Model-based Analytics for Profiling Workloads in Virtual Network Functions

Roberto Bruschi[2], Franco Davoli[1,2], Paolo Lago[1,2] and Jane Frances Pajo[1,2]
[1]DITEN – University of Genoa, Genoa, Italy
[2]CNIT – Research Unit of the University of Genoa, Genoa, Italy
roberto.bruschi@cnit.it, franco.davoli@unige.it, {paolo | jane.pajo}@tnt-lab.unige.it

***Abstract*—With the flexibility and programmability levels offered by Network Functions Virtualization (NFV), it is expected to catalyze the upcoming "softwarization" of the network through software implementation of networking functionalities on virtual machines (VMs). While looking into the different issues thrown at NFV, numerous works have demonstrated how performance, power consumption and, consequently, the optimal resource configuration and VM allocation vary with the statistical features of the workload - specifically, the "burstiness" of the traffic. This paper proposes a model-based analytics approach for profiling (virtual) network function (VNF) workloads that captures traffic burstiness, considering - and adding value to - hardware/software performance monitor counters (PMCs) available in Linux host servers. Results show good estimation accuracies for the chosen PMCs, which can be useful to enhance current methods for fine-grained provisioning, usage-based pricing and anomaly detection, and facilitate the way towards an agile network.**

I. Introduction

The growing trend towards network "softwarization" has put the limelight on commodity hardware (i.e., industry standard high volume servers, switches and storage) and the Virtualization technology. For instance, with Network Functions Virtualization (NFV), networking functionalities are envisioned to run as software on commodity hardware [1], which can either be in the Cloud or Fog computing domains. Besides the various advantages offered by such softwarization solutions, some issues on performance, energy efficiency and the ensuing management complexity must be addressed to justify their viability.

Commodity hardware inherently provides much lower performance and energy efficiency in contrast to the special-purpose hardware populating the majority of today's networks. Although power management techniques (i.e., Low Power Idle (LPI) and Adaptive Rate (AR)) have become widely available through the Advanced Configuration and Power Interface (ACPI) specification [2], the power saving promise comes with performance degradation [3] or even negative savings if used naively [4]. On top of that, the additional processing delays due to the virtualization overhead when implementing (virtual) network functions (VNFs) as virtual machines (VMs) further lower the performance, consuming even more energy than their physical counterparts [5].

Moreover, this shift towards an extremely modular and virtual network architecture - that could potentially provide the service "agility" required by tomorrow's demands - calls for automated resource configuration and provisioning mechanisms to cope with the resulting complexity of network/service management. The ETSI NFV Management and Orchestration (NFV-MANO) framework [6] is stipulated for this purpose, designating the virtual infrastructure manager (VIM) for the management and control of resources in an NFV infrastructure based on their capacity/usage and fault/event notifications - information that generally does not directly expose key performance indexes in the network.

In today's research scene, numerous studies have been devoted to understanding and/or addressing the aforementioned concerns. Among others, [4] proposes dynamic idle period prediction for intelligent sleeping state entry, [7] models the power consumption and performance of commodity hardware based on renewal theory principles, optimizing their trade-off, [8] studies and models the virtualization overhead, [9] models VNF performance using discrete-time analysis, and [10] tries to incorporate available power management techniques in VM consolidation. Most of these works demonstrate how performance, power consumption and, consequently, the optimal resource configuration and VM allocation vary with the statistical features of the network workload - specifically, the "burstiness" of the traffic. A typical approach to capture traffic burstiness involves analyzing packet-level traces as in [7], which may be unsuitable for profiling highly dynamic workloads. This, together with the long-held dream of fine-grained provisioning and usage-based pricing, as well as anomaly detection, drive the need for workload profiling on the fly.

To this end, we propose a model-based analytics approach for profiling workloads in VNFs using - and adding value to - available hardware/software performance monitor counters (PMCs) in Linux host servers that could potentially enhance the operation of the VIM. In particular, this paper evaluates different PMCs to choose the most suitable ones for measuring/estimating the offered load, utilization (due to the network workload), batch arrival rate and average batch size, comparing different black-box approaches with existing models.

The remainder of this paper is organized as follows. Firstly, the system under test (SUT) is described in Section II. The proposed approach is then discussed in Section III, giving details on the model, the PMCs and the estimation process. Experimental results are then presented in Section IV, and finally, conclusions are drawn in Section V.

