consumption models for resources were considered to vary linearly with the workload.

Rossi *et al.* [22] verified that when threads are fairly distributed through the cores of a CPU, DVFS could provide a significant reduction in the energy consumption. However, while being able to save only about 10% energy, stock DVFS policies implemented in Linux increased response time of a web server by about 70%.

As evidenced in [18], virtual network functions exhibit a wide variation of CPU resource usage in relation to the bandwidth consumed. Simple techniques that only employ inter-packet arrival times cannot cope with such complexity. Techniques that are based on observing congestion indicators react only after the traffic was already impacted by a low power mode that was selected for the CPU. Workload consolidation solutions have good results in addressing diurnal variations in the traffic pattern, but exhibit too large overheads for adapting to fast time-wise variations. We seek a solution that could operate at the same tens of millisecond time scale as DVFS, yet be able to adapt to the traffic and to the virtual network function that is executed in the cloud platform.

III. Technical and System Description

A. Energy saving potential

As already mentioned, NFV clouds usually disable CPU energy efficiency features such as DVFS in order to ensure a predictable high-performance operating mode. In this subsection we take a closer look at the amount of energy that could be saved by enabling DVFS.

A server equipped with an Intel Xeon®CPU E3-1230 V2 @3.30GHZ was used for the experiments. The server had 16GB of RAM and was running Ubuntu 14.04.5 LTS as operating system; it had 8 CPUs when HyperThreading was enabled. A number of 15 different steps of frequencies were available to be controlled by DVFS, starting from 1.6 GHz to 3.3 GHz in 100Hz increments. The server also executed the open source Linux Kernel Virtual Machine (KVM) hypervisor. As an exemplary virtual network function, the well-known open source Snort intrusion detection system was chosen to be deployed within a VM. A traffic generator was deployed in another VM.

The energy consumed as reported by RAPL was measured for the case of a single server that contained both the NFV and the traffic generator. Also, the energy consumed was recorded for the server executing the NFV in the case when the traffic generator was executed on another server.

Fig. 1 and Fig. 2 present the energy consumption per frequency for a transmission rate of $9,000$ packets per second (pps). This particular value for the transmission rate was selected because it saturated the CPU when considering the lowest frequency (1.6 GHz). Therefore, Snort was always able to process all packets. The figures present the results in two scenarios: with a single server on the left and with two servers on the right. Fig. 1 shows energy measurements

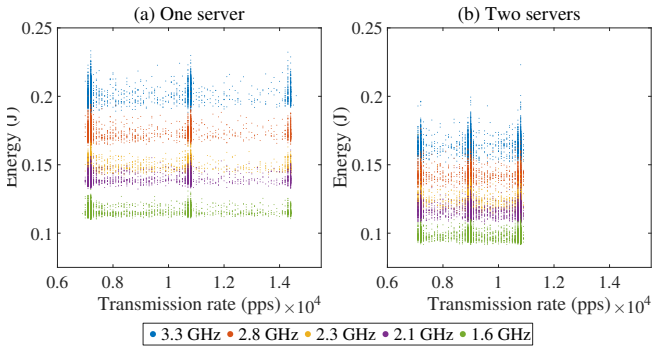

Figure 1. Energy per number of packets received when transmission rate = 9,000 pps for a single server (left) and for two servers (left).

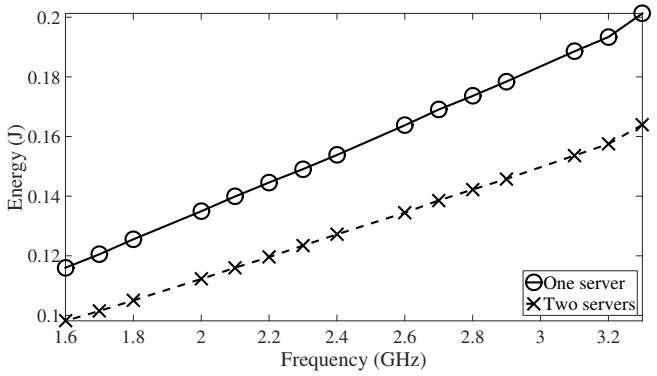

Figure 2. Average energy consumption per frequency when transmission rate is 9,000 pps.

points that represent a 10 ms time interval, while Fig. 2 indicates the average energy consumption per frequency. For readability purposes, in Fig. 1 we present results for only five of the measured frequencies, chosen such that they cover the entire operating frequency range of the CPU.

