MERIT: Model-driven Rehoming for VNF Chains

(Technical Report)

Muhammad Wajahat*, Bharath Balasubramanian†, Anshul Gandhi*, Gueyoung Jung†,
Shankaranarayanan Puzhavakath Narayanan†
*Stony Brook University †AT&T Labs - Research

***Abstract*—Network service providers often run service chains of Virtual Network Functions (VNFs) on privately owned clouds with limited capacity. These specialized service chains need to meet strict Service Level Objectives (SLOs), especially along the lines of availability (e.g., First responder services). Hence, VNFs in such thinly provisioned clouds may need to be frequently moved, or *rehomed*, when reacting to various cloud events like hotspots, failures and upgrades. In this report, we perform a detailed measurement study to show that naive strategies for rehoming, applied uniformly across all VNFs of the service chain, are often sub-optimal when considering different metrics like the user-perceived service downtime and the provider-incurred time delay to complete the rehoming. We propose a novel ModEl-driven RehomIng Technique (MERIT) for VNF chains and empirically analyze the effect of various system parameters on different rehoming actions. Based on our analysis, we develop generic rehoming cost models and further, design and implement an autonomous rehoming system based on MERIT that identifies and executes the optimal rehoming action for each VNF in a service chain. Our experimental results on OpenStack using real-world chains show that MERIT can reduce the chain rehoming delay by up to 47% and the chain downtime by up to 49%.**

I. Introduction

Network function virtualization (NFV) has several benefits such as cost-efficient deployment on commodity hardware, elasticity, and reduced tie-in to proprietary hardware. To avail these benefits, major Network Service Providers like AT&T and Verizon are replacing specialized networking hardware with Virtual Network Function (VNF) chains [1], where individual VNFs provide integrated network services (the *tenants*) on virtualized infrastructure (the *cloud provider*).

VNF service chains often have stringent performance Service Level Objectives (SLOs) [2]. To maintain the performance, resiliency and stability of these network services, the stringent Service Level Objectives (SLOs) of VNF service chains need to be met [2]. For example, first responder services (EMS, police, fire) requirevery high system availability [3]. This is challenging since events like hotspots (compute/network bottlenecks), VM workload shifts, etc., lower the performance and availability of the service [4], [5], [6], [7]. Traditional cloud providers, like Amazon EC2 [8] and Google Cloud [9], which have considerable resources, typically rely on redundancy and over provisioning to achieve stringent tenant SLOs. Network provider clouds are typically resource constrained as they primarily consist of cloud sites (including edge sites) that host anywhere between 10–500 servers [10]. To maintain service chain SLOs under these constraints, network providers often have to *rehome* (or move) one or more VNFs (or VMs, used interchangeably) of the service chain to a different host.

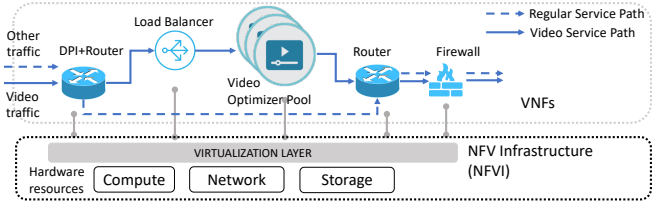


Fig. 1: An NFV ecosystem showing a video optimizing service chain, the VNFs of which are deployed on virtualized infrastructure (NFVI). The NFV-MANO helps in orchestration and life-cycle management of the service including monitoring the health of VNFs/NFVI and taking the appropriate remedial rehoming actions.

Figure 1 shows an NFV ecosystem with a service chain that optimizes video traffic through the Video Service Path, and routes all other traffic flow through the Regular Service Path. The VNFs of the service are deployed on top of the virtualized infrastructure (NFVI) by the orchestrator with the help of VNF Managers and virtual infrastructure managers. The NFV-MANO also helps in the life-cycle management including monitoring the health of the VNFs and NFVI, and taking the appropriate remedial actions.

The typical approach to rehoming an entire service chain involves applying the same corrective action, such as VM live migration, to all VNFs in the chain [11], [12]. In this report, we contend that this approach is too simplistic, and argue that it may be more beneficial to apply heterogeneous rehoming actions for different VNFs in a service chain. For example, in Figure 1, to address a hotspot in one of the physical servers that hosts video optimizer VNFs, all the resident VNFs can be live-migrated to different hosts. However, knowing that the video optimizer VNFs are stateless, rebuilding them (which recreates VMs and re-routes traffic) may be a quicker option. In Section II, we validate our intuition with a real experiment that illustrates how for a simple service chain consisting of a firewall and a web server the optimal rehoming action is in fact to migrate the firewall and rebuild the web server VM. Based on this intuition, this report aims to address the following question – "*What is the optimal rehoming action for each VNF in a service chain?*".

In particular, given a service chain and a set of potential rehoming actions, our goal is to determine the exact rehoming action that must be employed for *each* VNF in the chain;

different VNFs may have a different optimal rehoming action.

To the best of our knowledge, no prior work has addressed the question of identifying the optimal set of rehoming actions for a service chain. While most of the prior work ([11], [12], [13], [14], [15], [16], [17], [18]) has focused exclusively on live migration, and on the effect of rehoming on *individual* VMs, we show that the rehoming action of one VNF can impact the rehoming cost of other VNFs in a service chain. Also, unlike traditional performance metrics (e.g., end-to-end service latency) which measure the overall impact to the user, we focus on metrics that effectively capture the impact of rehoming actions. Note that our focus is not on developing new rehoming actions or on enhancing existing rehoming actions, but on determining *which of the available actions to employ for each VNF in the service chain*. This is a challenging task for several reasons:

- The candidate set of rehoming actions for a chain grows exponentially with the number of VNFs. For example, in Figure 1, if we only consider rebuild, cold-migrate and live-migrate as available rehoming actions, there are 3^5 possible rehoming choices for the service chain with 5 VNFs.
- The optimal rehoming actions for a chain depend on the exact metric(s) being considered. Even for a single VNF, optimizing for time taken to complete the rehoming (or *rehoming delay*) can lead to a choice which is different from the choice for optimizing the service *connectivity downtime*.
- The optimal rehoming action(s) can change with the state of the VMs (e.g., the image size or disk size) *and* the state of the underlying platform (e.g., total traffic).

