Data-driven Resource Flexing for Network Functions Virtualization

Lianjie Cao
Purdue University
cao62@purdue.edu

Sonia Fahmy
Purdue University
fahmy@cs.purdue.edu

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

Shandian Zhe
University of Utah
zhe@cs.utah.edu

ABSTRACT

Resource flexing is the notion of allocating resources on-demand as workload changes. This is a key advantage of Virtualized Network Functions (VNFs) over their non-virtualized counterparts. However, it is difficult to balance the timeliness and resource efficiency when making resource flexing decisions due to unpredictable workloads and complex VNF processing logic.

In this work, we propose an Elastic resource flexing system for Network functions VIrtualization (ENVI) that leverages a combination of VNF-level features and infrastructure-level features to construct a neural-network-based scaling decision engine for generating timely scaling decisions. To adapt to dynamic workloads, we design a window-based rewinding mechanism to update the neural network with emerging workload patterns and make accurate decisions in real time. Our experimental results for real VNFs (IDS Suricata and caching proxy Squid) using workloads generated based on real-world traces, show that ENVI provisions significantly fewer (up to 26%) resources without violating service level objectives, compared to commonly used rule-based scaling policies.

CCS CONCEPTS

• **Networks** → *Network dynamics*; *Network manageability*;

ACM Reference Format:
Lianjie Cao, Sonia Fahmy, Puneet Sharma, and Shandian Zhe. 2018. Data-driven Resource Flexing for Network Functions Virtualization.

ANCS '18, July 23–24, 2018, Ithaca, NY, USA

ACM ISBN 978-1-4503-5902-3/18/07…$15.00
https://doi.org/10.1145/3230718.3230725

In *ANCS '18: Symposium on Architectures for Networking and Communications Systems, July 23–24, 2018, Ithaca, NY, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3230718.3230725

1 INTRODUCTION

Virtualization and automated resource orchestration, usually referred to as cloudification, have transformed IT operations and management. Virtualization allows applications and services to be deployed on commodity hardware in public or private data centers to reduce capital expenses (CapEx). Telecommunication providers are now leveraging virtualization technologies to move network services (*e.g.*, intrusion detection systems (IDSes), caching proxies and Evolve Packet Core (EPC)) from proprietary hardware to virtualized implementations on commodity devices. This adoption, called Network Functions Virtualization (NFV), increases agility, scalability, and elasticity of their IT infrastructure.

The savings in operational expenses (OpEx) can only be attained and realized if the virtualized infrastructure is *Resource Proportional. Resource Proportionality* can be defined as allocation and consumption of compute, memory, networking and storage resources proportionally to the workload incident on the infrastructure. Most of the cloud computing workload is unimodal in terms of resource bottlenecks, e.g., CPU, memory or network. In contrast, NFV workloads are more complex and network functions can be bottlenecked at different or multiple resources. For example, caching proxies (*e.g.*, Squid) can be bottlenecked on both CPU and memory; IDSes (*e.g.*, Snort and Suricata) are primarily CPU-bottlenecked and may depend on incoming bit rates; L3 routing/firewall processing costs are typically proportional to packet rates.

While NFV orchestrators can elastically scale a deployment to adjust the system capacity and meet the requirements of varying workloads, there is a need for more sophisticated resource management. An important part of resource management is making decisions about when to scale/adjust resource allocation. 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 service level objectives (SLOs) and even service disruption. Designing an elastic resource management system for NFV can be very challenging due to (1) the lack of detailed performance specifications of VNFs for different configurations and workloads; (2) distinct processing logic of various VNFs; (3) the variable composition and volume dynamics of workload traffic; and (4) the cascading effects of scaling VNFs along a service function chain. We elaborate on these challenges in §2.

Most cloud platforms provide static rules-based policy interfaces (*e.g.*, OpenStack Heat [30], Amazon EC2 AutoScaling [1], and Google Cloud AutoScaler [13]) for users to define scaling strategies for their applications and services. These rules are specified in terms of basic resource utilization information (*e.g.*, CPU, memory and network) gathered by monitoring of the underlying infrastructure. Some researchers model VNF placement and resource allocation as an optimization problem that minimizes cost or maximizes system capacity. Although these solutions are successful in certain scenarios with single resource bottlenecks, most fail to account for the complex resource consumption behavior of today's services and VNF classes. As is evident from Figures 1(a) and 1(b), a static rule such as *scale when CPU utilization* $\geq$ 70% is insufficient for different HTTP workload types.

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. Such log data can be found (actively or passively) for most VNFs. For instance, both the Snort and Suricata IDSes report packet classification, throughput, and rule matching statistics in their logs. 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 understand the running status. We argue that combining these VNF-level features and infrastructure-level features (as composite feature sets) can enhance our understanding of VNF run-time dynamics, and hence increase the accuracy of elastic scaling.

In this paper, we propose an Elastic resource flexing system for Network functions VIrtualization (ENVI) to make scaling decisions based on an evolving neural network by periodically collecting data on VNF and infrastructure resources. ENVI uses these VNF-specific and infrastructure utilization information as input features to train a multilayer neural network during offline performance tests (*offline stage*). After deployment (*online stage*), ENVI continues to collect the same information and uses the previously trained neural networks as initial classifiers to make scaling decisions. Retraining is activated when false positive or false negative decisions are observed, in order to adapt neural networks to new patterns in workload traffic. Our earlier work [10] briefly explored the offline stage, whereas this paper introduces the online stage with new algorithms and experiments.

Due to the complexity of VNF processing logic and workload dynamics, it is infeasible to formulate precise mathematical models for VNFs even with a composite feature set. Therefore, we take an alternative approach. We model the scaling decision making process as a binary classification problem by capturing sophisticated relationships between VNF runtime status and system scaling using a neural network for each VNF. To make scaling decisions prior to actual VNF overload, the decision boundary generated by neural network classifiers should be ahead of the actual VNF capacity. However, VNF capacity is affected by a number of factors, such as composition of workload traffic, type and amount of virtualized resources provisioned, and VNF processing logic. We achieve this by enforcing a safety margin or "buffer zone" between decision boundaries and actual system capacities during the sample labeling process.

During the online stage, ENVI starts making scaling decisions using the initial neural networks. However, as with many machine learning algorithms, new input patterns arise when workload varies, leading to incorrect decisions. We need to continue updating the initial neural networks to keep up with dynamic workload traffic. Additionally, ENVI aims to avoid VNF overload by taking timely scaling actions, which may lead to a significantly smaller number of 0s ("not scale") than 1s ("scale") decisions produced by neural networks. This imbalanced number of the two classes creates a biased data set for training neural network during the online stage. To address these problems, we developed a window-based sample selection mechanism along with a rewinding mechanism to relabel and select balanced numbers of samples of the two classes to train the current neural network whenever false decisions are observed.

The remainder of this paper is organized as follows. §2 discusses the challenges in resource flexing and motivates the need of a more sophisticated approach. §3 describes the ENVI design and the tradeoffs involved. §4 discusses our experimental results and §6 summarizes related work. We conclude the paper in §7.

2 CHALLENGES

Resource flexing is complicated by the dynamic and composite nature of workload, and the resource consumption

behavior of different VNFs or applications. We discuss these challenges in this section, and explain why they render static or threshold-based resource flexing ineffective.

2.1 Workload Dynamics

Table 1 shows samples of Internet traffic protocol composition based on traces collected collected by CAIDA in Chicago from January 2016 to April 2016 [7]. Workload composition includes factors such as the mix of protocols used, packet size distribution, and application-level content. Although long-term traffic volume patterns (*e.g.*, diurnal and weekly patters) can be observed on certain links [5, 8, 43], both volume and composition are not sufficiently predictable to make scaling decisions at minute or even second time scales.

Table 1: Internet traffic composition collected by CAIDA in Chicago [7].