As expected based on the literature, the average energy consumption as shown in Fig. 2 increased almost linearly with the increase of the frequency even though the processor had to handle the same generic workload within each one of the scenarios. The value of the energy consumed for the highest frequency value of the CPU corresponds to the normal operating mode in an NFV deployment. Thus, this simple experiment showed that it would be possible to reduce the energy consumption by 42% or 39% for the same throughput served by the NFV.

B. Machine Learning

For planning purposes, developers of a NFV are usually able to specify the amount of resources required for the function to handle a certain traffic workload. However, this value is expressed as a number of CPUs and is used for dimensioning the system. Often, it represents the maximum quantity of resources that need to be reserved for the NFV. The momentary usage of the CPU differs for each packet received or forwarded, and also depends on the content of the packet and the processing code executed by the application. Thus, the relation between computing resources consumed and the traffic processed is non-linear for many NFV. Furthermore, different processor instructions consume different amounts of energy depending not only on the instructions themselves, but also on where the data needed for the execution resides. Determining the optimum energy performance mode is a problem that would be extremely difficult to approach analytically, making it an excellent candidate for the use of machine learning.

Machine learning based on Artificial Neural Networks (ANNs) was used for building an operational performance profile of a NFV. ANNs were shown to have high prediction ability with relatively low overhead, fitting in a dynamic real-time setting [23].

The features for characterizing the network traffic were recorded by accumulating packet statistics within a 10ms time window. The length of the time window was chosen arbitrarily, but it took into account a number of constraints including the ability to buffer excess packets that could not be handled due to the classifier determining a too low frequency compared to what would actually be needed for handling the traffic. It also attempted to keep under control the power consumption associated to the classification step itself, which meant that the number of times to execute it needed to be limited. Other considerations, such as transition delays between different CPU frequencies, played a minor role but were double-checked as to not be detrimental.

A training dataset was built by sending traffic repeatedly from the same traffic trace at a transmission rate that induced about 95% utilization on the processor core that executed the NFV and recording the energy consumed by the CPU package for each one of the available frequencies. This enabled us to estimate the throughput that could be supported in the testbed for each of the frequency. It also allowed for a small spare capacity in the system that could handle short-term overloads without holding too much traffic in buffers.

An initial offline learning process was executed using a network structure called Multi-Layer Perceptron (MLP). Our solution used a MLP with three layers, where the input layer corresponds to the traffic features, such as TCP, UDP, number of packets, the hidden layer is denoted by n_i, and the output layer indicates the frequency that was selected, as shown in Fig. 3.

C. System Description

The overall architecture of the system is presented in Fig. 4. The key components are the Network Module, the Machine Learning Model, the Prediction Engine Module, the DVFS Module and the Energy Consumption Module.

The Network Module was inserted in the network path between the network interface card and the NFV. It was

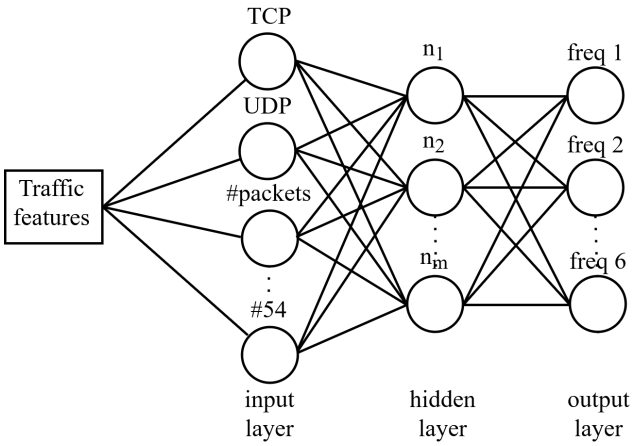

Figure 3. Neural Network structure for prediction.