We address these challenges through a novel Model-driven Rehoming Technique, MERIT, that determines the optimal rehoming actions for the *entire VNF chain*, based on four key features: (1) *Model-driven*: developing VNF-agnostic models for each rehoming action based on detailed empirical studies that capture the impact of various parameters on connectivity downtime and rehoming delay; (2) *Contention-aware*: accounting for the resource contention that results when rehoming the entire service chain as multiple rehoming actions migrate state simultaneously across the network and on shared storage; (3) *Graybox*: leveraging information exchange (e.g., statefulness of VNFs) between tenants and network provider, typically common in such private clouds, to identify the feasible rehoming actions for each VNF, thus reducing the state space of candidate actions; and (4) *Autonomous*: employing an end-to-end implementation to automatically rehome VNFs and continually update cost models at runtime. Based on these features, we make the following contributions in this report:

- ***Looking beyond live migrate*** *(Section II):* In sharp contrast to prior works that only focus on live migrate, we show that *rebuild and cold migrate* could be potentially superior alternatives, especially for memory-intensive VNFs or non-shared storage environments where live migrate is infeasible.
- ***Empirical analysis of rehoming costs*** *(Section III):* We perform measurement studies on an OpenStack cluster to capture the impact of system parameters on the rehoming cost for different out-of-the box rehoming actions on both shared and non-shared storage configurations. Our analysis yields crucial insights that guides our MERIT design. For example, while rehoming delay is typically hundreds of seconds lower for rebuild than for cold/live migrate, the connectivity downtime can be several tens of seconds higher for rebuild, especially for low VNF disk sizes. Likewise, while live migrate typically provides very low connectivity downtime, it can incur a substantial rehoming delay of several minutes, and may even time out.
- ***Modeling of rehoming costs*** *(Section IV):* Using our measurements, we leverage supervised learning to model the delay and downtime for different rehoming actions. Our modeling efforts reveal that while simple regression models work well for rebuild and cold migrate, more complex models, e.g., neural networks, are needed for live migrate.
- ***Implementation and evaluation on real-world service chains*** *(Sections V and VI):* We implement an (open-source [19]) autonomous rehoming system that leverages empirically trained models and interacts with OpenStack to execute optimal rehoming on service chains. Our system obtains service and VNF models from the service provider inventory, and uses the empirically trained rehoming cost models to predict the rehoming delay and connectivity downtime. Finally, it interacts with OpenStack orchestrators to execute optimal rehoming on service chains. Our experimental evaluation on real-world service chains shows that MERIT accurately predicts the rehoming costs for the entire chain by accounting for the network contention created by simultaneously rehomed VNFs. Compared to the existing practice of applying homogeneous rehoming actions for all VNFs in the chain, MERIT reduces the chain rehoming delay by about 26% on average (and up to 47%) and the chain downtime by about 20% on average (and up to 49%). Importantly, the chain rehoming costs incurred by MERIT are almost always within 10% of the costs incurred by the unrealistic but optimal clairvoyant (oracle) policy.

II. Background and Overview

Network service providers often rehome VNFs of the service from one host to another in response to infrastructure dynamics like hardware failures and hot-spots. In this section, we first describe the different rehoming actions we consider for MERIT, along with their pros and cons, especially with respect to the action's time-to-completion, or **rehoming delay**, and the service **connectivity downtime** it induces; we collectively refer to the rehoming delay or connectivity downtime as the **rehoming costs**. Then, through an experiment on a real testbed, we illustrate how these rehoming actions may have a different time-to-completion, or rehoming delay, for different VNFs in the service chain, and may impact the service connectivity downtime differently, thereby motivating the need for a model-driven approach in MERIT. We next describe the different rehoming actions we consider for MERIT, along with their pros and cons, and then discuss the prior work on the analysis and evaluation of different rehoming actions.

When analyzing the rehoming performance, we consider the following metrics, collectively referred to as *rehoming costs*:

- **Rehoming delay:** This is the time to perform the rehoming action, and represents the (resource) cost incurred by the provider to rehome a VNF.
- **Connectivity downtime:** This is the time until the service chain's end-to-end connectivity is restored, and represents the performance loss for the tenant due to rehoming.

Cloud platforms typically provide out-of-the-box mechanisms like rebuild, migrate, etc., that help with rehoming; we consider three such practical rehoming actions, as described below. While variants of rehoming actions are possible, such as rebuilding on a different host or migrating to the same host or live migrating only a fraction of the memory state. our focus is on determining the optimal VNF rehoming actions which are practically used by service operators, from among the available actions, and not on optimizing the rehoming actions themselves. Nonetheless, our methodology is not specific to the considered actions and can be applied to cases where additional rehoming actions are available. We thus employ the actions described above that are already available in OpenStack. In the following, we refer to the VM prior to rehoming as the "original" VM and the VM after rehoming as the "rehomed" VM.

1) **Rebuild**[1]**:** This involves taking down the original VM and rebuilding it from the VM's image while retaining some metadata (e.g., IP address and interfaces) [20]. Rebuild has *low rehoming delay*, but it can only be used for stateless VNFs, since the disk and memory state of the original VM are not preserved, resulting in *loss of state*.
2) **Cold migrate:** This involves migrating a VM with its disk contents [21]. By default, only disk contents are copied to the rehomed VM. The in-memory state of the original VM can be optionally restored on the rehomed VM by writing to disk prior to migration. Thus, cold migrate *does preserve some state*. However, it is a relatively slow rehoming action as the *rehoming delay under cold migrate can be high*, especially for a large disk size. and/or low network bandwidth.
3) **Live migrate:** This action migrates an *active* VM instance to a different host, and *tends to induce minimal disruption (connectivity downtime)* to the hosted application [11]. However, live migration may incur variable rehoming delay, depending upon many factors including the storage configuration (shared vs non-shared), memory page dirtying rate, etc. Live migration (using "pre-copy") involves a warm-up phase where memory pages are copied from the source to destination host before starting the rehomed VM, and a stop-and-copy phase where the original VM is suspended and the remaining dirty pages are copied over to the rehomed VM. Note that live migrate *requires shared storage* to be feasible (Section III-C3). Live migrate does not shut down the original VM during rehoming (thus consuming the resources of *both* hosts) and migrates the memory contents until a specified threshold. *Live migration can take a long time to complete*, especially for applications with a high page dirty rate [22]. In such cases, live migration can be aborted via a timeout feature [23].

[1]The default OpenStack implementation only rebuilds within the same host; we use a combination of delete and boot to allow rebuilding the VM on a different host.

III. Empirical analysis of rehoming

Our problem statement is as follows: *Given the various rehoming actions, when a service chain needs to be rehomed, which rehoming action should be employed for each VNF in the chain to optimize rehoming costs?* The "optimize" here refers to minimizing the connectivity downtime and/or rehoming delay, as specified by a user-provided objective or utility function.

To help address the above question, we empirically analyze the rehoming costs for different rehoming actions to understand the trade-offs between them. Clearly, it is infeasible to empirically analyze all possible VNF chains. Even for a single VNF being rehomed, the empirical analysis is complicated by the complex interactions between the different rehoming actions and the cloud environment. For example, while rebuild is relatively unaffected by the VNF disk size or available bandwidth, cold and live migrate are affected by these features. In the multi-VNF service chain setting (Section IV-D), the problem of identifying the optimal rehoming action for each VNF in the chain is further complicated by the fact that the various rehoming actions, when executed simultaneously, can create resource contention, potentially amplifying the rehoming costs of each VNF. In this section, we start by comprehensively analyzing a simple VNF chain under various parameter settings and then extrapolate our results, in Section IV, to arbitrary settings and VNF chains. For our empirical analysis, we measure (i) rehoming delay based on the relevant timestamped entries in the OpenStack logs and/or from the Nova status API [24], and (ii) connectivity downtime by calculating the delay between successful pings from the client to the server in the chain; this delay includes the time to get console access to the VM. In general, the rehoming cost can be expressed as a utility function in terms of these two individual metrics.