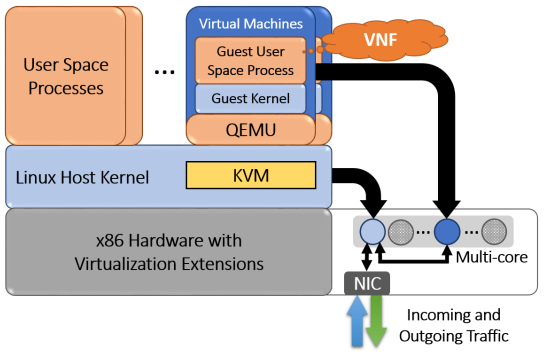


Fig. 1: Overview of the system.

II. System Description

As virtualization extensions are made available in x86 hardware, Kernel-based Virtual Machine (KVM) [11] - a full virtualization solution for Linux - has gained much popularity for its simplicity and ability to run VMs with unmodified guest operating systems (OSs). KVM uses the Linux kernel as bare-metal hypervisor and has been integrated in the kernel since the 2.6.20 release, making it the default virtualization mechanism recommended for most Linux distributions [12]. In this respect, KVM virtualization is the basis of this work, but the approach can be also applied to other platforms.

Fig. 1 shows the SUT mapped to the KVM architecture. The user space process Quick Emulator (QEMU) [13] implements the guest networking of KVM, providing an emulated network interface card (NIC) to the guest OS - a detailed description of the network I/O path can be found in [14]. In traditional VM implementation of VNFs, the network function runs as a guest user space process, as indicated in Fig. 1, giving a rough idea on how much overhead is introduced by virtualization. Moreover, a separate core is allocated to the VM process to ensure its isolation from other processes in the monitoring - this is done by setting the VM process's CPU affinity.

Like most commodity hardware today, the multi-core server used is equipped with power management mechanisms via the ACPI. It models the LPI and AR functionalities through the power and performance states ($C-$ and $P-$ states, respectively) accessible at the software level. With these energy-aware states, idle cores can go to low power sleeping states, indicated by C_x, $x \in \{1, \ldots, X\}$, while at the active state C_0, a core can work at different performance states, indicated by P_y, $y \in \{0, \ldots, Y\}$. As the indexes x and y increase, the lesser the power consumption - but performance is degraded due to increased latencies (i.e., due to wakeup times and increased service times, respectively). More on this power/performance trade-off optimization can be found in [7] and [10].

In addition, the `ethtool` command [15] is used to set the interrupt coalescing and RX/TX ring parameters in the NIC such that the incoming/outgoing traffic to/from the core are left almost as they are. Particularly, the options for adaptive interrupt coalescence (i.e., `adaptive-rx` and `adaptive-tx`) are disabled, and the parameters defining the waiting times before raising RX/TX interrupts (i.e., `rx-usecs`, `rx-frames`, `tx-usecs` and `tx-frames`) are set to 0. When interrupts are disabled, the status is updated after a packet is received or transmitted (i.e., `rx-frames-irq` and `tx-frames-irq` set to 1). Although these settings somehow generate hardware interrupts that match the load, it is important to note that the utilization overhead due to the interrupts can become significantly high in heavy load conditions. In such a case, traffic shaping via interrupt moderation can be desirable - this requires optimizing the trade-off between the overhead due to interrupts and end-to-end latency. On the other hand, setting the `rx` and `tx` buffer sizes of the RX/TX rings to the preset maximums (i.e. 4096) ensures that the NIC's capability to handle burst arrivals is maximized.

III. Model-based Analytics

In this first take on model-based analytics for profiling workloads in VNFs, we suppose a one-to-one correspondence between VMs and cores for simplicity (i.e., each VM is allocated one virtual CPU (vCPU) and each core runs one VM). Consequently, the core workload and utilization are assumed to conform with those of the VNF.

Here we recall a model proposed in [10] for an energy-aware core running VMs, which becomes the basis for comparison of the PMC-based measurements and estimates. Then, the set of PMCs considered and their roles in the analytics will be presented.

A. Model and Proposed Approach

We model the core running the VM as a M^X/G/1/SET queue [16], a generalization of the well-known M^X/G/1 queue [17] that takes into account a setup period SET. This model not only captures traffic burstiness, but more importantly, SET captures the operations required between idle/busy transitions (e.g., core on/off transitions and reconfiguration, context switches, etc.).

