Data-driven Resource Allocation in Virtualized Environments

Lianjie Cao
Hewlett Packard Labs
lianjie.cao@hpe.com

Sonia Fahmy
Purdue University
fahmy@cs.purdue.edu

Puneet Sharma
Hewlett Packard Labs
puneet.sharma@hpe.com

***Abstract*—Modern advances in virtualization technologies have revolutionized how we build and manage computer systems. Virtualization technologies, however, adversely impact the predictability of system performance, which introduces several challenges in balancing performance and resource utilization.**

In this dissertation, we explore and address performance challenges by characterizing and modeling application performance for resource allocation in two application scenarios: distributed network emulation and network functions virtualization (NFV). More specifically, we focus on preserving experiment fidelity for distributed network emulators running on heterogeneous physical machines, while, in NFV, we characterize performance impacts of various virtualization and configuration options and make timely resource flexing decisions.

***Index Terms*—virtualization, resource allocation, network functions virtualization, network emulation**

I. Introduction

In the past decade, virtualization technologies have become the cornerstone of many modern computer systems. Server virtualization, storage virtualization and network virtualization have revolutionized the business model of the IT industry and academic research in many areas such as networking, distributed systems and data science. For instance, public cloud platforms (*e.g.*, Amazon AWS, Microsoft Azure and Google Cloud) transform infrastructure resources of large-scale data centers to virtualized resources and many enterprises have been migrating their services to public clouds to reduce capital expenditure (CapEx) and operational expense (OpEx) by leasing or purchasing virtualized resources from public cloud providers. Experiment platforms (*e.g.*, GENI, CloudLab and Chameleon) also provide virtualized resources to researchers to conduct large-scale networking and distributed systems research experimentation and education.

Virtualization technologies improve the flexibility of resource allocation by introducing an abstraction layer between hardware resources and applications. For instance, with server virtualization, public clouds mask hardware specifications of the physical machines and allow users to provision virtualized resources to their applications. This model enables users to flexibly instantiate distributed applications without concerning the specifications of underlying infrastructure and elastically adjust resource allocation of applications in accordance with workload variations. However, as virtualization technologies simplify resource management of applications, it also affects the predictability of system performance, especially for applications that are conventionally implemented and deployed in proprietary hardware (*e.g.*, network functions and telco systems). With the same amount of virtualized resources, applications may exhibit significantly different system capacity (i.e., the maximum application throughput) if the virtualized resources are mapped to different infrastructure under the hood. The mapping between virtualized resources and underlying infrastructure is a nondeterministic process to users and it is usually handled by the resource management component of cloud orchestration platform. It makes resource allocation a challenging task for application operators to achieve target performance goals (*e.g.*, service-level objectives (SLOs)). The impact of this problem may become even more significant as we embrace hybrid cloud/IT, serverless computing and intelligent edge in which applications are highly decomposed and deployed on heterogeneous infrastructure across multiple geographical locations [1].

978-3-903176-15-7 © 2019 IFIP

While a large number of research work in the literature focus on virtual machine migration [2]–[4], performance improvement of virtualization technologies [5], [6]) and comprehensive management frameworks of various virtualized applications [7], [8]), there are very few work explicitly focus on the performance problem in virtualized environments. A straightforward approach to address this problem is to tolerate the performance unpredictability by over-provisioning applications. However, this contradicts the key original motivations of adopting virtualization technologies – cost saving. Stratos [7] and E2 [8] propose comprehensive NFV orchestration frameworks to manage VNF instances and distribute flows efficiently. However, E2 [8] relies on the VNF developer to give the overload indicator for scaling a VNF with more instances. The resource provisioning strategy in Stratos [7] monitors OS-level statistics only and does not specify how they are used to detect overload. For network emulation, VT-Mininet [9] proposes an adaptive virtual time system to scale network emulator Mininet [10] which prolongs the experiment duration. More detailed discussion of related work can be found in our papers.