A. Experimental setup

Our experimental test-bed comprises of several bare metal servers with Intel E5-2683 CPUs and 256GB memory in CloudLab (Clemson site) running OpenStack (Rocky release). We deploy our service chains on VMs (running Ubuntu 18.04) hosted on this test-bed. Figure 2 shows the experimental setup along with the simple VNF chain used for empirical analysis. Each VM has multiple NICs and IP forwarding enabled in order to route traffic through the chain; static routes are configured on each VM to properly route traffic. The corresponding ports for these NICs are also configured in Neutron [25] to allow traffic pass through [26]. For shared storage, we employ GlusterFS (v4.1) as our shared network

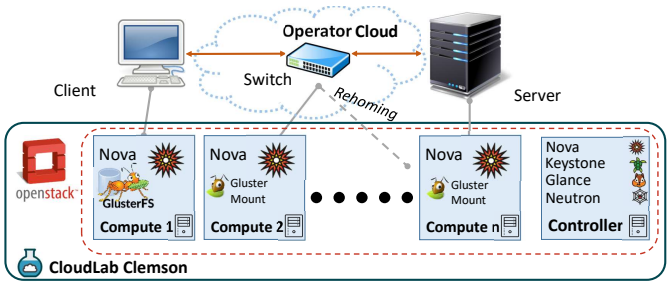


Fig. 2: Illustration of our service chain used for empirical analysis comprising a client VM, a switch VM, and a server VM. The VMs are hosted via OpenStack on top of CloudLab physical servers.

filesystem over ext4 using a single brick Gluster volume to avoid consistency issues.

B. Experimental methodology

We collect empirical data for analysis on a simple VNF service chain consisting of a client VM, a switch VM, and a server VM as shown in Figure 2, with ping traffic going from client to server via the switch. We consider the client and the server VM to be outside the provider cloud, and only rehome the switch VNF under various configurations. We analyze rehoming costs separately for both shared and non-shared cloud storage.

For the empirical analysis in this section, we rehome the switch VM under various configurations; we expect similar results for other VMs as well, though the post-bootup process may be different, depending on the VNF's functionality.

VNF parameters. We experiment with image sizes of 250MB–3GB and disk sizes of 700MB–5GB (by adding software and data). We consider different VM sizes (parameterized by their memory capacity): 4GB (medium), 8GB (large), and 16GB (xlarge). We also vary the page dirty rate (PDR) by running Memcached on the switch VM and Mutilate [27]load generator on a different VM to send traffic to Memcached. We vary the request rate and data store size for Memcached to generate different data points for PDR and working set size, respectively. To track PDR, working set size, and other relevant features, we use a modified QEMU with additional profiling capabilities [22].

Cloud Infrastructure parameters. We consider different infrastructure parameters such as available network bandwidth, VM CPU usage, and I/O contention. To study the impact of available bandwidth on rehoming, we employ WonderShaper [28] to limit the bandwidth to 100-900Mbps; the link capacity in our setup is 950Mbps. We vary VM CPU utilization by modifying the request rate of Memcached; this also impacts PDR. For I/O contention, we use stress-ng [29] to generate different background I/O loads at the target host. We use mpstat and iostat to measure I/O statistics such as the mean and standard deviation of the I/O wait percentage, the average I/O operations per second (IOPS), and the total kilobytes read and written per second.

C. Empirical analysis of rehoming costs

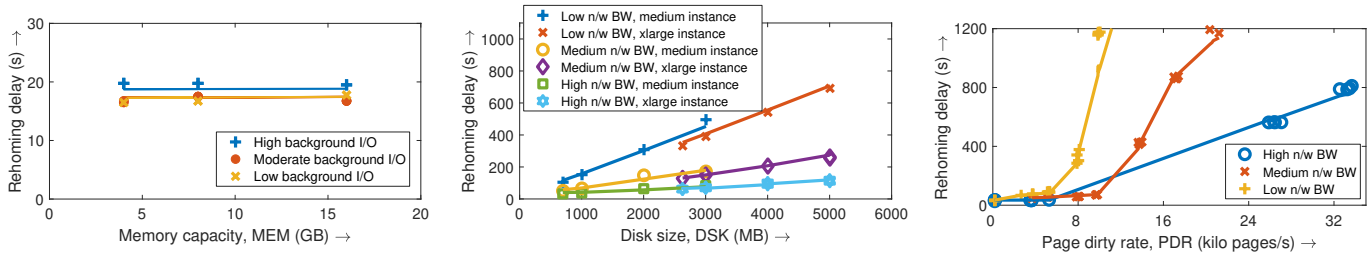
We use the VNF chain shown in Figure 2 and experiment with all three rehoming actions applied on the switch VM. All results are based on the experimental setup described in Section III-A; we report the results averaged across 4 runs. We now present our empirical results (key takeaways in Section III-D), starting with the results under shared storage.

1) Analysis of rehoming delay

The markers in Figure 3 show our empirical results for the rehoming delay of all three actions under shared storage. In each case, we plot the rehoming delay as a function of one of the parameters, and vary the other parameters to generate different sets of data points (markers). The solid lines are our modeling results, and will be discussed in Section IV. For ease of presentation, we only show a few dimensions/features in the 2-D plot, but discuss all relevant findings in text. Figure 3(a) shows the rebuild rehoming delay as a function of instance size, parametrized via the memory capacity of the instance (MEM). We show results under a 1GB image size and 335Mbps bandwidth; results are qualitatively similar for other parameter settings. We see that the rehoming delay under rebuild is quite small, almost always around 20s. We also see that the rehoming delay is largely insensitive to the instance size, and is only slightly impacted by the mean background I/O load, with higher load contributing to higher rehoming delay. This is likely because the background I/O load on shared storage contends with the disk read and write operations required for the booting process that are part of the rebuild action on the target host. The other parameters we experiment with, including network bandwidth, disk size, etc., did not much affect the rebuild rehoming delay; this is to be expected as there is no disk transfer under rebuild. We also did not find any noticeable impact of other parameters, such as VM CPU usage and page dirty rate, on the rebuild rehoming delay.

Figure 3(b) shows the cold migrate rehoming delay as a function of disk size for different combinations of background network bandwidth (BW) and instance size. We see a nearly linear relationship between rehoming delay and disk size, and find that delay increases as BW decreases. This is because, under OpenStack, the source VM's disk contents are copied onto shared storage over the network before migration is considered complete. The rehoming delay under cold migrate can be quite high, as much as 700s, for larger disk sizes. Other parameters did not significantly impact the cold migrate rehoming delay, and are thus omitted.