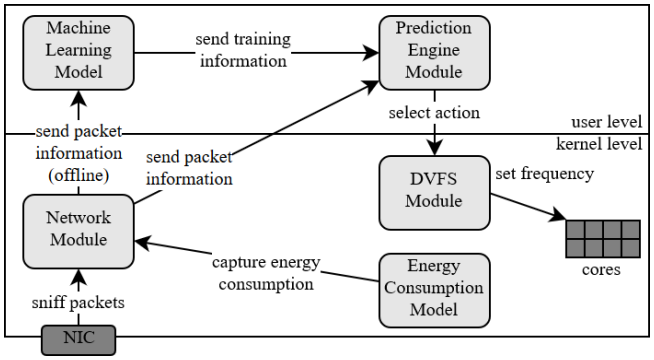

Figure 4. DVFS prediction engine architecture.

responsible for extracting features in real-time by examining the packets that were in transit towards the NFV.

The Prediction Module executed the classification algorithm based on the trained network and determined the frequency to be set on the processor.

The Frequency Setting (DVFS) Module is the component that actually configured the desired frequency in the processor registers, triggering the frequency change during real-time operation.

The Energy Consumption Module was active primarily during the data generation for the offline training phase. It accessed operating system counters to determine the energy consumed during the amount of time between two consecutive readings. This module is also active during the real-time measurements, but the data provided was used only for estimating the power consumption for the different methods that were evaluated. The Energy Consumption Module would not be active during normal operations in a real deployment.

The Machine Learning Module was active only during the offline training phase. It performed the training of the neural network, determining the weights to be used during the real-time operation.

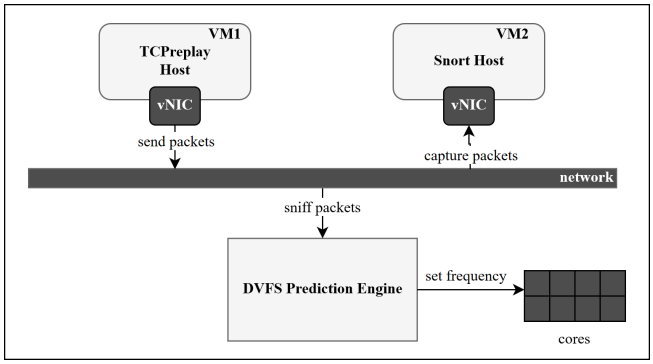

Figure 5. System implementation

D. Implementation

Fig. 5 presents the architecture of the system used for the measurements, where the DVFS Prediction Engine contains the architecture presented in Fig. 4.

The server hosted two Virtual Machines (VM), each one running on an exclusively-dedicated core. Snort was deployed as a virtual network function in the first VM. The second VM ran the traffic generator tool TCPreplay, which read the traffic from pcap files obtained from real traces and sent them towards Snort at a transmission rate that we could control through the command line parameters. We employed a trace without malicious activity provided by University of New Brunswick [24] that was previously employed in literature for Snort performance characterization.

The Network Module was implemented in C using the Pcap library [25]. We are aware of the significant overhead introduced by this implementation choice. However, it provided a familiar API that allowed parsing the content of incoming packets, selecting only those addressed to the Snort instance and extracting a number of 53 features corresponding to different protocols present in the packets up to Layer 7 in the OSI stack.

The Energy Consumption Module read energy consumption and CPU cycles values from the MSR registers of the Intel processor via the RAPL interface. These values, together with the traffic features, were presented as input to the Prediction Module.

The Prediction Module was implemented in Python using the well-known scikit-learn library [26]. The Prediction Engine Module with neural network consisted of three layers with 20 neurons in the hidden layer nodes is used for the training.

The frequency value determined after the classification step, represented as a numerical lookup key value, was then transmitted to the DVFS module. This last module, also implemented in C, determined the correct frequency by performing a lookup in a pre-configured table and configured

the frequency value in the CPU using the interface of the *userspace* Linux power capping governor.

A number of 5300 data samples in the training stage for each frequency, out of which 5000 data samples were used for training, and 300, for the validation step during offline learning process. The samples were measured at 10 ms interval. During the validation step, the accuracy module from scikit-learn was used in order to determine the training precision. To this means, the prediction engine analyzed the features from the data set and the predicted frequency was then compared to the real value. Consequently, the accuracy represents the average of times in which the frequency was well predicted.