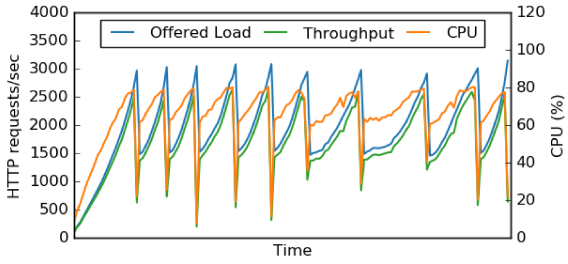
Application	01/21/16	02/18/16	03/17/16	04/06/16
HTTPS	52.07%	59.15%	49.91%	40.40%
HTTP	32.55%	28.60%	35.12%	38.76%
Other UDP	7.60%	5.24%	6.96%	5.99%
Other TCP	4.45%	3.66%	5.15%	11.73%
SSH	0.47%	0.44%	N/A	0.24%
ICHAT	0.24%	0.36%	0.31%	0.23%
RTMP	0.23%	N/A	0.34%	0.47%
Other	2.60%	2.53%	2.21%	2.18%

2.2 VNF Diversity and Complexity

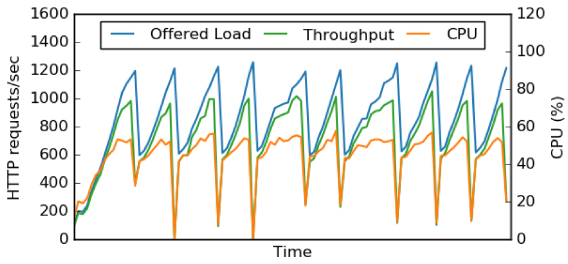
A variety of VNFs is being used, including NAT devices, firewalls, IDS/IPS systems, load balancers, WAN Optimizers, traffic shapers, proxies, and VPNs. Each of these consumes resources in very different ways. Resource consumption is influenced by a number of factors, including the amount of state maintained, the type of processing done, and the structure of the code (*e.g.*, single-threaded or multi-threaded). For example, some VNFs may terminate or originate TCP connections, and some do not. Some VNFs take actions per TCP connection, per individual packet, or per source IP address. Some VNFs inspect and/or modify packet headers only, while some inspect payloads.

Each VNF or application exhibits very different resource consumption patterns, based on both the workload *and* configuration. The workload and configuration impact the program paths taken, and hence the resource consumption. To study this impact, we conducted several experiments to investigate how resource consumption of the Snort IDS changes when the traffic is primarily UDP, versus HTTP, versus other TCP. We found that Snort resource consumption with UDP traffic scales per-packet (with each rule needing to be evaluated on each packet). Certain specific IDS rules interact badly with Snort design choices, causing unusually poor performance. In contrast, we found that resource consumption scales per TCP connection, not per TCP packet. In the case of HTTP, certain IDS rules only need to be evaluated once per-connection, but other rules are matched with the stream content, impacting resource consumption patterns.

We also conducted experiments with the Squid web proxy to quantify the relationship between its throughput (measured in completed HTTP requests per second) and CPU utilization under different workloads (different HTTP response sizes in our case). Fig. 1 illustrates the results of two types of workloads. The figure shows that a single resource usage indicator such as CPU utilization does not correspond to the same throughput under the two workloads. We conclude that taking resource flexing decisions based on a simple threshold on resource usage (such as CPU utilization) is insufficient.



(a) HTTP workload with 30 KB response size



(b) HTTP workload with 100 KB response size

Figure 1: Different capacities and CPU utilization of Squid with two types of workloads.

3 ENVI DESIGN

In this section, we describe the design and implementation of ENVI. ENVI includes two decoupled components: **VNF monitor** and **scaling decision engine**. As shown in Fig. 2, ENVI can be plugged into existing NFV management frameworks (*e.g.*, [6, 19, 32, 33]) and works with a control plane orchestrator/controller to enforce scaling decisions. ENVI has two operational stages: an offline stage to train an initial neural network from experimental data sets, and an online

stage to make scaling decisions and maintain the neural network up-to-date.

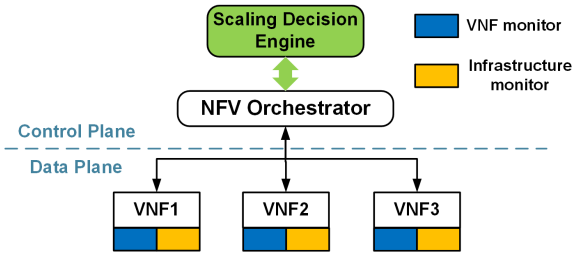


Figure 2: Example usage of ENVI.

3.1 Data Collection

ENVI utilizes both infrastructure-level and VNF-level features to understand run-time performance. Prior work (*e.g.*, [11, 45]) has reported improved performance when leveraging application information in scheduling. Although such information may not always be available due to privacy concerns in cloud computing scenarios, in the case NFV, administrators usually have access to both application (VNF) and infrastructure resources, since VNFs are often managed by service providers. VNF developers decide what information to expose. We have found that VNF system logs are universally-available sources of application information.

A VNF monitor is installed on each VNF instance and contains two agents: an infrastructure monitoring agent and a VNF monitoring agent. The same infrastructure monitoring agent is shared across all VNFs, whereas a VNF monitoring agent is developed for each deployed VNF. The VNF monitoring agents are standalone programs that monitor VNF system logs and extract critical information (mostly counters) in a key-value format. For instance, the Suricata monitoring agent periodically checks the event log file located at `/var/log/suricata/eve.json` and converts it to data entries we use. [1] The infrastructure monitoring agent collects 16 features related to CPU, memory, IO, networking and system usage. The two monitoring agents collect information for every time interval T ($T = 10$ seconds by default) as a monitoring data point.

3.2 Data Engineering

A scaling decision engine module pulls information consisting of VNF-level and infrastructure-level features from the VNF monitor every time window $W = nT$ (n is set to 10 by default) to avoid unnecessary scaling decisions caused by transient bursts in workload traffic. The module then aggregates the n monitoring data points in the same W. It computes statistical measures (*e.g.*, max, min, mean, median and variance) for each feature as extended features to capture temporal dynamics of W, and uses these values as one input sample for classifiers.

Some machine learning algorithms, especially those using gradient descent or similar algorithms, favor input values of similar range, since steepest descent is very sensitive to feature scaling. Based on our observations, value ranges of VNF-level features usually vary drastically leading to poor performance of neural networks if we do not scale input values. Thus, we create and fit standardization (Z-score normalization) scalers for all features during the offline stage to transform values of samples with a mean of 0 and a standard deviation of 1 before feeding them into the classifier. These feature scalers are continuously updated during the online stage as more samples are collected. More details are given in §5.

3.3 Training Neural Networks

With the preprocessed samples, ENVI employs a fully connected supervised neural network (using the *scikit-learn* library) as the classifier for scaling decisions. We empirically evaluate the key parameters of the neural network provided by *scikit-learn* including the neural network size (up to 6 hidden layers and 300 nodes per layer), activation function (*e.g.*, logistic, *tanh* and ReLU), learning rate ($1e-4$ to $1e-1$) and L2 penalty ($1e-7$ to $1e-1$). We select the best set of values for Suricata and Squid in our evaluation (a neural network with four layers: an input layer, two hidden layers (with 150 and 50 nodes) and an output layer, *tanh* activation function, $1e-2$ learning rate, and $1e-5$ L2 penalty). The neural network tuning process is part of the offline training stage and we may need to retune some parameters for different VNFs. We discuss other neural network architectures in §5.

As discussed above, making timely scaling decisions ahead of VNF overload while avoiding significant resource overprovisioning is the major challenge. ENVI addresses this problem by carefully labeling samples ahead of system capacities to guide neural networks to make timely scaling decisions. While the same data collection and data engineering mechanisms are used in the offline and online stages, neural network training differs among the two stages.