Figure 3(c) shows the live migrate rehoming delay as a function of PDR for different network bandwidths (BW). While we also experiment with other feature values, we find empirically that PDR and BW have a higher impact on delay. As discussed in Section II, live migration does not migrate the image or disk contents due to the shared storage requirement. Instead, live migrate copies over the memory contents from the original to the rehomed VM, including any pages that were modified during the rehoming delay duration. To prevent live migration from getting stuck, we set a timeout of 20 minutes;

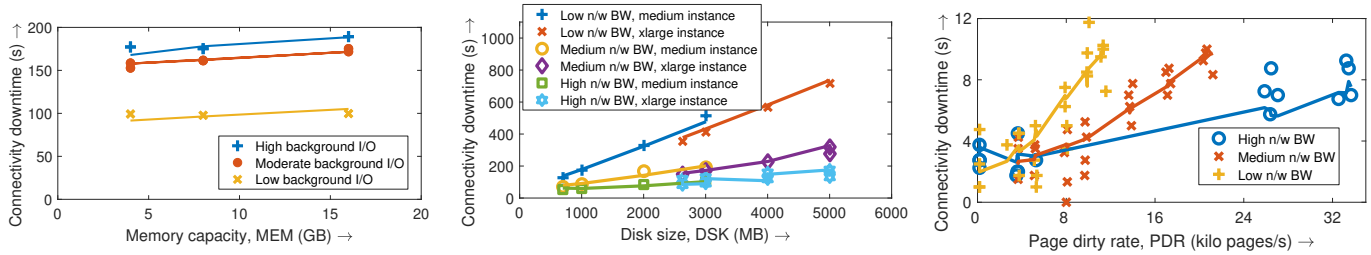


(a) Rebuild rehoming delay vs. instance capacity for different background I/O loads.

(b) Cold migrate rehoming delay vs. disk size for different instance sizes and background I/O loads.

(c) Live migrate rehoming delay vs. page dirty rate for different network bandwidths.

Fig. 3: Empirical results for rehoming delay of rebuild, cold migrate, and live migrate under shared storage.



(a) Rebuild connectivity downtime vs. instance capacity for different background I/O loads.

(b) Cold migrate downtime vs. disk size for different instance sizes and background I/O loads.

(c) Live migrate connectivity downtime vs. page dirty rate for different network bandwidths.

Fig. 4: Empirical results for connectivity downtime of different rehoming actions under shared storage.

this is more than the time it takes live migrate to complete with a stop-and-copy phase duration of 5s [23]. We see that the rehoming delay under live migration can be quite high for moderate to high PDR, often exceeding the peak rehoming delay under rebuild and cold migrate. This is because the live migrate process has to continually copy pages as they get dirtied. In fact, for high PDR, we see that live migrate times out (shown as rehoming delay of 1200s in Figure 3(c)); thus, for high PDR, live migrate is infeasible. At low PDR, the rehoming delay is in the 25–90s range (higher than rebuild but lower than cold migrate). In terms of trend, the delay increases with PDR; further, the delay also increases with a decrease in network bandwidth.

2) Analysis of connectivity downtime

The markers in Figure 4 show our empirical results for the connectivity downtime of all actions under shared storage. The results in Figure 4 are from the same experiments as Figure 3, and thus the VNF and system parameters are the same.

Figure 4(a) shows the connectivity downtime for the rebuild action. We see that downtime follows the same trend as delay, except that there is now a slightly linear relationship between downtime and instance size, and the downtime is more sensitive to background I/O load. In general, the downtime numbers for rebuild are in the 100–200s range, with higher values for higher background I/O load. The downtime is higher than delay under rebuild because restoring the connectivity requires some post-boot processes to execute, such as network configuration and host-ssh key generation (because in rebuild the rehomed VM's boot-up is a first boot). By contrast, rehoming delay does not include these post-boot processes, and is considered complete once OpenStack reports the rebuilt VM as "Active" on the target host.

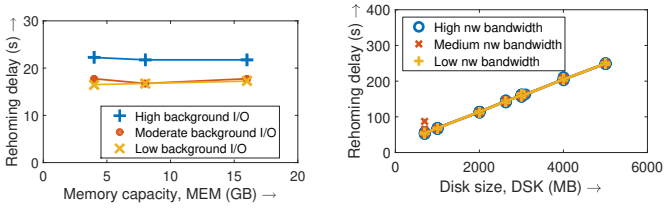
Figure 4(b) shows the connectivity downtime for cold migrate. The trends are very similar to those in Figure 3(b). For cold migrate, the downtime and delay values are not very different, unlike rebuild; this is because both rehoming costs include the time needed to copy the disk. Nonetheless, downtime is typically higher under cold migrate than rebuild, especially for larger disk sizes.

Finally, Figure 4(c) shows the downtime for live migrate. In stark contrast to rebuild and cold migrate, the downtime under live migrate is much shorter. While we see a loosely linear correlation between downtime and PDR, the downtime is less than 12s in all cases. This is because downtime only includes the stop-and-copy phase wherein the source VM is stopped and only the remaining dirty memory is copied to the target VM. *However, there is a caveat here*. Since live migrate times out under high PDR, the rehoming does not complete in such cases; consequently, there is no downtime to be reported.

3) Analysis of rehoming costs under non-shared storage

The rehoming cost results for rebuild and cold migrate under non-shared storage do not change significantly when compared to shared storage. Note that live migrate is infeasible under non-shared storage.

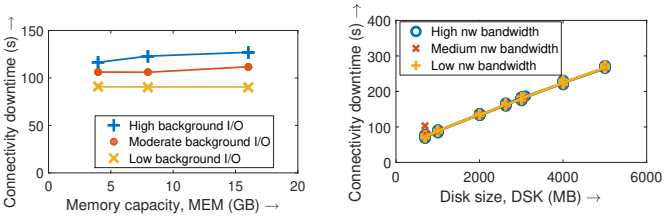
For live migrate under non-shared storage (known as block live migrate), we find that the action time often times out, as both disk and memory need to be transferred with active dirtying. Similar conclusions have been made by prior work [30], [15]. Further, the block live migrate implementation in OpenStack is restrictive, only allowing the EXT storage



(a) Rebuild delay vs. memory capacity for different I/O loads.

(b) Cold migrate delay vs. disk size for different network bandwidths.

Fig. 5: Empirical results for rehoming delay of rebuild and cold migrate under non-shared storage.



(a) Rebuild downtime vs. memory capacity for different I/O loads.

(b) Cold migrate downtime vs. disk size for different network bandwidths.

Fig. 6: Empirical results for downtime of rebuild and cold migrate under non-shared storage.

repositories and disallowing any attached volumes [31]. We thus omit live migrate from the list of feasible rehoming actions under non-shared storage.

Figures 5 and 6 show the rehoming delay and connectivity downtime, respectively, for rebuild and cold migrate under non-shared storage. With non-shared storage, we expect higher rehoming costs for actions that require data movement, such as cold and live migrate. We thus do not see a significant difference between the rehoming costs for rebuild under shared and non-shared storage.

For cold migrate, in Figures 5(b) and 6(b), we see a nearly linear relationship between the rehoming costs and disk size. This is because a significant amount of time during cold migrate is spent in migrating the disk. However, unlike for shared storage, we now see that the rehoming costs are largely insensitive to network bandwidth. In our setup, the cold migrate state transfer was unable to exploit the network capacity, and was thus not much affected by the bandwidth.