IV. Results

This section presents the measurement results considering the setup presented in Section III-D. At first, we discuss the data obtained offline and used for training purpose. Following that, we show the results from measurements on real-time traffic, comparing the solution proposed in this paper to the energy consumed at the maximum frequency and to the *ondemand* governor from the Linux kernel that attempted to dynamically adapt the power consumption to the CPU workload.

A. Training

Fig. 6 and Fig. 7 present, respectively, the energy consumed, measured in joules and the number of CPU cycles as a function of the transmission rate in which packets arrive for each frequency. In order to facilitate the visualization of the graphics, only five out of eight measured frequencies are shown. As expected, the frequency increase leads to a higher number of cycles that the processor execute per time unit and thus a higher power consumption. Moreover, Fig. 6 indicates that it is possible to save more than 40% of energy in some cases, as the lowest frequency could be used when transmission rate is 15,000 pps rather than the highest one, consuming approximately 0.12 J instead of 0.21 J.

During initial experiments using all the sixteen available frequencies, we observed that many neighboring frequencies overlapped others in terms of energy consumption and of number of cycles. The use of this data for training the machine learning algorithms could introduce errors in the prediction engine as it would not be able to distinguish the frequencies that have similar energy consumption. Indeed, we verified that after training the Machine Learning Module with 20 neurons the accuracy was only 63%, which we considered unsuitable for the system.

To overcome this problem, we selected a subset of eight frequencies that were further apart. Using this data to train the machine learning algorithms, it was possible to achieve an accuracy of 79% on the Machine Learning Module with 20 neurons, and 81% with 200 neurons. While this value may seem low, we considered it to be satisfactory for the system. An incorrect prediction would have only minor effects on the traffic (in the worst case introducing some additional latency) due to the spare compute capacity built into the system through the labeled training set. The small time window allows for rapid correction of an erroneous prediction in one of the next time intervals.

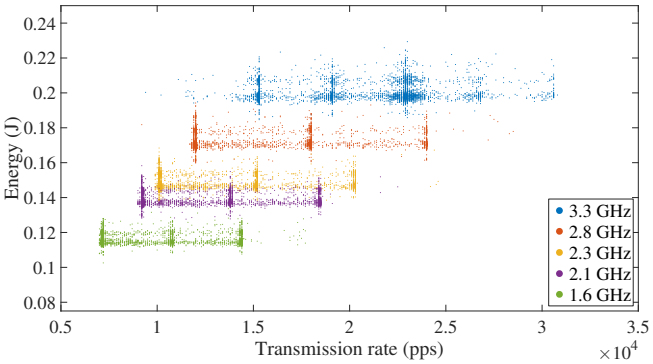

Figure 6. Energy per number of packets received for five frequencies.

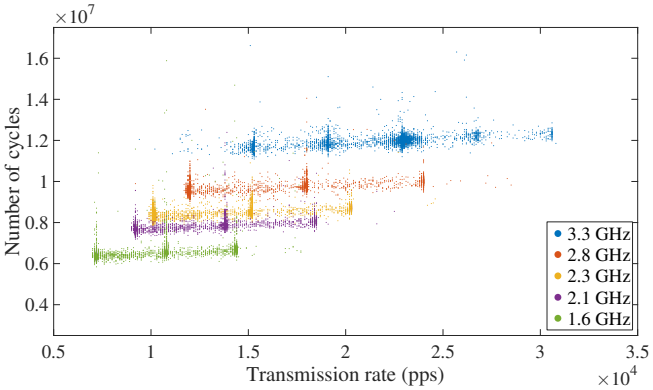

Figure 7. Cycles per number of packets received for five frequencies.

B. Real-time

As mentioned in Section IV-A, when trained with 200 neurons, the system gained a small amount of accuracy when compared to using only 20 neurons. Theoretically, it would thus be possible to predict the correct frequency more often and to have a greater reduction in the energy consumption. However, this hypothesis was invalidated by measurements as shown in Fig. 8, presenting the energy consumption as a function of transmission rate for a DVFS Prediction Engine using 20 and 200 neurons.