It has been established in [18] and [19] that traffic behaviour in telecommunications networks can be effectively modeled by using the Batch Markov Arrival Process (BMAP), where packets arrive in batches rather than singly. In more detail, batches of random size X arrive at the core at exponentially distributed inter-arrival times with batch arrival rate λ and average batch size β, defining the offered load $OL = \lambda\beta$ packets per second (pps). A batch of customers that arrives at an empty system initiates the setup period $SET = \tau_{wu} + \tau_{cs}$ required before service can be resumed, where τ_{wu} and τ_{cs} are the setup components corresponding to the CPU wakeup times (which depends on the $C-$ state) and context switching (which depends on the $P-$ state), respectively. SET is supposed to be deterministic for a given (C_x, P_y) pair. Then, when the setup is finished, exhaustive service begins. Packets are served individually with generally distributed service times, at an average rate μ (pps). The core utilization can be expressed as:

$$\rho = \frac{\lambda\beta}{\mu} \qquad (1)$$

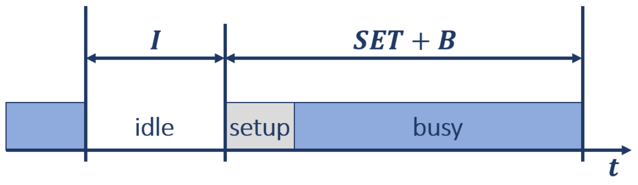


Fig. 2: Generic renewal cycle of a M^X/G/1/SET queue.

which is assumed to be less than 1 for system stability.

Based on classical renewal theory principles, independent and identically distributed (iid) "cycles" of alternating idle and delay busy (i.e., actual busy period plus the setup) periods can be identified, as shown in Fig. 2. From this perspective, the core utilization can be expressed in terms of the renewal cycle components:

$$\rho = \frac{E\{B\}}{E\{B\} + E\{I\} + E\{SET\}} \tag{2}$$

where $E\{B\}$, $E\{I\}$ and $E\{SET\}$ are the expectations of the actual busy period, idle period and setup period, respectively.

Suppose that μ is given for a certain VNF and (C_x, P_y) pair; we want to measure ρ and estimate λ. An estimation of OL can be provided, as well. Then, β can be estimated by inverting the utilization equation or, alternatively, OL.

$$\hat{\beta} = \begin{cases} \rho\mu/\hat{\lambda} \\ \hat{OL}/\hat{\lambda} & \text{if } OL \text{ can be estimated} \end{cases} \tag{3}$$

In the latter case, the utilization can also be estimated as:

$$\hat{\rho} = \hat{OL}/\mu \tag{4}$$

B. Performance Monitor Counters

Different utilities for Linux performance monitoring are evaluated to find the most suitable PMCs in obtaining the aforementioned model parameters. In this work, the following commands/tools have been chosen for the black-box measurements/estimations. The terms 'CPU' and 'core' will be used interchangeably hereinafter (note that in the case of hyper-threading, 'CPU' may also refer to a logical core).

1) Mpstat command: The `mpstat` command [20] comes with the sysstat package and it reports utilization and interrupts statistics per CPU.

The `-u` option reports the CPU utilization with a percentage breakdown of the time spent in user space (with/without nice priority), kernel space (excluding the time spent in servicing interrupts), servicing hardware and software interrupts, idle (with/without outstanding disk I/O requests), running a virtual processor, among others. For the interrupts, the `-I ALL` option is able to report both hardware and software interrupts received per second according to those listed in /proc/interrupts and /proc/softirqs files, respectively. To report either one of them, `CPU` or `SCPU` is used in place of the keyword `ALL`. To specify a CPU to be analyzed, the `-P <cpu#>` option is used, otherwise `-P ALL`. Lastly, the `<interval>` and `<count>` parameters are used to set the time in seconds between each report and the number of reports to be generated, respectively.

2) Idlestat tool: Idlestat [21] is a tool useful for CPU power/performance state analysis. In its trace mode, the `idlestat` command monitors and captures $C-$ and $P-$ state transitions of CPUs, as well as raised interrupts, over a user-defined interval. To achieve this, it relies on the kernel's `ftrace` function, which requires root privilege.