D. Key takeaways

- While the delay is typically higher for cold migrate than for rebuild (due to additional disk state migration under cold migrate), the *downtime can be higher for rebuild than cold migrate*, especially for smaller disk sizes. We see a similar tradeoff under non-shared storage. This is because rebuild necessitates post-boot (re)configuration, unlike cold migrate.
- Under shared storage, if live migrate does not time out (low PDR), live migrate is the optimal choice for downtime.
- Under shared storage, even with negligible PDR, the rehoming delay is typically higher for live migrate when compared to rebuild. At moderate PDR, even cold migrate can result in lower rehoming delay than live migrate.
- Under non-shared storage, live migrate is infeasible.

IV. Modeling the Rehoming Costs

This section first presents our modeling results for a single VNF, and then, in Section IV-D, we show how MERIT leverages the single VNF models to predict the optimal simultaneous rehoming actions for the entire service chain. We later show, in Section V, how we leverage the single VNF models developed in this section to accurately predict the optimal rehoming actions for *multiple* VNFs in the service chain, where the decision is complicated by the resource contention due to simultaneous rehoming of VNFs.

A. Modeling methodology

Our approach to modeling the rehoming costs is to consider a simple VNF under different configurations for training, and build service agnostic models that are widely applicable. We consider the simple client-switch-server VNF chain and focus on modeling the rehoming costs for the switch VM, which was analyzed in Section III.

1) Learning techniques

To build the rehoming cost models, we employ supervised machine learning techniques. We employ multiple linear regression (LR), support vector regression (SVR), and neural networks (NN) to train our rehoming delay and connectivity downtime models.

- **Linear Regression (LR):** is a statistical technique that models the dependent variable (rehoming cost) as a linear weighted combination of the independent variables [32].
- **Support Vector Regression (SVR):** finds a hyperplane that minimizes the number of data points that lie beyond a certain threshold (ϵ). SVR can use kernel functions to map the input space to a higher dimensional feature space and can thus perform non-linear regression as well [33]. We use the Radial Basis Function (rbf) in our SVR models [34].
- **Neural Networks (NN):** is a learning algorithm that learns how to best combine the features, possibly in a non-linear manner, using adaptive weights, to estimate the dependent variable. We specifically employ a single hidden layer in our feed-forward network. We consider two different activation functions for the hidden layer in our NN models: (i) sigmoid, and (ii) rectified linear unit (ReLU). Sigmoid function is a special case of logistic function with a range of [0,1], and hence does not amplify the activations [35]. ReLU is a function defined as the positive part of its argument with range [0, ∞). An advantage of ReLU is that it greatly accelerates the convergence of stochastic gradient descent compared to the sigmoid function [36], [37]. For more details, see Haykin [38].

2) Features used for model training

We use a different feature set for each rehoming action, based on the exact mechanism involved in the rehoming and based on our empirical analysis from Section III-C. However, to model a given action, we employ the same features for all learning techniques and rehoming costs (delay and downtime). We now discuss the features for our model training and how

we obtain them; a listing of the features is provided in Table I. Details of how we leverage the model and features at runtime are discussed in Section IV-D.

For rebuild, we use the following features: (i) image size of the original VM (IMG), (ii) instance size, denoted by its memory capacity (MEM), (iii) available bandwidth (BW), (iv) mean I/O wait time (IO_μ), and (v) standard deviation of I/O wait time (IO_σ). The intuition for including these specific features is that rebuild involves booting a new VM using the image, and as we observed in our empirical analysis, the instance size and I/O load can impact the rehoming cost under rebuild. We also include system-level features, CPU and BW, to investigate any impact they may have on the rehoming costs. For training, we explicitly limit the network bandwidth of the target OpenStack hypervisor that the VM will be rehomed to, and we use this limit as the value for the available bandwidth feature. When applying the model in practice (testing), we estimate available bandwidth at runtime, as discussed in Section IV-D.

For cold migrate, in addition to the above features, we also use the disk size (DSK) since cold migrate involves moving the disk contents. To capture I/O contention, we also use the IOPS and the read and write kBps as features.

For live migration, there are prior works that have investigated the set of useful features for modeling [11], [22], and so we leverage these results to finalize our feature set as: (i) instance size (MEM), (ii) network bandwidth (BW), (iii) page dirty rate (PDR), (iv) working set size (WSS), (v) modified words per page (MWP), (vi) working set entropy (WSE), and (vii) non-working set entropy (NWSE). For details about these features, refer to Jo et al. [22]. The live migrate specific features, PDR, WSS, MWP, WSE, and NMWSE, are measured periodically, every second, and we use the most recent measurements (just before rehoming) as the feature values for training. To capture I/O contention, as before, we also use the IOPS and the read and write kBps as features. Note that live migration is more complex than the other rehoming actions, and thus we require several features. The intuition for including these features is to capture the memory state transfer time under live migration, which is the only state to be migrated since image and disk are on shared storage. While additional features such as CPU usage and memory utilization of source and destination hosts can also be considered, we found that these features do not improve model accuracy, and add to the complexity of the model.

The intuition for including these features is to capture the memory state transfer time under live migration, which is the only state that needs to be migrated since image and disk are on shared storage; recall, from Section III-C, that under non-shared storage, live migrate is not practical.

B. Modeling results for rehoming costs

Our models employ the empirical data collected in Section III for training. We have about 400, 1600, and 1000 empirical data points for rebuild, cold migrate, and live migrate, respectively. In all cases we remove outliers, and in the case of live migrate, we omit data points where live migrate times out.

Variable	Description (with units)
IMG	Image size (MB)
DSK	Disk size (MB)
MEM	Memory capacity of the VM (GB), used as a (numerical) proxy for the VM instance size
CPU	VM CPU usage (percentage)
BW	Network bandwidth (Mbps)
IO_μ	Mean of (fractional) I/O wait time
IO_σ	Standard deviation of (fractional) I/O wait time
PDR	Page dirty rate (pages dirtied/sec)
WSS	Working set size (pages)
MWP	Modified words per page (words/page)
WSE	Working set entropy (bytes)
NWSE	Non-working set entropy (bytes)

TABLE I: List of features used in our modeling.

Model	Rebuild		Cold Migrate		Live Migrate	
	delay	dtime	delay	dtime	delay	dtime
LR	4.2%	2.8%	5.6%	4.4%	92.6%	35.4%
SVR	4.1%	3.8%	11.2%	7.7%	27.8%	33.7%
NN sig	4.6%	2.0%	4.4%	3.7%	11.0%	34.6%
NN ReLU	4.3%	2.0%	4.4%	3.9%	13.0%	36.6%

TABLE II: 5-fold cross validation error for different modeling techniques under shared storage.

Model	Rebuild		Cold Migrate	
	delay	dtime	delay	dtime
LR	5.0%	4.7%	1.4%	1.6%
SVR	5.3%	2.6%	3.3%	2.8%
NN, sigmoid	5.6%	2.1%	1.1%	1.3%
NN, ReLU	5.4%	2.6%	1.1%	1.1%

TABLE III: 5-fold cross validation error for different modeling techniques under non-shared storage.