In this dissertation [11], we argue the need for introducing performance models for allocating resources to applications in virtualized environments. We propose to *evaluate performance impact of infrastructure configurations or create performance models of applications and leverage the performance in-*

formation/models to make resource allocation decisions in virtualized environments. As the operating model of different virtualized environments varies, we explore the effectiveness of the proposed approach in two application scenarios: distributed network emulation and network functions virtualization. To summarize, this dissertation makes the following contributions:

- **Achieving high performance fidelity in distributed network emulation** – Running network emulation experiments on a heterogeneous cluster may lead to low experimental fidelity if the experiment is not properly separated and allocated. We tackle this problem by quantifying the traffic processing capability of physical machines. Then we model the problem as a graph partitioning problem and design the *Waterfall* algorithm [12] that leverages the traffic processing model of physical machines, together with the experiment topology, to determine an efficient mapping of the network experiment while preserving high experiment fidelity.
- **Characterizing performance of virtual network functions** – Determining the right set of configuration options and virtualization/hardware knobs to achieve the maximum potential performance of virtual network functions (VNFs) is a challenging task for VNF operators. We propose a performance characterization framework, *NFV-VITAL* [13], [14], to automatically evaluate the performance impact of hardware and software options and determine the optimal configurations for the initial deployment of a VNF.
- **Making auto-scaling decisions for virtual network functions** – With a proper initial deployment, operators need to make timely scaling decisions to reduce SLO violations with minimum amount of resources. We propose an Elastic resource flexing system for Network functions VIrtualization (*ENVI*) [15], [16], to make accurate online scaling decisions based on evolving neural network classifiers. ENVI trains initial neural network classifiers using experimental data sets collected during the offline stage, continues to update them using a window-based rewinding mechanism during online operation to capture emerging workload patterns and leverages the updated classifiers to make online scaling decisions.

II. Distributed Network Emulation

Virtualization technologies are widely used by modern network emulators [10], [17], allowing researchers to run network experiments using limited physical resources. Compared to network simulators (*e.g.*, ns-2 and ns-3) and testbeds (*e.g.*, Emulab, GENI and DETER), network emulators provide an intermediate point between simplicity and experimental fidelity. A key requirement for network emulation is to maintain high *performance fidelity* for any network experiment. When the experimental topology is large or the experiment is highly traffic-intensive, a physical machine (PM) running the experiment may become overloaded, leading to fidelity loss [18], [19]. A natural way to address this problem is to extend the network emulator to run across multiple PMs as shown in Fig. 1. However, for heterogeneous cluster which is typical as hardware upgrades cannot be performed at the same time, violation of performance fidelity becomes even worse if the network experiment is not properly allocated.

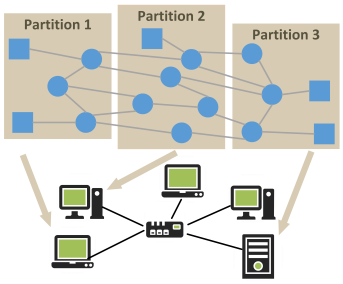

Fig. 1. Mapping a network experiment onto a heterogeneous cluster.

Following the proposed approach, we design a framework for mapping a network experiment onto a heterogeneous cluster to address this challenge. In this context, we consider emulated network devices and end hosts as the application and the experiment fidelity as the application performance metric. We leverage MaxiNet [20], which extends the popular Mininet network emulator [10] to run on a set of physical machines. Our framework quantifies the capacity of the PMs in the cluster, and uses this information in our *Waterfall* algorithm which leverages a popular graph partitioning engine METIS [21], [22] to map a network experiment onto some or all of the cluster PMs.

A. Modeling Traffic Processing Capacity

In the context of network emulation, preserving performance fidelity means assuring users not only correct connectivity of the topology, but also accurate performance of the end hosts and network devices which are emulated by software switches and routers. The experimental fidelity of network emulations largely depends on the performance of the software switches. Therefore, we quantify both the hardware specifications such as CPU type and memory size and the traffic processing capacity of software switches for each PM in the cluster.

