On the Learnability of Software Router Performance via CPU Measurements

Charles Shelbourne
University College London
charles.shelbourne.18@ucl.ac.uk

Leonardo Linguaglossa
Telecom ParisTech
leonardo.linguaglossa@gmail.com

Aldo Lipani
University College London
aldo.lipani@ucl.ac.uk

Tianzhu Zhang
Telecom ParisTech
tianzhu.zhang@nokia.com

Fabien Geyer
Technical University of Munich
fgeyer@net.in.tum.de

ABSTRACT

In the last decade the ICT community observed a growing popularity of *software networking paradigms*. This trend consists in moving network applications from static, expensive, hardware equipment (e.g. router, switches, firewalls) towards flexible, cheap pieces of software that are executed on a commodity server. In this context, a server owner may provide the server resources (CPUs, NICs, RAM) for customers, following a Service-Level Agreement (SLA) about clients' requirements. The problem of resource allocation is typically solved by *overprovisioning*, as the clients' application is opaque to the server owner, and the resource required by clients' applications are often unclear or very difficult to quantify. This paper shows a novel approach that exploits machine learning techniques in order to infer the input traffic load (i.e., the expected network traffic condition) by solely looking at the runtime CPU footprint.

1 CONTEXT

The need for replacing complex network hardware components with simpler (yet less performing) software equivalents has fueled a novel research area: this has brought several advances in the state-of-the-art high-speed packet processing engines, by bringing line-rate capabilities to commodity off-the-shelf servers. Alongside with the growing popularity of networking paradigms such as Software-Defined Networking [10] (SDN) and Network Function Virtualization [15] (NFV), we witness the rise of several network applications relying solely on software components, but capable of performance comparable to legacy hardware equipment [4, 6, 12].

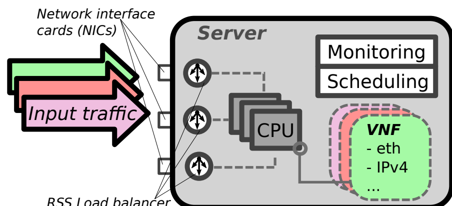

Figure 1: System architecture.

The typical structure of a modern software networking application is shown in Fig. 1. In this scenario, a server owner may provide one (or more) CPU(s) to her tenants. A SLA is signed between the owner and the tenants, which defines the usage of the resources such as bandwidth, or CPUs. Tenants can deploy the desired application in the form of Virtual Network Functions (VNFs). Receive-side scaling (RSS) [9] is the *scheduling* technique commonly used to assign one (or more) CPU(s) to each tenant, and to steer the incoming traffic to the respective CPU. *Monitoring* can be performed at NIC level (by the tenants), or at CPU level (by the OS, therefore by the server owner). In this context, the server owner is not allowed to access the deployed VNFs. Furthermore, the input traffic is also opaque to the owner as NICs usually adopt a *kernel-bypass* approach to reach high speed (cfr. Sec.III of [11]). As such, after resource allocation no optimization is possible on the owner's side: for example, the CPUs must be always active even in the case of zero traffic.

With machine learning (ML) techniques becoming more and more popular in the network community [14], we believe that it is possible to learn which traffic is related to the measurable CPU behavior (which is in turn indirectly related to the input traffic). In fact, differently from NICs, CPUs are still under the control of the operating system, which can monitor the underlying behavior and export the data by using standard tools. The server owner can exploit this knowledge to optimize the usage of resources, and keeping at the same time this process transparent to the tenants.

The main contribution of this paper is a novel approach for detecting network traffic conditions via *indirect* measurements: that is, we infer the network load by watching the CPU behavior. Rather than collecting network-related measurements, our methodology can be used to infer the network behavior by solely looking at the CPU patterns of the host machine, as different load conditions will produce different effects on several features of the CPU, such as the number of instructions issued per time unit, or the number of cache misses due to the processing. In particular, (i) we collect the CPU measurements using `perf` tools; (ii) we use the collected data to train some ML models and (iii) we validate our results with a variety of traffic typologies and rates. We target the simple case of one server with a single CPU handling the traffic coming from one input NIC to one output NIC). We show that, in this way, it is possible to infer the observed input rate with an accuracy value greater than 0.9 with a variety of neural networks. We observe that the input traffic (especially the size of the received packets) can alter the learning process and highly reduce the accuracy.