This result is a consequence of the neural network computational complexity. Although it could predict slightly more accurately the frequency for a certain number of packets, the increased computational complexity lead to greater energy consumption. Therefore, the neural network with 20 neurons provided better overall energy savings even though it had lower accuracy.

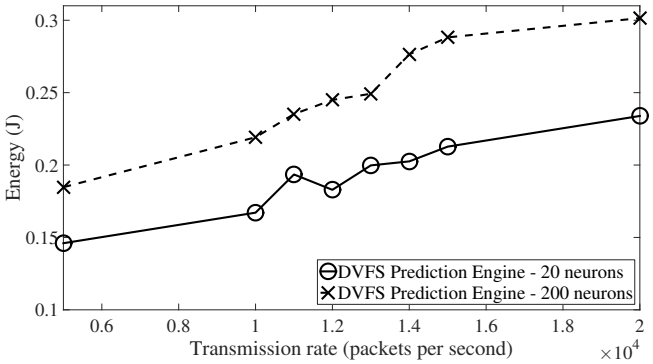

Figure 8. Energy consumption as a function of transmission rate for 20 and 200 neurons.

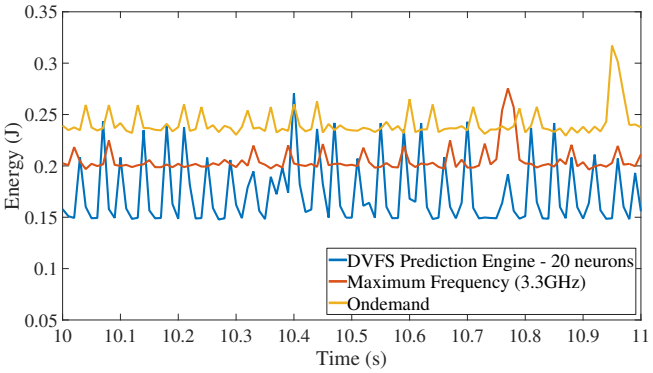

Figure 9. Energy consumed in time with transmission rate $10,000$ pps.

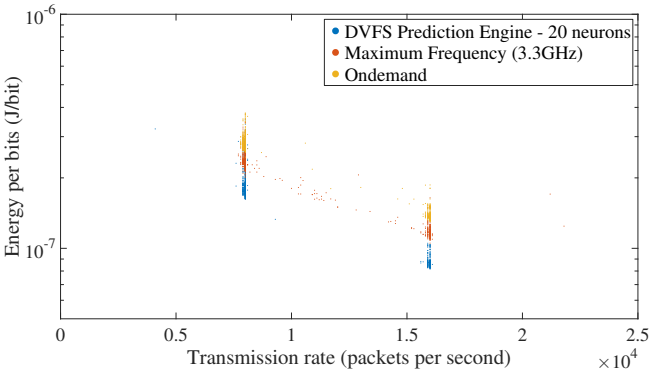

Figure 10. Energy efficiency as a function of bytes received with transmission rate $10,000$ pps.

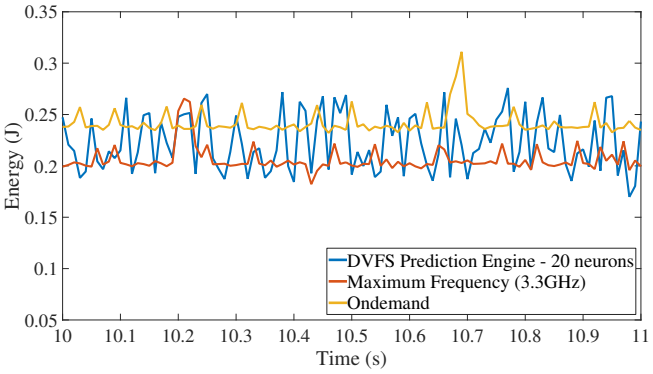

Figure 11. Energy consumed in time with transmission rate $15,000$ pps.

A comparison between the DVFS prediction engine versus setting the maximum frequency or using the *ondemand* governor is shown in Fig. 9 and Fig. 10. In order to generate a workload in the Snort Host, the average packet transmission rate was chosen to be $10,000$ pps in the TCPreplay Host, larger than what could be handle by the NFV when simply setting the lowest possible frequency on the CPU.