Our measurements show that most software switches are CPU-intensive. Hence, we focus on CPU utilization in this work and leave other types of resource limits for future work. Note that, NIC may become the bottleneck in some experiments in which our approach becomes less effective. We consider two dominating factors for modeling CPU performance: single core performance and number of cores. We assign each core with 100 share and use a coefficient to represent the relative strength of single core performance. We compute the traffic processing capacity function $P^i(u_s^i)$ of PM i, where u_s represent the number of CPU shares on PM i. For instance, $P^i(50)$ denotes the maximum traffic rate that PM i can handle when allocating 50 CPU shares to traffic processing. Capacity functions are determined by running our resource quantification module on a simple linear topology. We investigated 6 different packet sizes from 64 to 1250 Bytes, and observed that throughput in Mbps varies significantly at

the same CPU usage for a given software switch. Eq. 1 shows an example of the capacity function of Stanford reference switch (UserSwitch) running on a PM with quad-core CPUs at 2.40 GHz and 16 GB RAM derived using 4 different linear topology sizes. We perform up to fifth-order polynomial regression on the data sets we collected and select the model with minimum mean square error (MMSE) using 5-fold cross validation. A user can choose a different regression model, if desired.

$$P_{4core@2.39GHz}(u) = 0.279u^2 + 316.796u + 948.393 \quad (1)$$

B. Partitioning and Mapping Experiments

We model the process of mapping a network experiment onto a heterogeneous cluster as a graph partitioning problem. We first convert the network topology to a weighted graph $G = (V, E)$ including $|V|$ vertices which equals the number of switches and routers, and $|E|$ links which equals the number of edges among switches and/or routers. Weights $w(v)$ and $w((a, b))$ denote the weight of vertex v, and the weight of the edge between vertex a and vertex b, respectively.

As the core of our framework, we design the *Waterfall* algorithm for the partitioning and mapping module which takes three inputs: (1) weighted graph $G = (V, E)$, (2) resource requirements of end hosts (*e.g.*, CPU usage), and (3) traffic processing capacity function derived in section §II-A. This module generates k' subgraphs $S_1, S_2, \cdots, S_{k'}$, each of which corresponds to an experiment partition, where k is the number of available PMs, S_i is the experiment partition that will be executed on PM i and $k' \leq k$ which means not all PMs have to be used. A partition S_i includes a subset of the vertices of the original graph, where the union of these subsets is V and the intersection is ϕ. The objectives are (1) localizing traffic as much as possible by mapping vertices that are densely connected via high-bandwidth links onto the same PM, and (2) attempting not to overload selected PMs, while maximizing the utilization. The entire mapping process is iterated through the following steps and terminates when no further improvement is observed or the maximum number of iterations is reached.

Multi-level Graph Partition. *Waterfall* first selects a minimal set of PMs from all available PMs in the cluster by comparing the total weight of the graph and the maximum processing capacity derived from the PM capacity functions. It then allocates CPU shares for traffic processing and emulated end hosts on each PM and computes parameters for the partitioning engine METIS. Then METIS is invoked to perform the partitioning task.

Result Evaluation. Based on the min edge-cut value and assignment returned by METIS, *Waterfall* computes and ranks the actual assignment of CPU shares for traffic processing and emulated end hosts of each PM. An assignment is ranked according to (1) the number of PMs used in the assignment, (2) the number of over-utilized PMs, (3) the number of under-utilized PMs, (4) the degree of over-utilization of PMs and (5) The edge-cut given by the partitioning results. The first four factors are derived from usage information of the assignment, and the fifth is directly returned by METIS. We focus on reducing the overall resource usage, the number of PMs used, and the edge-cut.