The `--trace` option is used to run `idlestat` in trace mode, which comes with the `-f` and `-t` parameters specifying the trace output filename and the capture interval in seconds, respectively. Moreover, the `-c` and `-p` options are used to report $C-$ (including $POLL$ - the state in which the CPU is idle but did not enter a sleeping state) and $P-$ states statistics, displaying the minimum time, maximum time, average time and total time spent in each state per CPU. For the interrupts, `-w` is used to report the number of times a particular interrupt caused a core to wake up. Any combination of `-c`, `-p` and `-w` can be used in running an idlestat trace.

3) Perf tool: `perf stat` [22] is a command included in the kernel-based performance analysis tool perf, which gathers performance counter statistics of processes, threads or CPUs (system-wide/per-CPU).

Statistics of specific events can be obtained by using the option `-e`; the `perf list` command [23] can be run to get a list of all available events, categorized according to their type (i.e., hardware event, software event, hardware cache event, tracepoint event, etc.). The options `-p <pid>` and `-t <tid>` are used to specify a process or thread to be analyzed, while `-C <cpu#>` is used to specify a CPU and `-a` (`-A`) to obtain system-wide statistics from all CPUs with (without) count aggregation.

The Linux `sleep` command with the `<number>` parameter can be appended to the `perf stat` command to specify an interval in seconds for gathering statistics. The actual time elapsed Δt is displayed together with the counts.

C. Estimation with PMCs

For a given (C_x, P_y) pair, OL, ρ, λ and, consequently, β can be measured/estimated using the commands/tools described in the previous section.

1) Offered load: In this work, we use the total number of software interrupts received per second by the CPU, obtained with the `mpstat` command, to estimate OL. Although this already gives good accuracy levels, looking into the breakdown may allow for further improvement - particularly, the rate at which software interrupts are raised for packet reception events, NET_RX/s.

2) Utilization: As regards the utilization, both `mpstat` and `idlestat` commands provide performance counters for its measurement.

With the former, utilization is given by:

$$\rho_{meas,1} = 1 - \%iowait - \%idle \tag{5}$$

where $\%iowait$ and $\%idle$ are the percentage of time spent in idle with and without outstanding disk I/O request, respectively. On the other hand, the latter command provides utilization as:

$$\rho_{meas,2} = \frac{T_{P_y}}{T_{P_y} + T_{C_x} + T_{POLL}} \tag{6}$$

where T_{P_y}, T_{C_x} and T_{POLL} correspond to the average duration spent in P_y, C_x and $POLL$ states, respectively.

Both of these measurement approaches give approximately the same results ($\approx \rho_{meas}$) - however, in contrast to the model, it was observed that the measured busy period and, consequently, the utilization include the setup period. To correct this, we use the deterministic setup time as follows.

$$\rho_{corr} = \rho_{meas} - \frac{\tau_{wu} + \tau_{cs}}{T_{P_y} + T_{C_x} + T_{POLL}} \tag{7}$$

Lastly, since we were able to estimate OL in Sub-section III-C1, the utilization can also be estimated as in (4).

3) Batch arrival rate: Here we compare different approaches for estimating λ based on performance counters obtained with the `idlestat` and `perf stat` commands.

As presented in [24], the average idle period of a M^X/G/1/SET queueing system is simply: $E\{I\} = 1/\lambda$. Starting from this, we first estimate the batch arrival rate from T_{C_x} and T_{POLL} obtained with the `idlestat` command.

$$\hat{\lambda_1} = \frac{1}{T_{C_x} + T_{POLL}} \tag{8}$$

With the `perf stat` command, there are two events that can be useful for estimating λ - `kvm:kvm_vcpu_wakeup` and `kvm:kvm_ack_irq`. With the former, we obtain the number of vCPU wakeup events within the elapsed time Δt, estimating the batch arrival rate as:

$$\hat{\lambda_2} = \frac{\text{\# of } \texttt{kvm:kvm\_vcpu\_wakeup} \text{ events}}{\Delta t} \tag{9}$$

The latter, on the other hand, gives the number of interrupt controller interrupt acknowledgments, aggregating both RX and TX interrupts, within the elapsed time Δt. With this, we suppose that only half of the count is due to packet reception, estimating the batch arrival rate as:

$$\hat{\lambda_3} = \frac{\text{\# of } \texttt{kvm:kvm\_ack\_irq} \text{ events}}{2\Delta t} \tag{10}$$

4) Average batch size: Once we have evaluated the most suitable approach for estimating λ, the average batch size is estimated as in (3), using both measured and corrected utilizations, as well as with $\hat{OL}$.