In order to understand how the energy consumption changes in each scenario, Fig. 9 shows a one-second window measurements as a function of time. The use of the Prediction Engine Module presents the lowest energy consumption in average, representing a reduction of 15% with respect to the maximum frequency, and more than 27% when compared to the *ondemand* governor. However, the variation on the amount of power used by DVFS Prediction Engine is higher than the other two measurements, as it fluctuates between 0.15 J and more than 0.2 J in every measurement. Therefore, it presents the highest standard deviation, of 0.032 J against 0.014 J for both *ondemand* and maximum frequency scenarios. This result is a consequence of the Prediction Engine Module changing the frequency at a high rate.

Fig. 10 presents the energy efficiency per packet for the three measurement scenarios. We defined the energy efficiency as the amount of energy needed to process the packets in transit, expressed as energy per bit (J/bit). The scenario using the DVFS Prediction Engine has the best energy efficiency, implying that it consumes less energy for the same amount of traffic compared to setting the frequency as the maximum or the governor as *ondemand*.

Fig. 11 and Fig. 12 present the results when transmission rate is $15,000$ pps. Fig. 11 shows the energy consumption in time, and Fig. 12 presents the energy efficiency of all three measurements.

As predicted from the outcomes of Fig. 8, the energy consumption of the Prediction Engine Module increases at greater transmission rates, with an average of 0.221 J against 0.174 J. Moreover, this value surpasses the energy consumption of the case when the CPU operates at the maximum frequency by more than 5%. These results imply that a reduction on the energy efficiency is caused by the necessity to operate in higher frequencies combined with the CPU usage caused by the DVFS Prediction Engine.

As expected, the algorithm ceases to provide a reduction in energy consumption at high packet transmission rates. Such transmission rates already require the use of the highest frequency of the CPU. Also, the overhead introduced by the neural network and packet inspection could no longer be compensated by the energy consumption difference between

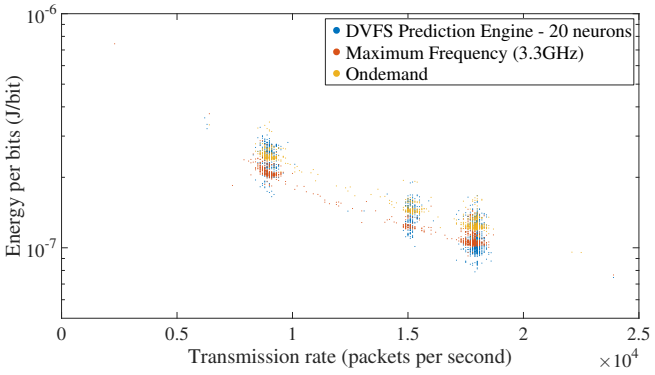

Figure 12. Energy efficiency as a function of bytes received with transmission rate $15,000$ pps.

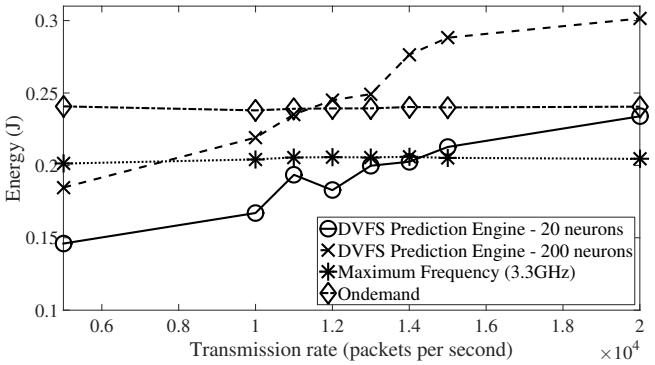

Figure 13. Energy consumption of *ondemand*, maximum frequency and machine learning measurements as a function of the rate.

two consecutive frequencies.