2 TRAFFIC LOAD INFERENCE VIA ML

Problem statement. Given an unknown traffic source, is it possible to infer the traffic load by considering only the CPU pattern that the VNF code is producing? We formulate this problem as a classification task. The *input* is a series of measurements (e.g., number

LSTM	fx		fxL		ps		pcpPINJ1		pcpPINJ2		pcpDC1		pcpDC2	
	ROC	acc	ROC	acc	ROC	acc	ROC	acc	ROC	acc	ROC	acc	ROC	acc
fx	0.973	0.964	0.509	0.345	0.971	0.962	0.618	0.491	0.509	0.345	0.548	0.397	0.558	0.41
fxL	0.499	0.332	0.948	0.931	0.499	0.332	0.503	0.337	0.542	0.389	0.518	0.358	0.517	0.355
ps	0.951	0.934	0.509	0.345	0.961	0.948	0.8	0.734	0.509	0.345	0.505	0.34	0.509	0.345
pcpPINJ1	0.817	0.756	0.43	0.24	0.954	0.938	0.815	0.753	0.384	0.179	0.437	0.249	0.432	0.242
pcpPINJ2	0.498	0.331	0.775	0.7	0.499	0.332	0.507	0.343	0.512	0.35	0.652	0.536	0.656	0.541
pcpDC1	0.921	0.895	0.522	0.363	0.84	0.787	0.526	0.368	0.508	0.345	0.793	0.724	0.674	0.565
pcpDC2	0.554	0.405	0.51	0.347	0.657	0.542	0.661	0.548	0.507	0.342	0.779	0.706	0.736	0.648

Table 1: ROC and Accuracy scores (trained on rows, tested on columns).

of instructions, branches, accesses to the CPU caches, etc.) collected at CPU level using the Linux `perf` tools. For a complete list of all the features, refer to the URL of the GitHub repository [3]. The *output* is one of three classes reflecting the input load: $\{low, medium, high\}$. The values are 0.5 Gbps, 5 Gbps and 9 Gbps respectively. We use three different neural networks architecture for our evaluations: multi-layer perceptron (MLP), convolutional neural network (CNN) and long short term memory (LSTM).

Experimental setup. All the data are collected from a multi-core commodity server, mounting two processors (each with 12 physical cores) and two Intel 82599ES dual-port 10-Gbps NICs [3]. A group of CPU cores is specifically isolated from the kernel and reserved for our experiments. We chose VPP v19.04 [4, 5], a high-speed software router under Linux Foundation's fd.io project, as VNF router. VPP executes a simple L2 forwarding VNF. MoonGen [8], a high-speed software traffic generator, is used to render network traffic with varied rates/patterns. We run `perf` [2] to profile and collect the CPU features connected to the packet processing performed by the VNF router.

Data collection. This step consists of two phases. In the first half, we start VPP in polling mode and link its process ID to `perf`. As no traffic is present, the measurements refer to the idle state of the CPU. In the second half of the experiment, MoonGen starts transmitting packets to VPP with a preassigned traffic rate. The measurements of this phase are stored with the preassigned traffic rates as labels.

Traffic types. We configure MoonGen to generate synthetic constant bit rate (CBR) as well as Poisson traffic [13]. We consider two extreme cases with small (64B) and large (1438B) packet sizes. The former represents the most stressful scenario in terms of packets per second, while the latter aims to detect VPP's behavior dealing with large packet payloads. In addition, we choose four different real traffic traces. Two traces come from a *packet injection* scenario (one mostly filled with small packets, while the other with mixed packet sizes) [1]. We finally select two data-center traces from the work of Benson et al. [7]. Both synthetic and real packet traces are used to create the training, validation and test datasets. For each traffic type we created a dataset that consists of 5 sequences of 2,910 instances with evenly distributed class labels, where each instance consists of 50 data-points sampled every 100 ms, and each pair of instances overlap by 40 data-points. We used 3 sequences for the training set, 1 sequence for the validation set and 1 sequence for the test set.