IV. Experimental Results

To evaluate the proposed model-based analytics approach, we consider a Linux server equipped with two Intel® Xeon® E5-2643 v3 3.40GHz processors [25]. The SUT is connected via RX/TX Gigabit Ethernet links to an Ixia NX2 router tester - the testbed component used for generating artificial traffic (64-byte Ethernet frames are considered in this evaluation). As regards the VNF, OpenWrt [26] is run on a VM and configured as a virtual firewall (VF).

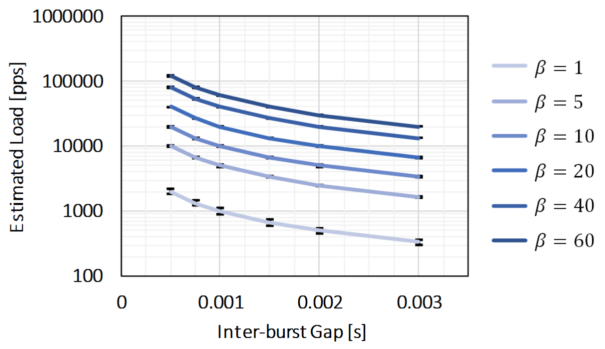


Fig. 3: Average offered load estimates and error bounds.

The ACPI configuration of the SUT is set to (C_1, P_0) to maximize the system throughput, where the C_1 power state corresponds to the lightest sleeping state, while the P_0 performance state to the maximum core frequency. For this configuration, the average service rate μ of the VF is assumed to be the maximum throughput of the system. We further suppose that the wakeup time from C_1 is $\tau_{wu} \approx 1\mu s$ (for the Sandy Bridge EP platform [27]) and the setup component due context switching $\tau_{cs} \approx 30\mu s$ (the proposed rule of thumb for real-world scenarios in [9]).

In this evaluation, traffic is generated from the router tester at varying inter-burst gaps (in seconds) and batch sizes (in packets), while ensuring that $OL < \mu$ for system stability. Particularly, we consider $1/\lambda \in \{0.0005, 0.00075, 0.001, 0.0015, 0.002, 0.003\}$ and $\beta \in \{1, 5, 10, 20, 40, 60\}$. Multiple runs (namely, 10) are performed for each (λ, β) combination to get a grasp on the reliability of the results.

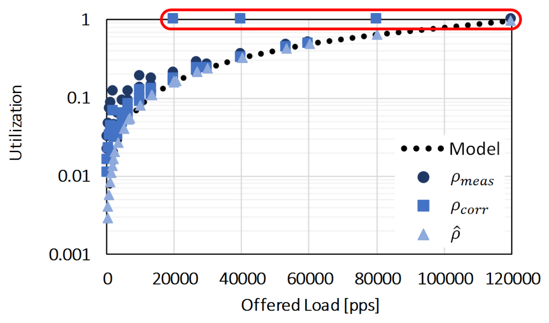
A. Offered Load

Fig. 3 shows the averages of obtained estimates for varying values of $1/\lambda$ and β. Error bars are used to indicate the ceilings of the maximum error percentage for each β value.

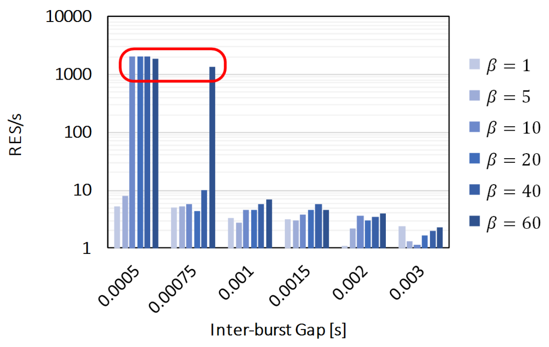
The worst estimate was obtained for $\beta = 1$ (i.e., $\%err = 8.84\%$). Then, the estimation accuracy gradually improves as traffic smooths out, with $\%err < 3\%$ for $\beta \in \{5, 10\}$ and $\%err < 1\%$ for $\beta \in \{20, 40, 60\}$. Since we considered the total number of software interrupts received per second in the estimation, this trend means that the impact of the interrupts raised for purposes other than packet reception (e.g., for kernel housekeeping) becomes more noticeable when the offered load is very low. An improvement could be to look into the breakdown and consider NET_RX/s, as pointed out in Sub-section III-C1.