Fig. 13 presents the energy variation according to the transmission rate for four scenarios: DVFS Prediction Engine with 20 and with 200 neurons, maximum frequency and *ondemand* governor. As expected, the energy consumption in the maximum frequency and *ondemand* scenarios are rather constant. The use of DVFS Prediction Engine, on the other hand, causes a raise on energy consumption in some cases due to the computational overload introduced by the neural network computation. Indeed, the DVFS Prediction Engine curves surpass the maximum frequency when the transmission rate was over $14,000$ pps and $8,000$ pps for 20 and 200 neurons, respectively. We demonstrated practically that when applying machine learning to increase energy efficiency, the computational overhead incurred by the method matters. In particular when the smaller network was used, our method was able to provide better results in an NFV cloud compared to both the industry best practice and to the widely available open source alternative.

V. Final Considerations and Future Work

We presented a method for saving energy in NFV clouds when NFV is deployed on general-purpose compute platforms. Our method employs a pre-trained machine learning model to dynamically determine the best CPU frequency to be used for processing packets. Generating the labeled training dataset required minimal expertise and could be easily adapted to other NFV implementations that are heavily biased towards CPU processing. Even with a largely unoptimized implementation, the viability of the solution was proven by the ability of saving 27% of the energy consumed compared with the standard *ondemand* power capping governor in the Linux operating system.

Several aspects of the work require further investigations. In many scenarios, NFV will employ technology such as Intel DPDK that uses 100% of the CPU cores allocated, regardless on whether traffic is present or not. As shown in our measurements in Fig. 6, an opportunity for saving more than 40% of the consumed energy exists simply by adapting the frequency of these CPU cores to the incoming traffic rate. More in-depth analysis with other types of virtual network functions, which make heavy use of storage resources or are memory-bound in the execution, could also be explored. Newer generations of processors enable controlling the frequency per-core for each one of the multiple cores of the chip. Another possible avenue for investigation would be to execute different virtual network functions within the same server and evaluate how the energy savings would follow the different momentary workloads, and whether large differences in terms of frequency between the cores may have any negative effects on the overall performance. Finally, other machine learning algorithms could be deployed, such as Support Vector Machine, to compare their performance.

Acknowledgment

This work was partially supported by CNPq (Brazil) (process 132234/2017-3) and by Innovation Center, Ericsson Telecomunicações S.A., Brazil.

References

[1] Ericsson, "Ericsson mobility report," Ericsson, Sweden, Tech. Rep. Rep. EAB-17, vol. 005964 Uen, Revision B, June 2017. [Online]. Available: https://www.ericsson.com/assets/local/mobility-report/documents/2017/ericsson-mobility-report-june-2017.pdf

[2] L. Peterson, A. Al-Shabibi, T. Anshutz, S. Baker, A. Bavier, S. Das, J. Hart, G. Palukar, and W. Snow, "Central office re-architected as a data center," *IEEE Communications Magazine*, vol. 54, no. 10, pp. 96–101, October 2016.

[3] L. Peterson, A. Al-Shabibi, T. Anshutz, S. Baker, A. Bavier, S. Das, J. Hart, G. Parulkar, and W. Snow, "Central office re-architected as a data center," *IEEE Communications Magazine*, vol. 54, no. 10, pp. 96–101, 2016.

[4] J. Whitney and P. Delforge, "Data center efficiency assessment - scaling up energy efficiency across the data center industry evaluating key drivers and barriers," The Natural Resource Defence Council (NRDC), Tech. Rep. 14-08-A, August 2014.

[5] J. Kuang, L. Bhuyan, and R. Klefstad, "Traffic-aware power optimization for network applications on multicore servers," in *DAC Design Automation Conference 2012*, June 2012, pp. 1006–1011.

[6] K. H. Kim, R. Buyya, and J. Kim, "Power aware scheduling of bag-of-tasks applications with deadline constraints on DVS-enabled clusters," in *Seventh IEEE International Symposium on Cluster Computing and the Grid (CCGrid '07)*, May 2007, pp. 541–548.

[7] Y. C. Lee and A. Y. Zomaya, "Energy conscious scheduling for distributed computing systems under different operating conditions," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 8, pp. 1374–1381, August 2011.

[8] N. B. Rizvandi, J. Taheri, and A. Y. Zomaya, "Some observations on optimal frequency selection in dvfs-based energy consumption minimization," *Journal of Parallel and Distributed Computing*, vol. 71, no. 8, pp. 1154–1164, 2011. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0743731511000165

[9] S. Saha and B. Ravindran, "An experimental evaluation of real-time DVFS scheduling algorithms," in *Proceedings of the 5th Annual International Systems and Storage Conference*, ser. SYSTOR '12. New York, NY, USA: ACM, 2012, pp. 1–12.