The average 5-fold cross validation errors for all rehoming actions under all learning techniques for shared and non-shared storage are shown in Tables II and III, respectively. In general, we find that SVR and NN typically have higher accuracy than LR, especially for cold migrate and live migrate; in fact, for live migrate, LR has very poor accuracy, suggesting the need to employ a non-linear model for predicting the rehoming costs of live migrate. For rebuild, since the empirical data exhibits a nearly linear relationship, LR performs equally well.

For rebuild, we find that the feature weights for instance size and image size for our empirical data are negligible. For cold migrate, disk size and bandwidth have significant weights. For live migrate, PDR and bandwidth have significant weights, whereas WSE and NWSE have negligible weights.

C. Final models employed for MERIT

Based on the final results in Tables II and III, *we choose LR models for rebuild and cold migrate* rehoming costs. While LR has slightly worse accuracy compared to other techniques for cold migrate, LR provides intuitive, closed-form expressions for the final models, and LR is quick to train. Further, the LR model is quick to train; for our empirical data, we train our LR (and SVR) model in about 0.4 ms; the NN models require about 22–26 ms to train; note that these numbers scale with

the amount of data, and the number of neurons in the NN. However, for live migrate, LR has poor accuracy, making it an impractical choice for MERIT. similar observations about the difficulty of modeling live migrate using linear regression techniques were made by prior works [11], [22]. Instead, *for live migrate, we choose NN with ReLU activation function*. Note that, for live migrate, the downtime modeling accuracy is not important as the downtime is almost always less than 12s (and thus superior to rebuild and cold migrate), except when live migrate times out under high PDR, making live migrate infeasible (see Section III-C2). The timeout event can be predicted by comparing the NN rehoming delay prediction with the timeout value (1200s, in our case).

The final LR models for rebuild and cold migrate are shown as the solid lines in Figures 3(a), 3(b), 4(a), and 4(b). For live migrate rehoming costs, the NN with ReLU model is shown as the solid lines in Figures 3(c) and 4(c).

The lower modeling accuracy for live migrate is in agreement with prior modeling efforts that reveal the difficulty in modeling live migration time [11], [22], [39].

D. Modeling network contention when applying MERIT

To apply MERIT, we monitor all relevant VNF parameters, such as image size, disk size, available bandwidth, page dirty rate, etc. Some of these parameters, such as page dirty rate, are monitored periodically (up until the rehoming action is invoked) as they are time-varying parameters. The static parameters, such as image and disk size, can be easily obtained from the hypervisor through Orchestration Wrapper. VM deployment logs. At runtime, when the rehoming is to be performed, we leverage the most recent monitoring information as input to our rehoming delay and connectivity downtime models from Section IV, and choose the optimal rehoming actions for each VNF in the chain.

To get the optimal set of rehoming actions, MERIT generates the possible combinations of feasible actions for the chain. Then for each combination, we use the learned models to predict the rehoming delay and connectivity downtime for each VNF, taking into account the resource contention created by simultaneous rehoming actions. Then, a feature vector is created for each VNF based on the VNF characteristics and runtime infrastructure metrics (some metrics are estimated, see Section IV-D), which is used as input for our learned models. The models predict the rehoming delay and connectivity downtime for each VNF in the combination as output, taking into account the resource contention created by simultaneous rehoming actions. The predictions are then used to calculate the chain rehoming delay, R, and chain downtime, D, for each combination. Finally, MERIT picks the rehoming actions combination that is predicted to maximize or minimize a given generic utility function, $U(R, D)$; we use $U(R, D) = R \cdot D$ as an example utility function to be minimized in our evaluation.

Thus far we assumed that available network bandwidth (BW) is a feature that can be easily obtained. At run-time, due to simultaneous rehoming of VNFs in the chain, the available network bandwidth (BW) is shared among them, and can be under contention. Thus, *BW for each VNF rehoming action must be estimated at run-time* when applying the rehoming models. We predict BW online based on the number of VNFs in the chain and the rehoming action being applied for each VNF; we can also account for background traffic by subtracting that amount from the total bandwidth.

Let the available network bandwidth at the host be B MB/s. We can account for background traffic by subtracting that amount from the available bandwidth. If the chain has n VNFs on a host, then the available bandwidth for each will be B/n, assuming they have the same amount of state to be migrated. If the amount of state to be transferred is different, then the bandwidth computation is more complex. Consider a chain with two VNFs being rehomed, with the first VNF requiring a state migration of x_1 MB, and the second VNF requiring a state migration of $x_2 > x_1$ MB. Then, assuming fair sharing, we estimate BW for VNF 1 to be $B_1 = B/2$ MB/s during its state migration time of $T_1 = \frac{x_1}{B/2}$ seconds. For VNF 2, it has available bandwidth of $B/2$ for time T_1 (during which it also migrates x_1 MB of state) and bandwidth B for time $(x_2 - x_1)/B$. Thus, state migration time for VNF 2 is $T_2 = T_1 + (x_2 - x_1)/B = \frac{x_1+x_2}{B}$ seconds, and the time-averaged BW is $\frac{x_2}{T_2} = \frac{x_2 \cdot B}{x_1+x_2}$ MB/s. In general, for a host with n VNFs, with the VNFs indexed in increasing order of state migration size $x_1 < x_2 < \ldots < x_n$, the state migration time and BW for the i^{th} VNF are:

$$T_i = \frac{x_1}{B/n} + \frac{x_2 - x_1}{B/(n-1)} + .. = \frac{\sum_{j=1}^{i-1} x_j + (n-i+1) \cdot x_i}{B} \quad (1)$$

$$B_i = \frac{x_i}{T_i} = \frac{x_i \cdot B}{x_1 + x_2 + \ldots x_{i-1} + (n-i+1) \cdot x_i} \quad (2)$$

Note that T_i is not the same as rehoming delay since the latter may include additional delays due to the rehoming process specifics (see Section III-C). The model training captures these additional delays as a function of the features (see Section IV-A2). The state size, x_i, depends on the rehoming action to be performed on VNF i. For rebuild, there is no state transfer involved. For cold migrate, the disk contents are transferred over the network. For live migrate, under shared storage, the memory contents, iteratively dirtied pages, and the final dirty memory during stop-and-copy phase comprise the state to be migrated; the size of the state can be estimated based on the PDR and available network bandwidth. Under non-shared storage, additionally, the disk and image are also transferred; the disk and image size can be obtained via hypervisor logs.

V. System Design and Implementation

Figure 7 shows the system implementation of our MERIT approach. While MERIT includes an offline component which constructs the rehoming action models using empirical data, the figure only shows the online components. We implement the system in Python and bash, consisting of ∼1200 lines of open-source code [19].