B. Utilization

Fig. 4a shows how the obtained utilization measurements and estimates compare with the model. For $OL < 20000$ pps, a spread among ρ_{meas} samples obtained at the same load conditions is observed. The correction proposed in Sub-section III-C2 reduces the spread, but not so much when the load is low. This behaviour implies that the virtualization



(a) measured and estimated utilization



(b) rescheduling interrupt rate (RES/s)

Fig. 4: Estimating the utilization due to the VNF workload.

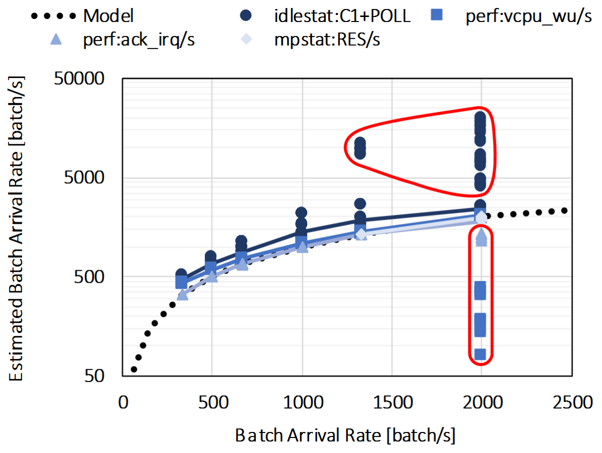


Fig. 5: Estimating the batch arrival rate using various PMCs.

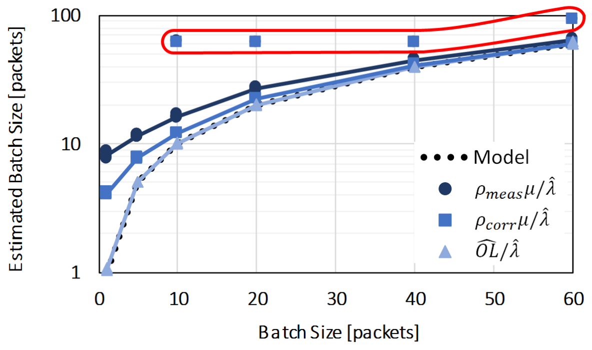


Fig. 6: Estimating the average batch size.

overhead is not fixed and depends on the traffic burstiness, further motivating the need to characterize VNF workloads. On the other hand, using OL for estimating the utilization matches the model, giving a hint on the fraction of the utilization due to actual packet processing by the VNF.

Although the incoming traffic is generated such that $OL < \mu$, it was observed that some cases result to $\rho_{meas} \approx 1$ (and $\rho_{corr} \approx 1$), as pointed out by the red shape. Looking at the statistics, it was found that the rescheduling interrupt rate (RES/s) in the five cases shot up to over two orders of magnitude than in normal working conditions, with values approximately equal to the batch arrival rates (RES/s $\approx \lambda$), as illustrated in Fig. 4b. This result can be useful to indicate an anomaly, for instance, in the system configuration, resource allocation or VNF behaviour. Moreover, utilization-based VM metering proves to be inadequate in such scenarios - this requires fine-tuning of existing usage-based pricing models or perhaps development of new ones.

C. Batch Arrival Rate

Fig. 5 shows a comparison of different approaches in estimating λ. As before, the red shapes point out the values obtained when $\rho_{meas} \approx 1$ - they are not considered in computing the averages indicated by the corresponding lines.

Aside from the utilization overhead (i.e., the part unaccounted by the proposed correction in Sub-section III-C2, such as the interval before a core actually sleeps) that keeps the core busy for a bit longer than in the model, the average idle period obtained with the `idlestat` command exhibits an increasing dependence on β as λ increases - hence, it is not the best choice for estimating λ. Both `kvm:kvm_vcpu_wakeup` and `kvm:kvm_ack_irq` gave stable estimates and anomalous measurements were only obtained in a single case - $(\lambda,\ \beta) = (2000, 60)$ - instead of five, but higher estimation accuracy is achieved with the latter, making it the most suitable PMC in most cases. Additionally, we show how RES/s matches λ when $\rho_{meas} \approx 1$. The results suggest that the batch arrival rate is best estimated as follows.