CPU Shares Update. *Waterfall* updates the CPU shares on each PM based on the ranking in the previous step. To adjust CPU shares, we first sort the PMs by the descending values of the PM utilization. We then compute the CPU shares for the next iteration based on the output of the current iteration. The intuition is that, if a PM is overloaded, we assign it with the maximum load it can handle and move the excessive amount to the next most powerful PM; if a PM is under-utilized, we increase its assignment but no more than the shares added to any "stronger" under-utilized PM. The CPU share adjustment thus moves excessive CPU shares like a waterfall: overloaded shares flow towards the next most powerful PMs, and the room left for expansion in an under-utilized PM is limited. This is the reason we name this algorithm "Waterfall."

We evaluate our framework using both simulation and testbed experiments. For simulation, we compare the *Waterfall* algorithm with other partitioning methods on three different types of topologies ranging from 41 nodes to 690 nodes: RocketFuel, Jellyfish, and Fat-tree. We found that the *Waterfall* algorithm yields minimum PM over-utilization (1∼2%) and under-utilization (up to 7%), while other partitioning methods shows unstable and significantly higher over-utilization (up to 60%) and under-utilization (up to 70%) on some PMs. We also evaluate our framework using DDoS [23] experiments using RocketFuel topologies of two different sizes (11 nodes and 31 nodes) on a cluster with 6 different PMs. We show that with our framework, all selected PMs reach 90% CPU utilization and high performance fidelity (*i.e.*, all links achieve the desired throughput). Other partitioning algorithms either fail to achieve the desired link throughput as shown in Fig. 2 or yields low HTTP throughput even before DDoS attack happens due to poor resource allocation.

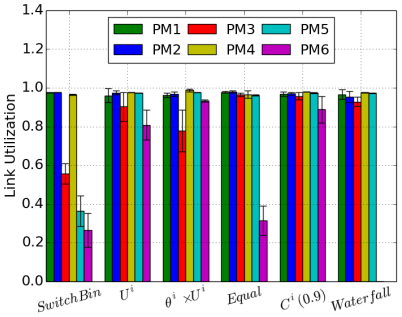

Fig. 2. Nomalized link utilization in DDoS experiments with 31 nodes

III. Network Functions Virtualization

Virtualization and automated resource orchestration, usually referred to as cloudification, have transformed IT operations and management. Telecommunication providers are leveraging virtualization technologies to move network functions (*e.g.*, intrusion detection system (IDS), caching proxy and Evolve

Packet Core (EPC)) from proprietary hardware to virtualized implementation on commodity devices. This adoption, called Network Functions Virtualization (NFV), increases agility, scalability, and elasticity of the IT infrastructure. However, the savings in operational expenses (OpEx) can only be attained and realized if virtual network functions (VNFs) are properly configured among the large number of configuration options and hardware settings (*e.g.*, CPU pinning, Intel DPDK, PF_RING, SR-IOV and NUMA) and scaled proportionally to the workload variations.

We address this challenge by applying the proposed approach to first evaluate the performance impact of various configuration and resource allocation options using *NFV-VITAL* and then make online scaling decisions using *ENVI*. Fig. 3 shows the overall architecture of *NFV-VITAL* and *ENVI* leveraging the NFV architectural framework from ETSI. In NFV context, VNFs are considered as the applications and we aim to improve the system capacity and reduce SLO violations. As VNFs are generally more diverse and complex than applications (*i.e.*, emulated network devices and end hosts) in network emulation, we use neural network to model the application performance in *ENVI*.