3 DISCUSSION

Table 1 shows the evaluation results for LSTM models trained on data from traffic types listed as rows, tested on different traffic types listed as columns. The traffic types are: CBR with small packets (fx), CBR with large packets (fxL), Poisson with small packets (ps), packet injection traces (pcpPINJ1 and pcpPINJ2) and data center traces (pcpDC1 and pcpDC2). We report the ROC score and the accuracy as performance metrics. The color shading shows dark blue as high score and dark red as low scores. Due to lack of space, we omit the results related to the CNN and the MLP models. We show strong evidence of learning, which however is affected by the packet sizes of the generated packets. In fact, our LSTM model trained with fx traffic can easily predict the same category of traffic, as well as the ps traffic, but struggles with the other traffic types. When the model is trained with ps traffic, it can as well predict well the traffic coming from the pcpPINJ1 trace (consisting of mostly small packets). This is because the Poisson process is able to generate some new information that the model can use to make better predictions. The fxL can only be predicted when the model is trained with the same traffic category, or the pcpPINJ2. This is due to the fact that the CPU behavior is different when large packets are sent (as with the same bitrate, the packet rate differs). Both data center traces show good results with a training performed on another data center dataset. Finally, the pcpPINJ2 traffic cannot be predicted by any of our training sets: since it represents a complex traffic pattern (with many different packet sizes), we plan to provide additional training with a more diverse synthetic traffic as future work.

REFERENCES

[1] 2012. Capture files from Mid-Atlantic CCDC. https://www.netresec.com/?page=MACCDC. (2012).

[2] 2019. Performance Counters for Linux, PCL. https://perf.wiki.kernel.org/index.php/Main_Page(version 4.15.18). (2019).

[3] 2019. Repository of the Project. https://github.com/CharlieShelbourne//master-thesis-project-tpt-ai4perf//blob/master/features.info. (2019).

[4] 2019. VPP - fd.io. https://wiki.fd.io/view/VPP. (2019).

[5] D. Barach, L. Linguaglossa, D. Marion, P. Pfister, S. Pontarelli, and D. Rossi. 2018. High-Speed Software Data Plane via Vectorized Packet Processing. *IEEE Communications Magazine* 56, 12 (December 2018), 97–103. https://doi.org/10.1109/MCOM.2018.1800069

[6] Tom Barbette, Cyril Soldani, and Laurent Mathy. 2015. Fast userspace packet processing. In *2015 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*. IEEE, 5–16.

[7] Theophilus Benson, Aditya Akella, and David a. Maltz. 2010. Network traffic characteristics of data centers in the wild. In *IMC*. 267. https://doi.org/10.1145/1879141.1879175

[8] Paul Emmerich, Sebastian Gallenmüller, Daniel Raumer, Florian Wohlfart, and Georg Carle. 2015. MoonGen, a scriptable high-speed packet generator. In *IMC*. 275–287. https://doi.org/10.1145/2815675.2815692

[9] Tom Herbert and Willem de Bruijn. 2011. Scaling in the linux networking stack. https://www.kernel.org/doc/Documentation/networking/scaling.txt. (2011).

[10] Diego Kreutz, Fernando Ramos, Paulo Verissimo, Christian Esteve Rothenberg, Siamak Azodolmolky, and Steve Uhlig. 2014. Software-defined networking: A comprehensive survey. *arXiv preprint arXiv:1406.0440* (2014).

[11] Leonardo Linguaglossa, Stanislav Lange, Salvatore Pontarelli, Gábor Rétvári, Dario Rossi, Thomas Zinner, Roberto Bifulco, Michael Jarschel, and Giuseppe Bianchi. 2019. Survey of Performance Acceleration Techniques for Network Function Virtualization [45pt]. *Proc. IEEE* (2019).

[12] Luigi Rizzo. 2012. Netmap: a novel framework for fast packet I/O. In *21st USENIX Security Symposium (USENIX Security 12)*. 101–112.

[13] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, George Porter, and Alex C Snoeren. 2015. Inside the social network's (datacenter) network. In *ACM SIGCOMM Computer Communication Review*, Vol. 45. ACM, 123–137.

[14] Junfeng Xie, F. Richard Yu, Tao Huang, Renchao Xie, Jiang Liu, Chenmeng Wang, and Yunjie Liu. 2019. A survey of machine learning techniques applied to software defined networking (SDN): Research issues and challenges. *IEEE Communications Surveys and Tutorials* 21, 1 (2019), 393–430. https://doi.org/10.1109/COMST.2018.2866942

[15] Bo Yi, Xingwei Wang, Keqin Li, Min Huang, et al. 2018. A comprehensive survey of network function virtualization. *Computer Networks* 133 (2018), 212–262.