3.3.1 Offline training. Offline training creates the initial neural networks. This process can be easily incorporated into VNF software testing before deployment. In our case, we conduct a series of experiments for each VNF using a software stress testing framework (*e.g.*, [9]) with several synthetic workloads. We design the experiments with increasing workload rates until we reach the capacity with homogeneous

[1] Example features for Suricata and Squid as well as additional data for this paper can be found at https://www.cs.purdue.edu/homes/fahmy/nfv/envi.html.

workloads, then we continue to run experiments with randomly selected workload rates around the capacity value. Examples are given in Fig. 3.

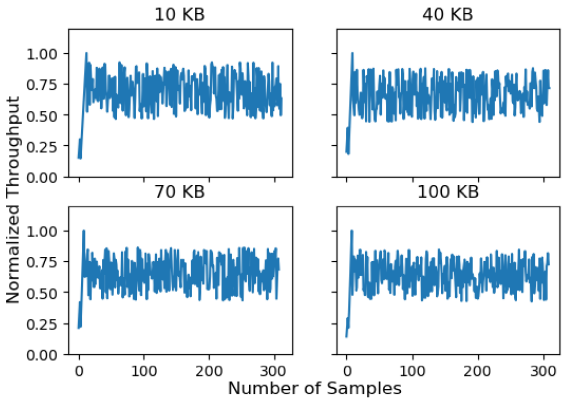


Figure 3: Samples of offline experiments conducted for Squid using different response sizes: 10 KB, 40 KB, 70 KB, and 100 KB.

To create a "buffer zone" between scaling decisions and actual VNF overload, we pick one feature as the key performance indicator (KPI) to estimate VNF capacity. We use a configurable threshold α (e.g., 80%) to control the size of the buffer zone while labeling the collected samples. The choice of KPI is based on the functionality of the tested VNF. For instance, we choose *requests/sec* for Squid and *packets/sec* for Suricata. Then, we label a sample i based on equation 1, where KPI_i is the value of the KPI feature in sample i and KPI_{cap} is the observed capacity value.

$$label_i = \begin{cases} 0 & \text{if } KPI_i \geq \alpha \cdot KPI_{cap}, \\ 1 & \text{otherwise.} \end{cases} \quad (1)$$

Labeled samples follow the data engineering steps discussed in §3.2, and are used to train initial neural networks in batch mode. This labeling process is designed to guide neural networks to form a decision boundary away from the VNF capacity (VNF overload in operation) and reinforce it through training samples.

Collecting training samples is the key to build the initial neural networks. Users may select software testing tools and workload generation methodologies accordingly, but there are a few guidelines to collect effective offline training samples:

- A VNF may have several candidate KPIs that can be used to estimate VNF capacity. We recommend selecting the most representative and separable feature to improve the performance of neural networks using domain knowledge. For instance, *requests/sec* and *service time* are equally important to Squid, but we choose *requests/sec* as the KPI because it is more separable in our experiments.
- To construct an accurate classifier, it is best to have a balanced number of samples of each class. Offline experiments should be carefully designed to generate equal numbers of samples of the two classes, while covering a large range of values around VNF capacity.
- Although VNF capacity is a function of several factors, our observations are that it is possible to find VNF capacities with workloads of fixed types (*e.g.*, workload composition does not change over time) in a strictly controlled lab environment. In our offline experiments, we fix all environmental variables (*e.g.*, hardware specifications, instance sizes) and generate homogeneous workloads to compute VNF capacities and label training samples.
- Our experiments on servers with different configurations show that VNFs may have different capacities with the same type of workload if the bare metal performance of servers are significantly different. To address this problem, we classify physical servers into different categories each of which share similar bare metal performance, and train initial classifiers for each server category.

3.3.2 Online updating. During the online stage, ENVI continues to collect composite feature information. Initially, ENVI starts making scaling decisions based on the initial neural networks. However, as with many machine learning algorithms, the neural networks may fail with unseen input values if the workload changes over time. For instance, if we train initial neural networks for Suricata with homogeneous UDP traffic, they may fail when processing hybrid UDP and HTTP traffic. The situation improves if we increase the workload coverage during offline training. However, it is infeasible to exhaustively train neural networks with all possible workload types for all VNFs. Therefore, instantly updating initial neural networks with new workload types/patterns during the online stage becomes a necessary step.

There are two major questions to address when updating neural networks during the online stage:

(1) How to maintain consistent buffer zones between decision boundaries of neural networks and VNF capacities during online stage? For the initial classifiers, the zone is introduced by labeling samples based on the parameter α and VNF capacities with homogeneous workload in a lab environment. However, predicting VNF capacity which varies as dynamic workload in real-time is itself a challenging task.
(2) How to train classifiers with imbalanced numbers of samples of the two classes? ENVI aims to avoid VNF

overload by effectively managing virtualized resources. As a result, we expect a significantly larger number of 0 labels than 1 labels, which naturally leads to imbalanced numbers of samples for online training.

3.3.3 Handling false negatives and false positives. Once scaling decisions are generated, we must validate them by checking for false negatives (FNs) and false positives (FPs). Only true negative (TN) and true positive (TP) decisions are enforced. More specifically, a false negative means that ENVI fails to scale a VNF when more resources are required, leading to SLO violations, whereas a false positive indicates unnecessarily aggressive upscaling decisions resulting in resource over-provisioning.

ENVI combines domain knowledge and user preferences to compose predefined policies for identifying inaccurate scaling decisions. To detect false positive decisions, ENVI allows users to define lower bounds for resource utilizations (U_{FP}). For instance, we may consider a scaling decision of 1 as false positive if both CPU and memory usages are lower than 30% for this sample (*e.g.*, $U_{FP} = 30\%$). Compared to false positives, false negatives have worse consequences such as SLO violations. Due to VNF diversity, ENVI allows users apply domain knowledge and define their own policies to detect false negative scaling decisions. In our evaluation (§4), we use software failures and mismatch of input and output traffic to detect false negatives for IDS Suricata and caching proxy Squid. Service time is also a good indicator of SLO violation.

3.3.4 Window-based rewinding. Classifiers need to adapt to emerging workload patterns and incorrect online scaling decisions. As discussed above, due to the imbalanced samples of the two classes and the difficulty in computing the VNF capacity for labeling, we cannot simply update the classifiers with every collected sample. In addition, it is inefficient to update classifiers with accurate scaling decisions for a large number of VNF instances. Therefore, we introduce a window-based rewinding mechanism to select a number of historical samples as a training window, relabel them with balanced classes, and update classifiers with the relabeled samples during the online stage.

Online classifier training is activated when false positive and false negative decisions are observed. First, a training window of a fixed size N_{win} ($N_{win} = 40$ slots by default) is created. For false positives, samples are sequentially inserted at the beginning of the first half of the training window with new labels of 0 and these false scaling decisions will not be enforced. When a false negative occurs, we first correct and enforce the scaling decision to positive. Then we add this sample to the end of the training window and mark the value of the KPI feature for this sample as KPI_{max}. We start to backtrace the previous samples and compare the value of the KPI feature KPI_i with KPI_{max}. If $KPI_i \geq \alpha \cdot KPI_{max}$, then this sample is inserted in the second last slot in the training window with a new label of 1. α is the same parameter we used in offline stage. This backtracing process is paused if $KPI_i < \alpha \cdot KPI_{max}$ or $n_{pos} \geq \frac{N_{win}}{2}$, where n_{pos} is the number of samples with positive labels in the training window. The training window contains $n_{pos} \in (0, \frac{N_{win}}{2}]$ samples with positive labels in the second half and possibly some samples with negative labels in the first half. Then we resume the backtracing to select the same number of samples, insert them in the first half of the training window starting from the $\frac{N_{win}}{2}th$ slot in reverse order, and relabel them as 0.