$$\hat{\lambda} = \begin{cases} \frac{\#\text{ of } \texttt{kvm:kvm\_ack\_irq} \text{ events}}{2\Delta t} & \text{if } \rho_{meas} < 1 \\ \text{RES/s} & \text{if } \rho_{meas} \approx 1 \end{cases} \qquad (11)$$

D. Average Batch Size

Fig. 6 shows the $\hat{\beta}$ values obtained according to the estimation approaches proposed in Sub-section III-C4, with the same convention for anomalous cases as before. Note that the samples for each batch size are obtained by generating β packets at varying batch arrival rates - yet the estimation results are quite stable, as it should be. Moreover, the average batch size is best estimated from $\hat{OL}$ and $\hat{\lambda}$.

On the other hand, Fig. 6 also portrays a different perspective on the utilization overhead as a whole (i.e., the sum of components due to the setup, virtualization, sleeping, etc.). In the worst-case scenario of single packet arrivals ($\beta = 1$), the core works as if $\beta \approx 8$. The impact of the overhead gradually decreases with increasing β - perhaps it can be thought to be spread among the packets in a batch. To reduce this overhead problem, a possible approach could be traffic shaping via interrupt moderation. Furthermore, having an accurate characterization of both λ and β can be useful in optimizing the interrupt coalescing parameters in the NIC according to the desired trade-off between the overhead and the end-to-end latency.

V. Conclusion

NFV is foreseen to play a major role in the upcoming softwarization of the network, as it provides flexibility and programmability levels that would help support future demands by implementing networking functionalities as software on VMs. Numerous studies have demonstrated how performance, power consumption and, consequently, the optimal resource configuration and VM allocation vary with the statistical features of the workload - specifically, the burstiness of the traffic. In this respect, a model-based analytics approach for profiling VNF workloads that captures traffic burstiness, considering - and adding value to - PMCs available in Linux host servers, is proposed.

The $M^X/G/1/SET$ core model is used as a basis for comparison of the PMC-based measurements and estimates. PMCs obtained with the `mpstat`, `idlestat` and `perf stat` commands are considered in this work, exposing the most suitable ones for measuring/estimating the offered load, utilization, batch arrival rate and average batch size. Then, to evaluate the proposed approach, an experimental testbed based on Intel® Xeon® E5-2643 v3 3.40GHz processors and Ixia's NX2 router tester is used. Results show good estimation accuracies for the chosen PMCs. In addition, the impact of the utilization overhead is also illustrated, pointing out possible anomaly detection use-cases and motivating the need for improved usage-based pricing models.

For future work, we would like to extend this approach to cover VM sharing and to evaluate its performance for VNF implementations based on Intel's Data Plane Development Kit (DPDK) [28].

Acknowledgment

This work was supported by the INPUT (In-Network Programmability for next-generation personal cloUd service supporT) project, funded by the European Commission under the Horizon 2020 Programme (Grant no. 644672).

References

[1] M. Chiosi, et al., "Network Functions Virtualization: An Introduction, Benefits, Enablers, Challenges & Call For Action," White Paper, 2012. [Online]. Available: http://portal.etsi.org/NFV/NFV_White_Paper.pdf

[2] "Advanced Configuration and Power Interface Specification." [Online]. Available: http://www.acpi.info/DOWNLOADS/ACPIspec40a.pdf

[3] R. Bolla, R. Bruschi, F. Davoli, and F. Cucchietti, "Energy Efficiency in the Future Internet: A Survey of Existing Approaches and Trends in Energy-Aware Fixed Network Infrastructures," *IEEE Commun. Surveys Tuts.*, vol. 13, no. 15, pp. 223–244, 2011.

[4] L. Duan, D. Zhan and J. Hohnerlein, "Optimizing Cloud Data Center Energy Efficiency via Dynamic Prediction of CPU Idle Intervals," in *Proc. 2015 8th IEEE Int. Conf. on Cloud Computing (CLOUD)*, New York, NY, 2015, pp. 985–988.

[5] E. Hernandez-Valencia, S. Izzo and B. Polonsky, "How Will NFV/SDN Transform Service Provider OPEX?" *IEEE Netw.*, vol. 29, no. 3, pp. 60–67, May 2015.