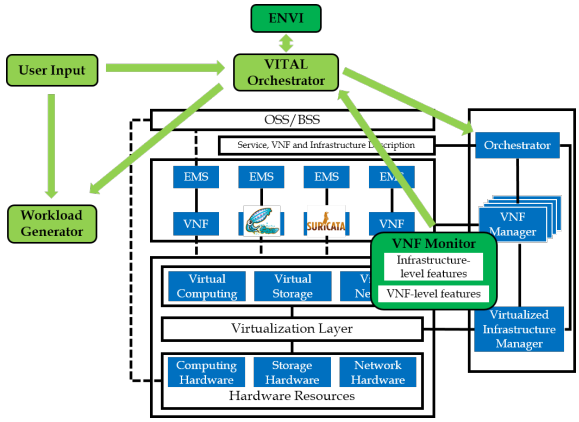

Fig. 3. NFV-VITAL and ENVI architecture

A. Characterizing VNF Performance

We first conduct a thorough experimental evaluation of a VNF with multiple components, Clearwater (an IP Multimedia Subsystem (IMS)), to understand how different resource allocation/scaling methods and configuration options affect VNF performance. We discover that different scaling methods (*e.g.*, scaling up/down and scaling out/in) may yield significantly different system capacity and performance behavior depending on the communication complexity and implementation details (*e.g.*, single-threading v.s. multi-threading and CPU-intensive v.s. memory-intensive).

Motivated by our observations of Clearwater case study, we design *NFV-VITAL* satisfying the following requirements: (1) accommodate different types of VNFs, (2) adapt the deployment size of a VNF with awareness of VNF components, (3) generate different VNF workloads, (4) collect resource utilization traces of all VNF instances, and (5) generate VNF performance evaluation reports. The framework consists of the following components: VITAL orchestrator, VNF workload generator, VNF load monitor, and user input.

User Input. *NFV-VITAL* allows users to provide their own deployment specification and workload specification files in `json` format. Deployment specifications include VNF information such as VM sizes to test, user preferences on server selection, testing modes and workload generation parameters.

VITAL Orchestrator. The VITAL orchestrator generates platform dependent deployment templates (*e.g.*, OpenStack Heat) indicating the deployment sizes to test. It then instantiates each deployment template and bootstrap application using installation scripts in user input. After deployment is completed, it invokes the VNF workload generator to initiate the testing process. During an experiment, the orchestrator executes a daemon process to receive and store resource utilization traces from the VNF load monitor and VNF performance logs from the VNF workload generator. After all tests are completed, system performance and resource utilization plots are generated for each test.

VNF Workload Generator. The workload generator is invoked by the VITAL orchestrator once a test deployment is completed. *NFV-VITAL* relies on the users to specify the range in which the workload generator operates and the type of traffic to use. The workload generator starts with the minimum workload rate, and gradually increases the rate until the stopping conditions are satisfied.

VNF Load Monitor. Since users may not have access to physical hosts on some platforms, the VNF load monitor runs inside VNF instances to collect CPU, memory, and network utilization, and stores them in a csv file during a test. Upon completion of a test, the utilization traces are collected by the VITAL orchestrator for further analysis.

Testing Modes. To make practical and thorough testing plans, *NFV-VITAL* offers three testing modes: custom sizing, exhaustive search, and component-aware directed search. Custom sizing allows users to specify different deployment sizes of their interests. The VITAL orchestrator then translates these deployment sizes into deployment templates. Exhaustive search is ideal when a user wants to test all possible deployment sizes for a given amount of resources (*e.g.*, number of vCPUs). The VITAL orchestrator first computes all possible combinations based on the given VM flavors of each VNF component that satisfy the resource requirements and then executes them sequentially. Component-aware directed search is designed based on the lessons we learned from the Clearwater case study – the overall VNF performance can be limited by a certain component. In this mode, the VITAL orchestrator starts testing with the minimal deployment size. It then analyzes the resource utilization traces when the system capacity is reached, and determines the VNF component with highest resource usage as the bottleneck component. Directed search then scales that component to generate the next deployment size to test. The entire testing process terminates either when the given target system capacity is achieved or when the given resource

limits are reached.

We demonstrate *NFV-VITAL* with IMS Clearwater and two IDSes Suricata and Snort using two different testing modes: component-aware directed search and custom sizing. We show that, for Clearwater, component-aware directed search is able to 1) accurately identify the bottleneck component for each tested deployment, 2) generate the optimal initial deployment configuration and 3) indicate the best scaling method – scaling up. For IDSes, *NFV-VITAL* suggests that scaling up Snort fails to improve the performance and the system capacity of Snort is sensitive to the traffic composition (*i.e.*, malicious traffic ratio), while Suricata scales very well with more resources added to the same instance and its performance remains stable as workload composition varies.