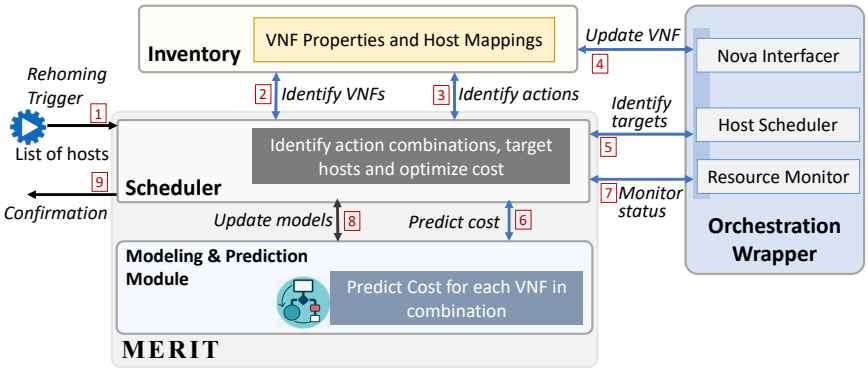


Fig. 7: Illustration of our MERIT system implementation.

The NFV Infrastructure (NFVI) monitoring systems (e.g, Ceilometer in OpenStack [24]) trigger a rehoming event and specify the physical host(s) that need to be evacuated in response to the event (1). From the **Inventory**, MERIT identifies the VNFs that reside on these physical hosts, and obtains their features, such as image size, disk size, PDR, etc. (2), along with the feasible actions for each of the VNFs (3); note that this information is kept up-to-date in the Inventory through communication with the hypervisors via the **Orchestration Wrapper** (4). Based on the obtained VNF information, MERIT uses a Cartesian product to list all possible rehoming action combinations. The Host Scheduler in the Orchestration Wrapper selects target hosts for each VNF that have the most spare resource capacity (such as available host disk space or available host memory); more sophisticated policies can be employed based on service/VNF provider requirements. The selected host information is then sent to the **Scheduler**, along with the monitored host bandwidth information obtained via the Orchestration Wrapper (5). The Scheduler then forwards this information, along with the list of possible rehoming action combinations, to the **Modeling & Prediction Module** (6), which in turn employs the trained rehoming cost models (stored as sklearn [40] model objects) to obtain predictions of the rehoming cost of each combination.

Once the Scheduler receives the predicted costs, it picks the minimum-cost action combination to optimize a given, user-specified, utility function, $U(R, D)$, where R and D are the rehoming delay and connectivity downtime of the chain, respectively. Examples of such a utility function include a weighted sum, $U(R, D) = \alpha \cdot R + (1-\alpha) \cdot D$ (where the weights are specified by the user depending on their relative importance of R and D), or the product $U(R, D) = R \cdot D$, which we employ in our experimental evaluation. Since MERIT predicts the utility value for *all* feasible rehoming combinations, the minimum-cost choice from among these combinations will be the (theoretically) optimal rehoming action combination.

Finally, the Scheduler communicates with Orchestration Wrapper to call the Openstack Nova API to perform the optimal rehoming for each VNF in a separate thread, and waits for their completion (7). Upon completion, Scheduler sends a confirmation back to the trigger (9) and directs the Modeling & Prediction Module to update its cost models based on the new data obtained during this rehoming run (8). We update our rehoming cost models through retraining since sklearn [40] currently does not support incremental (online) training.

VI. Evaluation Results

We now present experimental results for evaluating the efficacy of MERIT. For our experimental evaluation of MERIT, we implement several VNF service chains which are built using real-world reference implementations of the VNFs from OPNFV [41][2]. Figure 8 and 9 show the VNF service chains used in our evaluation, These chains are representative of common network functionalities:

1) *Gateway-Internet Local Area Network (Gi-LAN) chain* comprises a client VM, packet gateway, firewall VM, IDS VM, switch VM, stream transcoder VM (FFMpeg [42]), cache VM (Apache Traffic Server [ATS] [43]), and a server VM. This chain, illustrated in Figure 9, is representative of a Gi-LAN and has two branches based on traffic type: (1) video stream traffic that is transcoded by FFMpeg, and (2) web traffic that is served through the ATS caching proxy.
2) *Intrusion detection system (IDS) chain* comprises a client VM, switch VM, IDS VM (Snort [44]), and server VM, and is representative of a network intrusion detection system.
3) *Web caching chain:* This chain comprises a client VM, firewall VM, cache VM (Apache Traffic Server [ATS], a caching proxy [43]), and server VM, and is representative of a web caching functionality.
4) *Firewall chain:* This chain comprises a client VM, switch VM, firewall VM, and server VM; thus, a client-switch-firewall-server chain. In our experiments, the firewall VM uses IPTables to enforce traffic rules.

For evaluation, we use VNF parameters (disk size, image size, network bandwidth availability) that are different from the training data used in Section VI.

We first describe our evaluation methodology and then discuss how to use MERIT in practice. We then present our model validation results and our evaluation results for MERIT.

[2]OPNFV is an Open source Platform under the Linux Foundation that facilitates the development of NFVs.

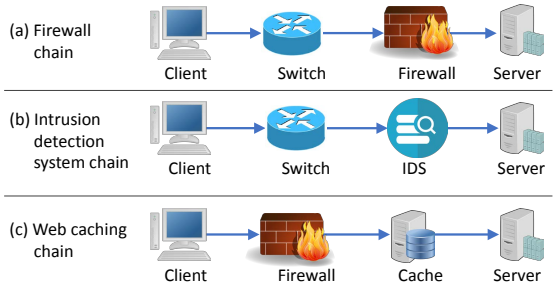


Fig. 8: Service chains used in our evaluation. VNFs are based on real-world reference implementations from OPNFV [41].

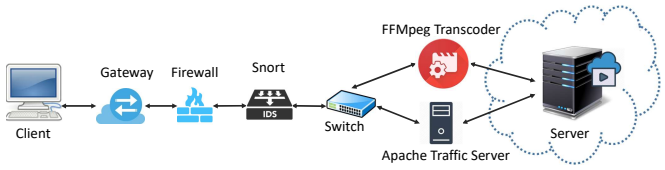


Fig. 9: Gi-LAN service chain used in our evaluation. VNFs are based on real-world reference implementations from OPNFV [41].

A. Evaluation methodology

In our experiments for VNF chain rehoming, we focus on the following chain-specific metrics:

- **Chain rehoming delay:** This is the sum of rehoming delay for all VNFs in the chain that are being rehomed, and represents the rehoming cost incurred by the provider.
- **Chain downtime:** This is the time until the service chain's end-to-end connectivity is restored, and is defined as the *maximum* connectivity downtime across all VNFs.