[6] "Network Functions Virtualisation (NFV); Management and Orchestration," ETSI NFV ISG Specification, 2014. [Online]. Available: http://www.etsi.org/deliver/etsi_gs/NFV-MAN/001_099/001/01.01.01_60/gs_nfv-man001v010101p.pdf

[7] R. Bolla, R. Bruschi, A. Carrega, and F. Davoli, "Green Networking with Packet Processing Engines: Modeling and Optimization," *IEEE/ACM Trans. Netw.*, vol. 22, no. 1, pp. 110–123, Feb. 2014.

[8] L. Chen, S. Patel, H. Shen and Z. Zhou, "Profiling and Understanding Virtualization Overhead in Cloud," in *Proc. 2015 44th Int. Conf. on Parallel Process. (ICPP)*, Beijing, China, Sep. 2015, pp. 31–40.

[9] S. Gebert, T. Zinner, S. Lange, C. Schwartz and P. Tran-Gia, "Performance Modeling of Softwarized Network Functions Using Discrete-Time Analysis," in *Proc. 2016 28th Int. Teletraffic Congr. (ITC)*, Würzburg, Germany, Sep. 2016.

[10] R. Bruschi, F. Davoli, P. Lago and J. F. Pajo, "Joint Power Scaling of Processing Resources and Consolidation of Virtual Network Functions," in *Proc. 2016 5th IEEE Int. Conf. on Cloud Networking (CloudNet)*, Pisa, Italy, Oct. 2016.

[11] "KVM." [Online]. Available: http://www.linux-kvm.org/

[12] W. von Hagen, "Using KVM Virtualization," Tech. Rep., 2014. [Online]. Available: https://www.ibm.com/developerworks/library/l-using-kvm/l-using-kvm-pdf.pdf

[13] "QEMU." [Online]. Available: qemu.org

[14] S. Zeng and Q. Hao, "Network I/O Path Analysis in the Kernel-Based Virtual Machine Environment Through Tracing," in *Proc. 2009 1st Int. Conf. on Inform. Sci. and Eng. (ICISE)*, Nanjing, China, Dec. 2009, pp. 2658–2661.

[15] "Ethtool(8) - Linux Man Page." [Online]. Available: https://linux.die.net/man/8/ethtool

[16] G. Choudhury, "An $M^X/G/1$ Queueing System with a Setup Period and a Vacation Period," *J. Queueing Syst.*, vol. 36, no. 1-3, pp. 23–38, 2000.

[17] H. Tijms, *A First Course in Stochastic Models*. John Wiley & Sons Ltd, England, 2003.

[18] A. Klemm, C. Lindemann, and M. Lohmann, "Modeling IP Traffic using the Batch Markovian Arrival Process," *J. Performance Evaluation*, vol. 54, pp. 149–173, Oct. 2003.

[19] P. Salvador, A. Pacheco, and R. Valadas, "Modeling IP Traffic: Joint Characterization of Packet Arrivals and Packet Sizes using BMAPs," *J. Comput. Networks*, vol. 44, pp. 335–352, Feb. 2004.

[20] S. Godard, "Mpstat Manual Page," Sep. 2016. [Online]. Available: http://sebastien.godard.pagesperso-orange.fr/man_mpstat.html

[21] "Ubuntu Manpage: Idlestat - A CPU Power-state Analysis Tool." [Online]. Available: http://manpages.ubuntu.com/manpages/xenial/man1/idlestat.1.html

[22] "Perf-stat(1) - Linux Man Page." [Online]. Available: https://linux.die.net/man/1/perf-stat

[23] "Perf-list(1) - Linux Man Page." [Online]. Available: https://linux.die.net/man/1/perf-list

[24] J. Medhi, *Stochastic Models in Queueing Theory*, 2nd ed. Elsevier Science USA, 2003.

[25] "Intel® Xeon® Processor E5-2643 v3 (20M Cache, 3.40 GHz)." [Online]. Available: http://ark.intel.com/products/81900/Intel-Xeon-Processor-E5-2643-v3-20M-Cache-3_40-GHz

[26] "OpenWrt." [Online]. Available: https://openwrt.org/

[27] R. Schöne, D. Molka and M. Werner, "Wake-up Latencies for Processor Idle States on Current x86 Processors," *Comput. Sci. - Research and Develop.*, vol. 30, no. 2, pp. 219–227, 2015.

[28] "DPDK." [Online]. Available: http://dpdk.org/