Since we also add samples to the training window when false positives are detected, it is possible that i) $n_{neg} > n_{pos}$, where n_{neg} is the number of samples with negative label, and ii) we may fail to add samples with negative labels during backtracing due to the existence of previously inserted samples. Considering the consequences of false negative and false positive scaling decisions, we view the correction of false negatives as higher priority. In other words, we pop out the existing samples with negative labels to guarantee the sample insertion during backtracing when slots in the first half of the training window are limited. In this case, n_{neg} may be slightly larger than n_{pos}. After the training window is constructed, the classifier is repeatedly trained with the selected samples until a certain level of accuracy (*e.g.*, 0.9) is reached in order to increase the weight of new samples. Another possible case is that the first half of the training window can be filled with samples detected as false positive before backtracing is activated, which means that the VNF is likely severely over-provisioned. If this happens, we update the classifier with the current training window. Essentially, the motivation behind window-based rewinding is to achieve consistent labeling and training of neural networks in both stages by approximating KPI_{cap} with KPI_{max}.

Fig. 4 illustrates the mechanism via an example of running Squid at online stage. *vnf.request* is selected as the KPI feature. We set 30% as the default threshold of resource utilization (*i.e.*, $U_{FP} = 30\%$) to detect false positive decisions. Sample #1 is detected as a false positive decision, then ENVI creates an empty training window with 10 slots (*e.g.*,, $N_{win} = 10$) and inserts sample #1 in slot #1 in the window. ENVI continues to collect more samples until sample #12 is detected as a false negative decision due to software failure which activates the backtracing process. Samples #12, #11, #10, and #9 are inserted into slots #10, #9, #8, and #7, respectively, and relabeled as 1, since $KPI_i \geq \alpha \cdot KPI_{max} = 807$. ENVI resumes the backtracing and selects the next four samples from #8 to #5, adding them to slots #5 to #2 in the training window. Eventually, the training window with 9 samples is used to update the neural network.

Figure 4: Example of window-based rewinding.

3.3.5 Enforcing scaling decisions. In addition to the two types of scaling decisions (0 for "not scale" and 1 for "scale") generated by neural networks, users may also want to reduce the amount of resources (scaling in/down) for VNFs if the workload is reduced, and SLO can be maintained with fewer resources. VNF downscaling depends on user preferences. For instance, users may choose to downscale a VNF if resource usage is below 30% instead of 50% to have less frequent deployment adjustments at the cost of higher resource over-provisioning. ENVI exposes an optional API for users to define downscaling policies based on resource usage thresholds (*e.g.*, U_{DS}).

Eventually, scaling decisions are pushed to the controller or orchestrator of the NFV management system and translated to executable commands. Previous studies (*e.g.*, [9]) have shown that scaling methods (up versus out and down versus in) may affect the overall VNF throughput. However, identifying the optimal scaling method for a given VNF is beyond the scope of this work. We offload the translation between scaling decisions and concrete commands with appropriate scaling methods to the underlying NFV management system.

The majority of the overhead incurred by ENVI comes from the data collection step performed offline. This can be incorporated into the software testing plan prior to production deployment of a VNF. During the online stage, we introduce the window-based rewinding mechanism to reduce the overhead caused by neural network update which is only activated when false decisions are detected. With the default settings (*e.g.*, neural network size and training window size), it takes less than 5 seconds to update the neural network in our experiments.

4 EVALUATION

In this section, we experimentally evaluate ENVI using two different VNFs: IDS Suriata (version 3.2.1) and HTTP caching proxy Squid (version 3.3.8). Our goal is to address three key questions:

(1) How effective are initial neural networks with previously unseen workload traffic?
(2) Does online updating of initial neural networks increase the accuracy of scaling decisions?
(3) Do scaling decisions consistently translate to improved system performance?

4.1 Methodology

4.1.1 Workload Generation. To train initial neural networks in the offline stage, we conduct a series of experiments with different types of homogeneous workloads. Each type of workload is defined by a combination of elements depending on the target VNF, such as source/destination addresses/ports, packets size, network protocol, flow size, and VNF-specific options. For instance, one type of workload traffic we generated for the IDS Suricata is denoted by < `address:(192.168.2.21:8000,192.168.2.31:8001),packet_size:1450,protocol:UDP,malicious_ratio:10` >. Here, `malicious_ratio` is a VNF-specific option for an IDS. We add a rule to the *Emerging Threats* rules [39] that matches UDP packets with the keyword "malicious" in the payload to mimic malicious traffic. For Squid, HTTP response sizes are used as the VNF-specific option to denote different workload types. We collected training data with 10 different types of workload for both Suricata (0 ~ 90% malicious traffic ratio) and Squid (10 ~ 100 KB HTTP response sizes).

For a given type of workload traffic, we first gradually increase the workload rate until SLO violation occurs. The value of the KPI feature is recorded as the VNF capacity for this specific workload type. We continue to generate workload traffic with randomly selected rate values within a custom range of the capacity (*e.g.*, $(0.6, 1.1) \times capacity$). We repeat this process for multiple types of workload traffic to generate offline data to train the initial neural network for each VNF.

Real-world network traffic is a mixture of different types of traffic generated by myriads of applications. Therefore, we also generate more complex workload traffic that mimics traffic captured from real networks. We extract the packet rate and flow rate information from a week-long network trace in Netflow format (collected on an access router connecting a university to its ISP [43]) and compress it to a 6-hour duration. To test IDS Suricata, we generate workload traffic that includes packet-level emulation of observed UDP traffic generated by hping3 [35] and flow-level emulation of observed TCP traffic generated by Harpoon [38]. For Squid, we generate HTTP workload using Web Polygraph [15] with HTTP request rates set to TCP flow rates extracted from the Netflow trace, and HTTP response sizes drawn from multiple statistical distributions.

4.1.2 Experimental Setup. ENVI is evaluated on a testbed of one Dell PowerEdge R430, two HP ProLiant DL120 G6 and one Gigabit Dell N2024 Switch and managed by the OpenStack Ocata release [29]. Details of the server configurations are shown in Table 2. Workload generation and VNF programs execute in dedicated VMs created and managed by OpenStack. We use the *networking-sfc* module of OpenStack to create MPLS tunnels on Open vSwitch and redirect workload traffic through VNF instances.

We tested ENVI with different values of parameters discussed in §3, such as $0.6 \sim 0.9$ for α and $20\% \sim 50\%$ for U_{FP} and U_{DS}. Lower values of α and U_{DS} indicate more aggressive upscaling decisions and more conservative downscaling decisions which lead to higher resource overprovisioning. We choose the following values for experiments in this section: $\alpha = 0.8$, $N_{win} = 40$, $U_{FP} = 30\%$, and $U_{DS} = 30\%$. *vnf.decoder.pkts* and *vnf.request* are selected as the KPI features for Suricata and Squid, respectively.

Table 2: Server and VM configurations

PM/VM	CPU	Cores	RAM
R430	2x Intel Xeon E5-2620 v4	16	64 GB
DL120	1x Intel Xeon X3430	4	8 GB
m1.small	1x vCPU	1	2 GB
m1.medium	2x vCPU	2	4 GB
m1.large	4x vCPU	4	8 GB

4.1.3 Metrics. We use two categories of metrics to evaluate the offline and online stages of ENVI. To address the first two evaluation questions, we use a set of typical statistical metrics: $accuracy = \frac{TPs+TNs}{TPs+TNs+FPs+FNs}$, $precision = \frac{TPs}{TPs+FPs}$, $recall = \frac{TPs}{TPs+FNs}$. *Accuracy* indicates the overall correctness of both positive and negative decisions. *Precision* represents the probability that positive decisions are correct, while *recall* denotes the coverage of positive decision. For instance, precision of 1 means all positive decisions are correct, but there may be false negative decisions. Recall of 1 means all positive events (ground truth) are successfully captured as positive decisions, but the generated positive decisions may also contain false ones. In addition, we evaluate all the methods in terms of the Area under ROC curve (AUC). AUC is one of the most commonly used metrics to evaluate the performance of classification algorithms. To calculate the AUC values, we use the predictive outcome for each test example as a threshold for classification, and obtain the True Positive Rate (TPR) and False Positive Rate (FPR). Connecting the (TPR, FPR) pairs for all the test examples, we obtain the ROC curve from which we calculate the AUC. Note that AUC measures the classification performance comprehensively, because it integrates the prediction accuracies with all possible thresholds.