B. Flexing VNF Resource Allocation

With the initial VNF deployment configuration generated by *NFV-VITAL*, operators need to make timely resource flexing decisions during online operation in accordance with workload variations. Accuracy and timeliness of scaling decisions allow balancing the tradeoffs associated with resource allocation. Making scaling decisions long before actual overload causes under-utilization of resources allocated to a VNF (and hence higher operational expenses). Conversely, scaling decisions after the fact can incur penalties associated with violations of SLOs and even service disruption.

In this thesis, we propose *ENVI* to model the scaling decision-making process as a classification problem and leverage neural network to generate "not scale" or "scale" decisions based on both infrastructure-level and VNF-level information. We observe that service developers and users rely on critical internal runtime state information to manage the stability of their operational systems. Such critical internal information is recorded in system/application logs and has become a common source for debugging and monitoring running programs. We collect and mine this log data, along with infrastructure resource utilization information (referred to as VNF-level features and infrastructure-level features, respectively) to enhance our understanding of VNF runtime dynamics, and hence increase the accuracy of elastic resource flexing. We design *ENVI* with two stages: offline training and online updating.

Offline Training Stage. During this stage, we run various performance tests for each VNF and collect infrastructure-level and VNF-level feature information every time window $W = nT$. The n monitoring data points for each sampling period T in the same W are aggregate and extended with statistical measures (*e.g.*, max, min, mean, median and variance) for each feature as extended features. This process is repeated for multiple types of workload each of which is treated as one data set. Then we create and fit standardization (Z-score normalization) scalers to normalize values for each feature since many machine learning algorithms favor input values of similar range. Since the timeliness of scaling decision is critical, we artificially create a "buffer zone" between scaling decisions and actual VNF overload to guide the scaling event not to happen too soon or too late. To achieve this, we pick one feature as the key performance indicator (KPI) to estimate VNF capacity and use a configurable threshold α (e.g., 80%) to control the size of the buffer zone while labeling the collected samples. The samples with KPI value greater than $\alpha \cdot system\ capacity$ are labeled with 1, otherwise samples are labeled with 0, where $system\ capacity$ is detected during performance testing. The choice of KPI is based on the functionality of the tested VNF. The labeled samples are then used to train initial neural networks in batch mode. The offline stage can be incorporated into the software testing plan such as performance evaluation process using *NFV-VITAL*.

Online Updating Stage. *ENVI* continues to collect composite feature information during online stage. Initially, *ENVI* starts making scaling decisions based on the initial neural network classifiers. However, similar to many machine learning algorithms, the neural networks may fail with unseen input patterns if the workload changes over time. Hence, we must validate the scaling decisions by checking for false negatives (FNs) and false positives (FPs) and only true negative (TN) and true positive (TP) decisions should be enforced. By default, we scale the VNF with one more instance if TP decisions are detected. The occurrences of FNs and FPs indicate potential new workload patterns that the current classifier is not able to handle; hence, the classifier needs to be updated. However, due to the imbalanced samples of the two classes (as *ENVI* is designed to avoid overload) and the difficulty in computing the VNF capacity during online (as workload is not controlled like VNF performance testing) for labeling, we cannot simply update the classifiers with the collected samples. In addition, it is inefficient to update the classifier with every sample for a large number of VNF instances. Therefore, we introduce a window-based rewinding mechanism to select a number of historical samples with balanced classes as a training window. The basic idea is to backtrace the historical samples once FN sample is detected. During backtracing, we select the samples with KPI feature values greater than α of the KPI value of the FN sample, and relabel those samples as 1. Then we continue to backtrace and select the same number of samples and relabel them as 0. This window of samples with balanced number of 0s and 1s are then used to update the current classifier. The motivation behind this process is to approximate the labeling process at offline stage for consistency purpose and maintain the balance between the two classes.