[10] E. Le Sueur and G. Heiser, "Dynamic voltage and frequency scaling: The laws of diminishing returns," in *Proceedings of the 2010 International Conference on Power Aware Computing and Systems*, ser. HotPower'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 1–8.

[11] E. Rotem, A. Naveh, A. Ananthakrishnan, E. Weissmann, and D. Rajwan, "Power-management architecture of the Intel microarchitecture code-named sandy bridge," in *IEEE Micro*, vol. 32, no. 2, March 2012, pp. 20–27.

[12] S. Rusu, H. Muljono, D. Ayers, S. Tam, W. Chen, A. Martin, S. Li, S. Vora, R. Varada, and E. Wang, "A 22 nm 15-core enterprise Xeon® processor family," *IEEE Journal of Solid-State Circuits*, vol. 50, no. 1, pp. 35–48, January 2015.

[13] A. Varma, B. Bowhill, J. Crop, C. Gough, B. Griffith, D. Kingsley, and K. Sistla, "Power management in the Intel Xeon E5 v3," in *2015 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, July 2015, pp. 371–376.

[14] E. Rotem, U. C. Weiser, A. Mendelson, R. Ginosar, E. Weissmann, and Y. Aizik, "H-earth: Heterogeneous multicore platform energy management," *Computer*, vol. 49, no. 10, pp. 47–55, October 2016.

[15] H. Zhang and H. Hoffman, "A quantitative evaluation of the rapl power control system," *Feedback Computing*, 2015.

[16] S. Desrochers, C. Paradis, and V. M. Weaver, "A validation of DRAM RAPL power measurements," in *Proceedings of the Second International Symposium on Memory Systems*, ser. MEMSYS '16. New York, NY, USA: ACM, 2016, pp. 455–470.

[17] D. Brodowski, "CPU frequency and voltage scaling code in the Linux(tm) kernel. linux cpufreq user guide," April 2017. [Online]. Available: https://www.kernel.org/doc/Documentation/cpu-freq/user-guide.txt

[18] Z. Xu, F. Liu, T. Wang, and H. Xu, "Demystifying the energy efficiency of network function virtualization," in *2016 IEEEACM 24th International Symposium on Quality of Service (IWQoS)*, June 2016, pp. 1–10.

[19] Q. Yu, T. Znati, and W. Yang, "Energy-efficient, delay-aware packet scheduling in high-speed networks," in *2015 IEEE 34th International Performance Computing and Communications Conference (IPCCC)*, December 2015, pp. 1–8.

[20] L. Cao, P. Sharma, S. Fahmy, and V. Saxena, "ENVI elastic resource flexing for network function virtualization," in *9th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 17)*. Santa Clara, CA: USENIX Association, 2017. [Online]. Available: https://www.usenix.org/conference/hotcloud17/program/presentation/cao

[21] A. Marotta and A. Kassler, "A power efficient and robust virtual network functions placement problem," in *2016 28th International Teletraffic Congress (ITC 28)*, vol. 01, September 2016, pp. 331–339.

[22] F. D. Rossi, M. G. Xavier, E. D. Conte, T. Ferreto, and C. A. F. D. Rose, "Green software development for multicore architectures," in *2014 IEEE Symposium on Computers and Communications (ISCC)*, June 2014, pp. 1–6.

[23] E. I. Nehru, B. Venkatalakshmi, R. Balalcrishnant, and R. Nithya, "Neural load prediction technique for power optimization in cloud management system," in *2013 IEEE Conference on Information Communication Technologies*, April 2013, pp. 541–544.

[24] A. Shiravi, H. Shiravi, M. Tavallaee, and A. A. Ghorbani, "Toward developing a systematic approach to generate benchmark datasets for intrusion detection," *Computers and Security*, vol. 31, no. 3, pp. 357–374, 2012.

[25] L. M. Garcia, "Programming with libpcap – sniffing the network from our own application," *Hakin9 Magazine*, vol. 3, no. 2, pp. 38–46, 2008.

[26] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, October 2011.