To understand the system performance impact of the scaling decisions generated by ENVI, we sum the number of VNF instances provisioned for each sample to estimate the resource utilization, and we compute the number of samples in which SLOs are violated to estimate the timeliness of scaling decisions. An optimal scaling method should minimize the number of SLO violations and the total number of instances. However, reducing service violations is typically of higher priority if both goals cannot be simultaneously satisfied.

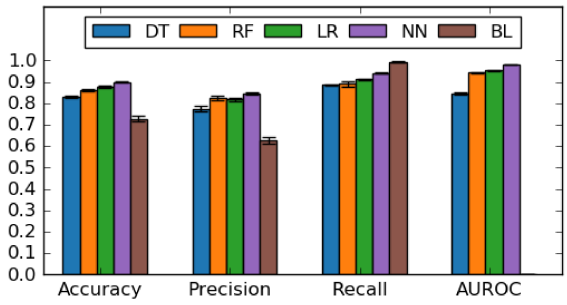
4.2 Classification Algorithms

We first compare the performance of a neural network (NN) with other classification algorithms: decision tree (DT), random forest (RF) and logistic regression (LR). We also choose a threshold-based policy which makes upscaling decisions if resource utilization exceeds 80% as a baseline (BL).

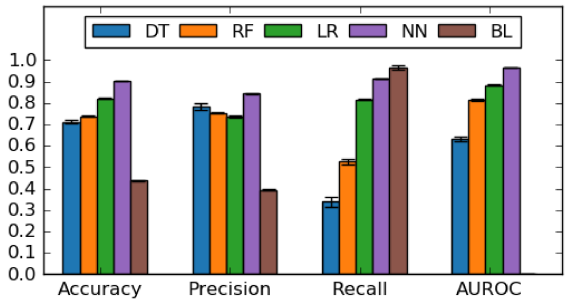
We generated 10 data sets each of which represents a unique workload type for Squid and Suricata, then we train a initial neural network model on n data sets and test it on the remaining $10 - n$ data sets, where $n = 1, 2, \cdots, 9$. For each n, we enumerate all possible combinations of data sets and run 5-fold cross-validation on the selected data sets to avoid overfitting before testing on the remaining $10 - n$ workload types.

Fig. 5 compares the performance of different classification algorithms and the baseline. For Suricata, the neural network yields slightly better results than other classification algorithms (*e.g.*, up to 8% in accuracy). However, for Squid, the neural network significantly outperforms other classification algorithms by up to 19% in accuracy. This is because Suricata is a simple VNF that captures packets and generates alerts based on predefined rule sets, while Squid is a more complex VNF that is capable of maintaining HTTP connections with clients and servers, caching repeated HTTP requests, etc. Hence, the relationship between scaling decisions and the composite features that a classification algorithm needs to approximate for Squid is more complex, which is one of the advantages of neural networks. This is consistent with another observation that the static baseline policy performs better for Suricata than Squid (72% vs. 43% in accuracy). A single resource utilization threshold fails to make accurate scaling decisions for different types of workload in the case of Squid.

Our earlier work [10] also compared the performance of classification algorithms using infrastructure-level features, VNF-level features, and composite feature sets. VNF-level features yield better results for complex VNFs, while composite feature sets always outperform the other two feature sets. Hence, we only use composite feature sets in this paper.

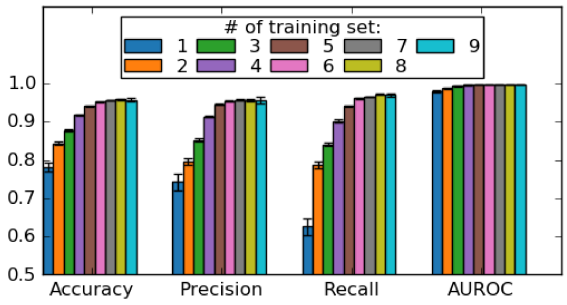


(a) Results with Suricata

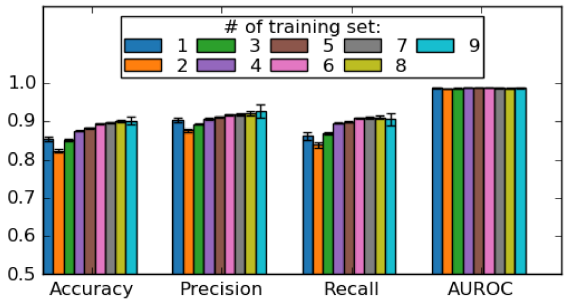


(b) Results with Squid

Figure 5: Comparison of neural network, decision tree, random forest and logistic regression in offline experiments.



(a) Results with Suricata



(b) Results with Squid

Figure 6: Neural network results using different numbers of workload types for training.

4.3 Online Updating

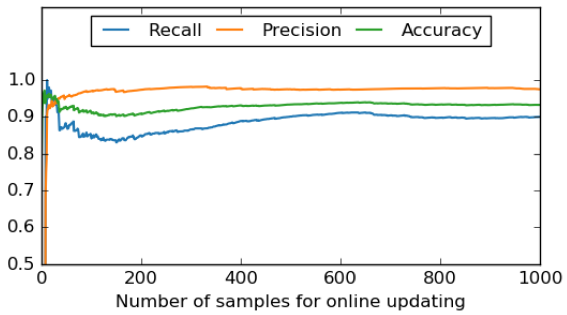
Fig. 6 shows results with the neural network using training data comprised of different numbers of workload types. For both Suricata and Squid, the performance of neural networks improves as additional workload types are covered during the offline stage. However, it is not practical or efficient to exhaustively test all possible workload types for a given VNF. Therefore, we evaluate how the ENVI online updating mechanism improves the performance of initial neural networks. We first train an initial neural network using one of the ten data sets we collected for Suricata and Squid during the offline stage. Then we emulate the online operation process by sequentially feeding samples from the remaining nine data sets, and we incrementally update the initial neural network using the online updating mechanism discussed in §3.3.2.

As shown in Fig. 7, the values of statistical metrics improve and stabilize as more samples are collected to update the initial neural networks for both Suricata and Squid. Interestingly, as we train with more samples, the performance fluctuation of neural networks caused by new workload types/patterns shrinks. Hence, the neural network becomes more stable and robust with online updating. Note that the statistical metrics in Fig. 7 are computed based on the original decisions produced by neural networks (without the false decision correction described in §3.3.3, which we incorporate in the remainder of this section).

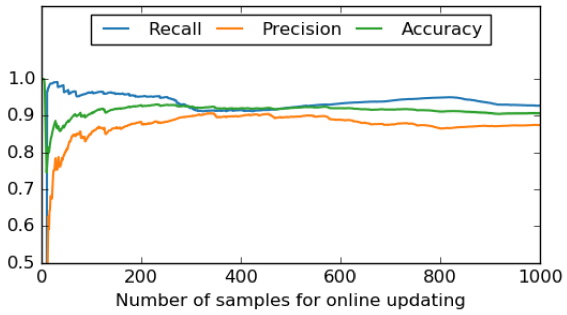
4.4 System Performance

In this section, we address the third evaluation question to understand the performance impact of ENVI scaling decisions during the online stage. We integrate ENVI with an NFV orchestrator to evaluate Suricata and Squid. ENVI makes "not scale" (0) and "scale" (1) decisions and a "scale" decision is interpreted as scaling out the current VNF deployment by one additional small instance. However, applying only these two types of scaling decisions leads to a monotonically increasing number of VNF instances. Hence, we introduce a heuristic downscaling policy in the orchestrator that reduces the number of instances by 1 if the average resource utilization is less than 30% and the corrected scaling decision given by ENVI is "not scale."