We evaluate both the offline stage and online stage using caching proxy Squid and IDS Suricata with synthetic workload traffic and a real-world week-long NetFlow trace. For offline stage, we show that combining infrastructure-level and VNF-level features yields better classification accuracy (up to 20%) and neural network outperforms other classification models (decision tree, random forest and logistic regression) by 5%$\sim$14%. Online evaluation indicates that the initial classifiers trained with one workload type at offline stage can quickly adapt to new workload pattern and the accuracy stabilizes using our online updating mechanism. We also observe more accurate scaling decisions using *ENVI* which result in less resource consumption (29%) and close-to-zero SLO violations

comparing with baseline scaling policies. Fig. 4 shows an example comparing the number of instances yielded by *ENVI* and the best baseline scaling policy.

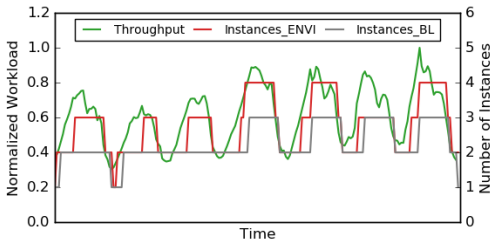

Fig. 4. Variation of workload and number of instances using Squid

IV. Conclusion and Future Work

In this dissertation, we propose to involve application performance modeling in making resource allocation decisions in virtualized environments. Following this approach, we introduce PM traffic processing capacity functions and leverage them to design the *Waterfall* algorithm to map network experiment onto a heterogeneous cluster for distributed network emulation. In NFV context, we develop *NFV-VITAL* to evaluate performance impact of various configuration and hardware options and generate initial VNF deployment configuration. We then propose *ENVI* to make online scaling decisions by modeling VNF performance using neural network classifier. In both cases, we are able to achieve more accurate and fine-grained resource allocation and better balance between application performance and resource allocation.

For network emulation, we are working on improving the accuracy of traffic processing capacity functions by including more factors such as memory usage and we are also planning to model the mapping process as an optimization problem and leverage integer linear programming (ILP) to solve it. The neural network classifier used in *ENVI* is not aware of the temporal information of the collected samples. However, the data sets collected from VNFs are typical time series data. We are planning to capture this temporal information with more powerful machine learning models such as long short-term memory (LSTM).

References

[1] "Hybrid Cloud Architectures with AWS," https://aws.amazon.com/enterprise/hybrid/.

[2] J.-J. Kuo, H.-H. Yang, and M.-J. Tsai, "Optimal approximation algorithm of virtual machine placement for data latency minimization in cloud systems," in *Proceedings of IEEE International Conference on Computer Communications (INFOCOM)*, 2014, pp. 1303–1311.

[3] X. Li, J. Wu, S. Tang, and S. Lu, "Let's stay together: Towards traffic aware virtual machine placement in data centers," in *Proceedings of IEEE International Conference on Computer Communications (INFOCOM)*, 2014, pp. 1842–1850.

[4] J. W. Jiang, T. Lan, S. Ha, M. Chen, and M. Chiang, "Joint VM placement and routing for data center traffic engineering," in *Proceedings of IEEE International Conference on Computer Communications (INFOCOM)*, 2012, pp. 2876–2880.

[5] M. Dobrescu, K. Argyraki, and S. Ratnasamy, "Toward predictable performance in software packet-processing platforms," in *Proceedings of USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2012, pp. 1–14.

[6] J. Martins, M. Ahmed, C. Raiciu, V. Olteanu, M. Honda, R. Bifulco, and F. Huici, "ClickOS and the art of network function virtualization," in *Proceedings of USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2014, pp. 459–473.