We compare ENVI with baseline threshold-based upscaling policies using different resource utilization threshold values (50% ∼ 90%). The optimal threshold for the baseline method uses the smallest number of instances while



(a) Online updating with Suricata



(b) Online updating with Squid

Figure 7: Online updating of initial neural networks for Suricata and Squid.

incurring zero SLO violations. We first compare the performance of ENVI and baseline method using multiple homogeneous workloads, then we evolve the traffic patterns by injecting temporal statistical variations. For instance, one workload generated for Squid may include *HTTP response size* $\sim \mathcal{N}(10\ KB, 2\ KB)$ for sample 0 to sample 30, *HTTP response size* $\sim \mathcal{N}(80\ KB, 16\ KB)$ for sample 31 to sample 50, *HTTP response size* $\sim \mathcal{N}(30\ KB, 6\ KB)$ for sample 51 to sample 60, and so on. This models changing traffic patterns, *e.g.*, the increasing sizes of web pages over the past decade [2]. The hybrid workload for Suricata consists of varying malicious traffic fractions (0% ~ 90%) over time.

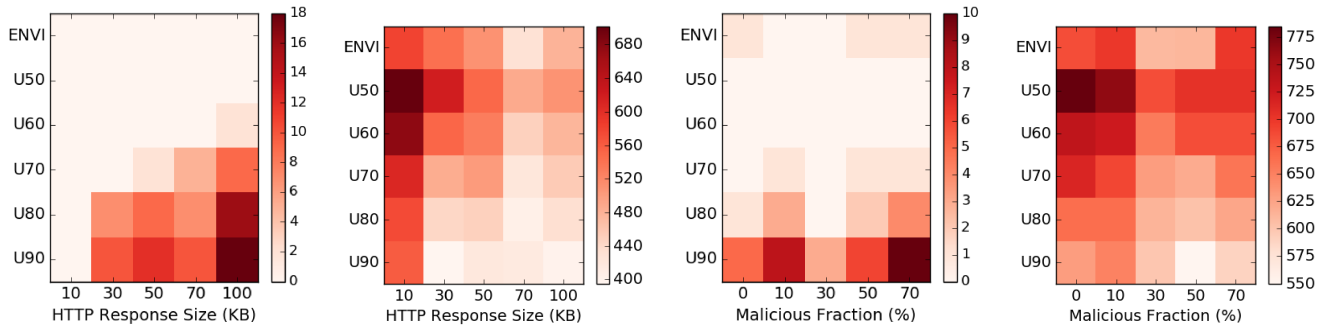
Homogeneous workloads. Fig. 8(a) and Fig. 8(b) compare the number of SLO violations and the total number of instances in Squid experiments for a selected group of workload types. Each column represents a certain workload type, while each row indicates a scaling method. We focus on identifying the optimal scaling method for each workload type instead of comparing various scaling methods across workload types. For a given workload type, reducing the threshold value of resource utilization in the baseline method reduces the number of SLO violations (denoted by the lighter color), while increasing the total number of instances (denoted by the darker color). For Squid, we found that (i) we can find an optimal resource utilization threshold for baseline method with homogeneous workload, and (ii) the optimal threshold varies for different homogeneous workload types. For instance, with 10 KB HTTP response size, 90% resource utilization (U90) is the optimal baseline method with zero violations and 560 total instances while 50% resource utilization (U50) becomes the optimal threshold that uses 410 total instances without any violations. With neural networks, ENVI is able to automatically generate accurate scaling decisions for all workload types while avoiding SLO violations or using too many instances. In the case of Suricata, different workload types yield distinct capacity values (packets/second), but resource utilization at capacity points are consistent. As shown in Fig. 8(c) and Fig. 8(d), a static threshold of 70% ~ 80% resource usage (U70 or U80) performs well for all tested workload types.

Hybrid workload. Table 3 shows the SLO violations and number of instances using hybrid workload for Suricata and Squid. As the hybrid workload combines all workload types in a single experiment, the only way to minimize SLO violations for baseline method is to use the smallest optimal threshold shown in Fig. 8(a) (U50) and Fig. 8(c) (U70). However, compared to ENVI, using U50 for Squid significantly overprovisions resources for some samples (*e.g.*, samples with smaller HTTP response sizes), leading to 26% more instances and much lower average CPU utilization (39% vs. 48%). For Suricata, U70 and ENVI yield similar results in terms of SLO violations and number of instances provisioned due to consistent resource utilization at VNF capacities. Fig. 9 shows how the number of instances tracks the variation of offered workload. We compare the two cases: when we execute the upscaling decisions generated by ENVI, versus the optimal baseline scaling policy. Downscaling decisions use the rule-based policy. ENVI appears to track the workload changes more closely.

To conclude, we find that it is feasible to compose a static threshold-based scaling policy for certain VNFs through exhaustive search if the workload type is fixed and known. In contrast, ENVI can automatically take appropriate scaling decisions with changing workloads, avoiding the extensive effort to continuously search for the optimal threshold.

5 DISCUSSION

Offline training data generation. ENVI uses offline experiments as training data to construct initial neural networks. Although the performance of initial neural networks can be improved during online updating, we suggest conducting offline experiments to cover as many workload types as possible. The motivation for using fixed workloads during the offline stage is to yield a steady VNF capacity in order to



(a) SLO violations for Squid (b) Number of instances for Squid (c) SLO violations for Suricata (d) Number of instances for Suricata

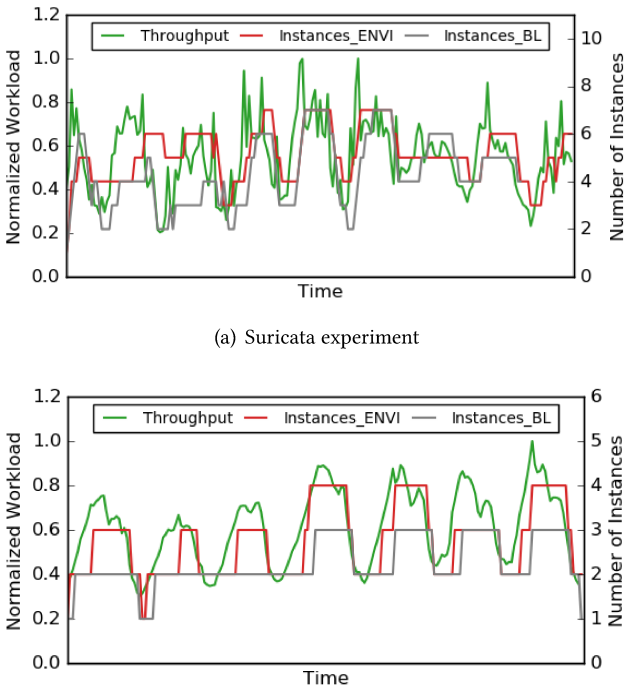
Figure 8: System evaluation using homogeneous workloads with Squid and Suricata.

Table 3: System evaluation using hybrid workload for Suricata and Squid.

	Suricata		Squid	
	Violations	Instances	Violations	Instances
U90	8	580	18	432
U80	3	604	11	470
U70	0	630	9	498
U60	0	663	5	545
U50	0	712	1	607
ENVI	0	600	0	503

label the training data. In practice, this may also be achieved by heterogeneous workloads as long as the workload composition is fixed. For instance, users may generate HTTP workloads with 50% 10 KB response size and 50% 100 KB response size for Squid during training. Our experience has been that using homogeneous workloads can simplify and speed up the offline training data generation process.

Choice of classification algorithms. We choose a fully connected small-scale neural network as the classifier for ENVI. Our evaluation (§4.2) shows that the neural network outperforms other classification models such decision tree, random forest and logistic regression. However, tuning a neural network with the right set of features/parameters is a challenging task. Creating a deep neural network may not necessarily improve the performance depending on the problem itself, and it usually requires extensive training. Since the composite feature data we collect for VNFs is a typical time series, it may be possible to exploit the temporal correlation among consecutive samples via more complex algorithms (*e.g.*, long short-term memory (LSTM)) for making more accurate scaling decisions. ENVI intentionally decouples scaling decision generation and execution to be compatible with existing NFV management systems. Another option is to model our problem as a *decision under uncertainty* problem that combines the two processes of decision generation and execution, and to leverage a reinforcement learning framework.



(a) Suricata experiment

(b) Squid experiment

Figure 9: Variation of hybrid workload and number of instances using ENVI with Suricata and Squid.

Feature scaling. Our experience shows that standardization performs best among feature scaling methods (*e.g.*, normalization, min-max and unit length scaling). In the ENVI prototype, we implemented a standardization scaler for each input feature in a neural network and we update the scalers

during the online stage. Intuitively, it would seem that creating multiple standardization scalers for each feature would work better, since workload changes may shift the center of scalers which can be significant if the workload changes drastically. Creating new scalers allows the neural network to track changes faster than updating the original scaler. We implemented this multi-scaler idea using a Dirichlet process. Interestingly, we did not observe a performance boost in our evaluation, compared to the simpler single-scaler idea. One possible explanation is that the traces we use to generate online workloads do not contain significant changes; hence multi-scaler is unnecessary in that case.

Scaling decision validation. During the online stage, ENVI validates scaling decisions before pushing them to the orchestrator for execution. False negative decisions are detected based on service quality violations while static rule-based policies (*e.g.*, resource utilization thresholds) are used to detect false positive decisions. Another approach we are planning to investigate is to apply statistical hypothesis testing to validate the correctness of a positive decision by leveraging the distribution of the latest resource utilization. Compared to static policies, this approach allows us to take the dynamics of workload and VNF status into consideration.

6 RELATED WORK

Previous work, *e.g.,* [12, 14, 16, 20, 22, 24, 26, 27, 37, 42], considered VNF resource allocation together with VNF placement, and modeled them as an optimization problem (*e.g.*, Integer Linear Programing (ILP)) that minimizes the total number of VNF instances, communication cost, or deployment cost under constraints of network traffic and physical network topology. This line of work assumes VNF capacities are known and static, ignoring performance characterization challenges. The work does not consider the impact of dynamic network traffic on different types of VNFs.

Stratos [18] and E2 [31] propose comprehensive NFV orchestration frameworks which manage VNF instances and distribute flows efficiently. However, E2 [31] relies on the VNF developer to give the overload indicator for scaling a VNF with more instances. The resource provisioning strategy in Stratos [18] bears some similarity to our approach. However, Stratos monitors OS-level statistics only and does not specify how they are used to detect overload. Woo et al. [44] propose to elastically scale network functions without compromising performance by organizing VNF state as a distributed shared object for stateful VNFs. ResQ [40] tackles the the high variability and unpredictability in throughput and latency for consolidated network functions by applying processor cache isolation to enforce performance service level objectives.

Resource allocation has also been studied in the context of cloud computing. In the public cloud, neither the provider nor the user is willing to share information, which hinders making efficient resource allocation decisions. In addition, the chaining requirements in NFV introduces special constraints. Most cloud platforms and third-party developers simply provide policy-based interfaces (*e.g.*, OpenStack Heat [30], Amazon AWS AutoScaling [1], Google Compute Engine AutoScaler [13] and RightScale [34]) to users to scale their application by monitoring basic infrastructure-level information. Some research work (*e.g.*, [21], [36] and [41]) assumes certain workload patterns exist, and identifies and stores resource assignment solutions for future use.

Learning approaches have proven effective in solving system problems. Nagaraj et al. [28], Liu et al. [23] and Arzani et al. [3] apply machine learning techniques to investigate system/network anomalies. Nagaraj et al. [28] compare distributed system logs using statistical tests and dependency networks to identify performance degradation. Liu et al. [23] and Arzani et al. [3] train random forest models on historical network data to determine network anomalies and root causes of failures, respectively. Gao et al. [17] predict power usage effectiveness (PUE) in Google data centers. Bao et al [4] guide cellular network resource allocation using user experience prediction. Mao et al. [25] solve the multi-resource allocation problem in a reinforcement learning framework. These approaches are orthogonal to our work.

7 CONCLUSIONS

An effective resource flexing system is important to take full advantage of the flexibility and scalability of network functions virtualization. Dynamic workload and complex VNF processing logic make timely resource flexing a challenging task. This paper presents ENVI, a modular component that works with existing NFV management systems, to make scaling decisions while balancing timeliness and resource efficiency. ENVI models VNF resource flexing as a classification problem and leverages a neural network to generate scaling decisions, using both infrastructure-level and VNF-level information as input features. Evaluation of IDS Suricata and HTTP caching proxy Squid shows that, compared to rule-based policies, ENVI can automatically scale complex VNFs to avoid overload for varying workloads while achieving good resource utilization.

ACKNOWLEDGMENTS

This work is sponsored in part by NSF grants CNS-1717493 and OAC-1738981, and was started and initially supported by HPE. We would like to thank Vinay Saxena (HPE) and our shepherd Michio Honda for their valuable input.

REFERENCES

[1] Amazon. 2018. Amazon EC2 Auto Scaling. https://aws.amazon.com/ec2/autoscaling/. (2018).

[2] HTTP Archive. 2018. HTTP Archive: Page Weight Report. https://httparchive.org/reports/page-weight. (2018).

[3] Behnaz Arzani, Selim Ciraci, Boon Thau Loo, Assaf Schuster, and Geoff Outhred. 2016. Taking the Blame Game out of Data Centers Operations with NetPoirot. In *Proceedings of ACM Conference on Special Interest Group on Data Communication (SIGCOMM)*. 440–453.

[4] Yanan Bao, Huasen Wu, and Xin Liu. 2016. From Prediction to Action: A Closed-Loop Approach for Data-Guided Network Resource Allocation. In *Proceedings of ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD)*. 1425–1434.

[5] Theophilus Benson, Aditya Akella, and David A Maltz. 2010. Network traffic characteristics of data centers in the wild. In *Proceedings of ACM SIGCOMM Conference on Internet Measurement*. 267–280.

[6] Anat Bremler-Barr, Yotam Harchol, and David Hay. 2016. OpenBox: A Software-Defined Framework for Developing, Deploying, and Managing Network Functions. In *Proceedings of ACM Conference on Special Interest Group on Data Communication (SIGCOMM)*. 511–524.

[7] CAIDA. 2016. The CAIDA UCSD Trace Statistics for CAIDA Passive OC48 and OC192 Traces - 2016. http://www.caida.org/data/passive/trace_stats/chicago-A/2016/. (2016).

[8] Lianjie Cao, Thibaut Probst, and Ramana Kompella. 2013. Phishlive: A view of phishing and malware attacks from an edge router. In *Proceedings of International Conference on Passive and Active Network Measurement (PAM)*. 239–249.

[9] Lianjie Cao, Puneet Sharma, Sonia Fahmy, and Vinay Saxena. 2015. 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)*. 93–99.

[10] Lianjie Cao, Puneet Sharma, Sonia Fahmy, and Vinay Saxena. 2017. ENVI: Elastic resource flexing for Network function VIrtualization. In *Proceedings of Workshop on Hot Topics in Cloud Computing (HotCloud)*. 1–11.

[11] Mosharaf Chowdhury, Yuan Zhong, and Ion Stoica. 2014. Efficient Coflow Scheduling with Varys. In *Proceedings of ACM Conference on Special Interest Group on Data Communication (SIGCOMM)*. 443–454.