[7] A. Gember, R. Grandl, A. Anand, T. Benson, and A. Akella, "Stratos: Virtual middleboxes as first-class entities," University of Wisconsin-Madison, Tech. Rep., 2012.

[8] S. Palkar, C. Lan, S. Han, K. Jang, A. Panda, S. Ratnasamy, L. Rizzo, and S. Shenker, "E2: a framework for NFV applications," in *Proceedings of ACM Symposium on Operating Systems Principles (SOSP)*, 2015, pp. 121–136.

[9] J. Yan and D. Jin, "VT-Mininet: Virtual-time-enabled Mininet for scalable and accurate software-define network emulation," in *Proceedings of ACM SIGCOMM Symposium on Software Defined Networking Research (SOSR)*, 2015, p. 27.

[10] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks," in *Proceedings of ACM SIGCOMM Workshop on Hot Topics in Networks (HotNets)*, 2010, p. 19.

[11] L. Cao, "Data-driven resource allocation in virtualized environments." Ph.D. Dissertation, 2018. [Online]. Available: https://docs.lib.purdue.edu/dissertations/AAI10830296/

[12] L. Cao, X. Bu, S. Fahmy, and S. Cao, "Towards high fidelity network emulation," in *Proceedings of IEEE International Conference on Computer Communication and Networks (ICCCN)*, 2017. [Online]. Available: https://ieeexplore.ieee.org/document/8038453

[13] L. Cao, P. Sharma, S. Fahmy, and V. Saxena, "NFV-VITAL: A framework for characterizing the performance of virtual network functions," in *Proceedings of IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN)*, 2015. [Online]. Available: https://ieeexplore.ieee.org/document/7387412

[14] A. Sheoran, X. Bu, L. Cao, P. Sharma, and S. Fahmy, "An empirical case for container-driven fine-grained VNF resource flexing," in *Proceedings of IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN)*, 2016. [Online]. Available: https://ieeexplore.ieee.org/document/7919486

[15] L. Cao, P. Sharma, S. Fahmy, and V. Saxena, "ENVI: Elastic resource flexing for network function virtualization," in *Proceedings of USENIX Workshop on Hot Topics in Cloud Computing (HotCloud)*, 2017. [Online]. Available: https://dl.acm.org/citation.cfm?id=3154591

[16] L. Cao, S. Fahmy, P. Sharma, and S. Zhe, "Data-driven resource flexing for network functions visualization," in *Proceedings of ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, 2018. [Online]. Available: https://dl.acm.org/citation.cfm?id=3230725

[17] M. Pizzonia and M. Rimondini, "Netkit: easy emulation of complex networks on inexpensive hardware," in *Proceedings of ICST International Conference on Testbeds and Research Infastructures for the Development of Netwoks & Communities and Workshops (TridentCom)*, 2008, p. 7.

[18] W.-M. Yao, S. Fahmy, and J. Zhu, "Easyscale: Easy mapping for large-scale network security experiments," in *Proceedings of IEEE Conference on Communications and Network Security (CNS)*, 2013, pp. 269–277.

[19] R. Chertov and S. Fahmy, "Forwarding devices: From measurements to simulations," *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, vol. 21, no. 2, p. 12, 2011.

[20] P. Wette, M. Draxler, A. Schwabe, F. Wallaschek, M. H. Zahraee, and H. Karl, "MaxiNet: Distributed emulation of software-defined networks," in *Proceedings of IFIP International Networking Conference*, 2014, pp. 1–9.

[21] G. Karypis and V. Kumar, "Multilevel algorithms for multi-constraint graph partitioning," in *Proceedings of ACM/IEEE Conference on Supercomputing (SC)*, 1998, pp. 1–13.

[22] ——, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM Journal on Scientific Computing*, vol. 20, no. 1, pp. 359–392, 1998.

[23] M. S. Kang, S. B. Lee, and V. D. Gligor, "The Crossfire attack," in *Proceedings of IEEE Symposium on Security and Privacy (SP)*, 2013, pp. 127–141.