[12] Stuart Clayman, Elisa Maini, Alex Galis, Antonio Manzalini, and Nicola Mazzocca. 2014. The dynamic placement of virtual network functions. In *Proceedings of IEEE Network Operations and Management Symposium (NOMS)*. 1–9.

[13] Google Cloud. 2018. Autoscaling Groups of Instances. https://cloud.google.com/compute/docs/autoscaler/. (2018).

[14] Rami Cohen, Liane Lewin-Eytan, Joseph Seffi Naor, and Danny Raz. 2015. Near optimal placement of virtual network functions. In *Proceedings of IEEE International Conference on Computer Communications (INFOCOM)*. 1346–1354.

[15] Measurement Factory. 2016. Web Polygraph. http://www.web-polygraph.org/. (2016).

[16] Xincai Fei, Fangming Liu, Hong Xu, and Hai Jin. 2018. Adaptive VNF Scaling and Flow Routing with Proactive Demand Prediction. In *Proceedings of IEEE International Conference on Computer Communications (INFOCOM)*.

[17] Jim Gao and Ratnesh Jamidar. 2014. Machine learning applications for data center optimization. *Google White Paper* (2014).

[18] Aaron Gember, Robert Grandl, Ashok Anand, Theophilus Benson, and Aditya Akella. 2012. *Stratos: Virtual middleboxes as first-class entities*. Technical Report. University of Wisconsin-Madison.

[19] Aaron Gember-Jacobson, Raajay Viswanathan, Chaithan Prakash, Robert Grandl, Junaid Khalid, Sourav Das, and Aditya Akella. 2014. OpenNF: Enabling Innovation in Network Function Control. In *Proceedings of ACM Conference on Special Interest Group on Data Communication (SIGCOMM)*. 163–174.

[20] Milad Ghaznavi, Aimal Khan, Nashid Shahriar, Khalid Alsubhi, Reaz Ahmed, and Raouf Boutaba. 2015. Elastic virtual network function placement. In *Proceedings of IEEE International Conference on Cloud Networking (CloudNet)*. 255–260.

[21] Zhenhuan Gong, Xiaohui Gu, and John Wilkes. 2010. Press: Predictive elastic resource scaling for cloud systems. In *Proceedings of IEEE International Conference on Network and Service Management (CNSM)*. 9–16.

[22] Xin Li and Chen Qian. 2016. An NFV Orchestration Framework for Interference-Free Policy Enforcement. In *Proceedings of IEEE International Conference on Distributed Computing Systems (ICDCS)*. 649–658.

[23] Dapeng Liu, Youjian Zhao, Haowen Xu, Yongqian Sun, Dan Pei, Jiao Luo, Xiaowei Jing, and Mei Feng. 2015. Opprentice: towards practical and automatic anomaly detection through machine learning. In *Proceedings of ACM Internet Measurement Conference (IMC)*. 211–224.

[24] Marcelo Caggiani Luizelli, Leonardo Richter Bays, Luciana Salete Buriol, Marinho Pilla Barcellos, and Luciano Paschoal Gaspary. 2015. Piecing together the NFV provisioning puzzle: Efficient placement and chaining of virtual network functions. In *Proceedings of IFIP/IEEE International Symposium on Integrated Network Management (IM)*. 98–106.

[25] Hongzi Mao, Mohammad Alizadeh, Ishai Menache, and Srikanth Kandula. 2016. Resource Management with Deep Reinforcement Learning. In *Proceedings of ACM Workshop on Hot Topics in Networks (HotNets)*. 50–56.

[26] Michael J McGrath, Vincenzo Riccobene, Guiseppe Petralia, Georgios Xilouris, and Michail-Alexandros Kourtis. 2015. Performant deployment of a virtualised network functions in a data center environment using resource aware scheduling. In *Proceedings of IFIP/IEEE International Symposium on Integrated Network Management (IM)n*. 1131–1132.

[27] Hendrik Moens and Filip De Turck. 2014. VNF-P: A model for efficient placement of virtualized network functions. In *Proceedings of IEEE International Conference on Network and Service Management (CNSM)*. 418–423.

[28] Karthik Nagaraj, Charles Killian, and Jennifer Neville. 2012. Structured comparative analysis of systems logs to diagnose performance problems. In *Proceedings of USENIX conference on Networked Systems Design and Implementation (NSDI)*. 26–26.

[29] OpenStack. 2018. OpenStack. http://www.openstack.org/. (2018).

[30] OpenStack. 2018. OpenStack - Heat. https://github.com/openstack/heat. (2018).

[31] Shoumik Palkar, Chang Lan, Sangjin Han, Keon Jang, Aurojit Panda, Sylvia Ratnasamy, Luigi Rizzo, and Scott Shenker. 2015. E2: a framework for NFV applications. In *Proceedings of ACM Symposium on Operating Systems Principles (SOSP)*. 121–136.

[32] Zafar Ayyub Qazi, Cheng-Chun Tu, Luis Chiang, Rui Miao, Vyas Sekar, and Minlan Yu. 2013. SIMPLE-fying Middlebox Policy Enforcement Using SDN. In *Proceedings of ACM Conference on Special Interest Group on Data Communication (SIGCOMM)*. 27–38.

[33] Shriram Rajagopalan, Dan Williams, Hani Jamjoom, and Andrew Warfield. 2013. Split/Merge: System Support for Elastic Execution in Virtual Middleboxes. In *Proceedings of USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. 227–240.

[34] RightScale. 2018. RightScale Optima. https://www.rightscale.com/products-and-services/products/rightscale-optima. (2018).

[35] Salvatore Sanfilippo. 2006. hping. http://www.hping.org/. (2006).

[36] Zhiming Shen, Sethuraman Subbiah, Xiaohui Gu, and John Wilkes. 2011. Cloudscale: elastic resource scaling for multi-tenant cloud systems. In *Proceedings of ACM Symposium on Cloud Computing (SoCC)*. 1–14.

[37] Runyu Shi, Jia Zhang, Wenjing Chu, Qihao Bao, Xiatao Jin, Chenran Gong, Qihao Zhu, Chang Yu, and Steven Rosenberg. 2015. MDP and machine learning-based cost-optimization of dynamic resource allocation for network function virtualization. In *Proceedings of IEEE International Conference on Services Computing (SCC)*. 65–73.

[38] Joel Sommers, Hyungsuk Kim, and Paul Barford. 2004. Harpoon: A Flow-level Traffic Generator for Router and Network Tests. *SIGMETRICS Performance Evaluation Review* 32, 1, 392–392.

[39] Emerging Threats. 2018. Emerging Threats rules. https://rules.emergingthreats.net/. (2018).

[40] Amin Tootoonchian, Aurojit Panda, Chang Lan, Melvin Walls, Katerina Argyraki, Sylvia Ratnasamy, and Scott Shenker. 2018. ResQ: Enabling SLOs in Network Function Virtualization. In *Proceedings of USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. 283–297.

[41] Nedeljko Vasić, Dejan Novaković, Svetozar Miučin, Dejan Kostić, and Ricardo Bianchini. 2012. DejaVu: accelerating resource allocation in virtualized environments. In *Proceedings of ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. 423–436.

[42] Xiaoke Wang, Chuan Wu, Franck Le, Alex Liu, Zongpeng Li, and Francis Lau. 2016. Online VNF scaling in datacenters. *arXiv* (2016).

[43] Simple Web. 2018. Simpleweb Traces. https://www.simpleweb.org/wiki/index.php/Traces. (2018).

[44] Shinae Woo, Justine Sherry, Sangjin Han, Sue Moon, Sylvia Ratnasamy, and Scott Shenker. 2018. Elastic Scaling of Stateful Network Functions. In *Proceedings of USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. 299–312.

[45] Timothy Wood, Prashant Shenoy, Arun Venkataramani, and Mazin Yousif. 2009. Sandpiper: Black-box and gray-box resource management for virtual machines. *Elsevier Computer Networks* 53, 17 (2009), 2923–2938.