With simultaneous rehoming of the entire chain, the rehoming costs for a given VNF *now depend on the rehoming actions for the other VNFs in the chain as well*, because of the consequent network contention. Fortunately, our models already include features to track this contention. We refer to rebuild, cold migrate, and live migrate actions as RB, CM, and LM, respectively. Under our graybox approach, we consider that the network provider knows about the nature of stateful VMs (see Section II). For our chains, the stateful VMs are the firewall VM (due to IPTables) and the ATS cache VM (due to cached contents). For these stateful VMs, RB is not an option as it will result in loss of disk state. For ATS, even CM is not an option as the contents could be cached in memory. For memory intensive VMs like IDS and FFMpeg, the page dirtying rate can be very high; thus, LM (which will time out) is considered infeasible for these VMs. We experiment with all feasible rehoming action combinations for all VNF chains, with the stateful VNF exceptions noted above. For example, for the IDS chain, we consider 6 rehoming combinations, with the switch VM possibly being rebuilt, cold migrated, or live migrated, and the IDS VM being simultaneously cold migrated or live migrated. When rehoming a chain, to avoid additional connectivity downtime, we consider all VNFs of the chain to be rehomed *simultaneously*; this mimics a real deployment

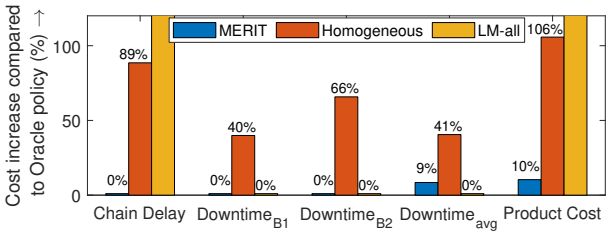


Fig. 10: Percentage increase in cost, relative to the Oracle policy, for MERIT and the best homogeneous policy for the Gi-LAN chain. Also shown, for completeness, is the infeasible live-migrate all policy.

where the *entire chain* needs to be rehomed in response to maintenance or failures. By employing the max function in the definition of chain downtime, MERIT optimizes for end-to-end connectivity downtime rather than per-VNF downtime local optima such as for LM. The client and server VM are typically outside the private cloud, so we do not rehome these VMs. For the rehoming, we assume that the target host is known (OpenStack decides the target host for migration); prior work has considered the problem of where to migrate [45], [46], which is orthogonal to our focus of determining the optimal rehoming actions. All experiments are run on the experimental setup described in Section III-A.

For each chain, we compare MERIT with the following:

- The **Oracle** policy applies the optimal rehoming actions at each VNF. This is an unrealistic but useful comparison baseline as Oracle knows the actual rehoming costs for each rehoming action combination a priori. We "implement" the unrealistic Oracle by experimenting with all feasible combinations and then labeling the minimum-cost optimal combination as the Oracle policy. Note that, by design, the rehoming cost of Oracle is indeed the optimal cost. Also note that the rehoming cost for MERIT may be worse than Oracle since the MERIT-predicted optimal actions may not be optimal in practice due to inaccuracies in predictions.
- The **Homogeneous** policy uniformly applies the same rehoming action across all VNFs that need to be rehomed. To implement this policy, we select the lowest cost feasible rehoming combination that applies the same action (from among RB, CM, and LM) for all VNFs to be rehomed.

B. Rehoming evaluation results

We start by evaluating the efficacy of MERIT's rehoming recommendations for the various VNF chains, using the two chain-specific metrics detailed in Section VI-A. For each VNF, we experiment with the feasible rehoming actions from among rebuild (RB), cold migrate (CM), and live migrate (LM).

1) Rehoming the Gi-LAN chain

For the Gi-LAN chain, the Firewall, Snort IDS, FFMpeg, and ATS VNFs are subject to rehoming while the gateway and switch are fixed. For the stateful firewall VNF (due to IPTables), RB is not a feasible option. Likewise, RB and CM are infeasible for ATS (since contents could be cached in memory) and LM is infeasible for memory intensive IDS and FFMpeg VNFs. Under our graybox approach, MERIT

only considers the remaining 8 feasible action combinations for the chain (CM and LM for firewall; RB and CM for IDS and FFMpeg). We also experiment with a homogeneous live-migrate all policy for better comparison. We use shared storage and medium instance size (2 cores, 4GB memory) for the above four VNFs, with 250MB image and 1GB disk for Firewall, 1.6GB image and 970MB disk for IDS, 330MB image and 700MB disk for FFM, and 850MB image and 500MB disk for ATS. We note that these configurations were not part of the model training data.

Rehoming cost results: Figure 10 shows the percentage increase in rehoming cost for MERIT and Homogeneous, relative to the Oracle rehoming cost. All reported results are averaged over 3 experimental runs. We consider the chain rehoming delay, chain downtime, and the product of chain rehoming delay and chain downtime (as an example of a utility function $U(R, D) = R \cdot D$). We also consider the branch-specific chain downtimes for the video traffic (branch $B1$) and web traffic (branch $B2$); the chain downtime is the average of the branch-specific downtimes. As the Gi-LAN chain constitutes two branches, there are two end-to-end connectivity downtime metrics, $B1$ for the transcoder branch and $B2$ for the web cache branch. Total cost is the utility function defined cost, and in our case it is the product of rehoming delay and averaged connectivity downtime of two branches.

We see that MERIT is almost always within 10% of the cost incurred by Oracle, and often has the same cost as Oracle. By contrast, the Homogeneous policy incurs a substantially higher cost for all metrics we consider, with an average cost increase of about 68% across all metrics. For the Gi-LAN chain, the Homogeneous policy employs CM for all rehomable VNFs (fireall, IDS, FFMpeg), except ATS since ATS can only be live-migrated. While various other combinations can be considered for comparison, we note that MERIT's cost relative to the Oracle policy demonstrates our superiority.

For completeness, we empirically evaluate the (infeasible) option of live-migrating all four rehomable VNFs; we refer to this policy as *LM-all* in Figure 10. *LM-all* is not considered in Homogeneous since LM is infeasible (times out) for IDS and FFMpeg VNFs due to high page dirty rate. Since some of the VNFs time out under *LM-all*, the rehoming never completes (infinite chain rehoming delay). While chain downtime increase is shown as 0%, note that the rehoming action times out and so the chain is never rehomed. Fortunately, MERIT is able to predict that LM will indeed time out for the FFMpeg and IDS VNFs, and so MERIT does not consider the *LM-all* option.

Prediction accuracy: The superior performance of MERIT can be attributed to its accurate rehoming cost prediction models. Recall that MERIT predicts the rehoming cost for all feasible action combinations and then executes the cost-optimal action combination. Figure 11 shows the actual and MERIT-predicted chain rehoming costs for all eight feasible rehoming combinations for the Gi-LAN chain. We see that

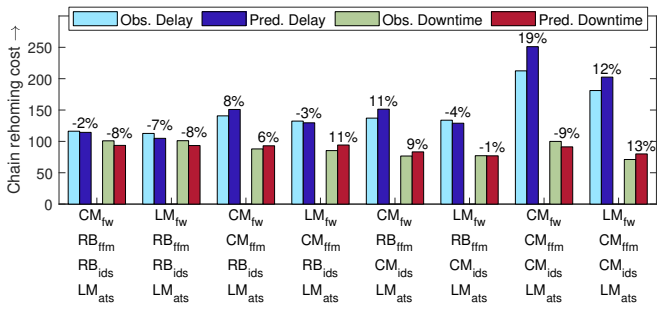


Fig. 11: Observed and model-predicted chain rehoming costs under various rehoming action combinations for the Gi-LAN chain. Numbers above the bars denote MERIT's model prediction error.

Cost Metric	Gi-LAN Chain	IDS Chain	Firewall Chain (L)	Firewall Chain (S)
Chain delay	7.7%	5.1%	12.2%	13.7%
Chain downtime	7.6%	8.2%	1.2%	10.7%

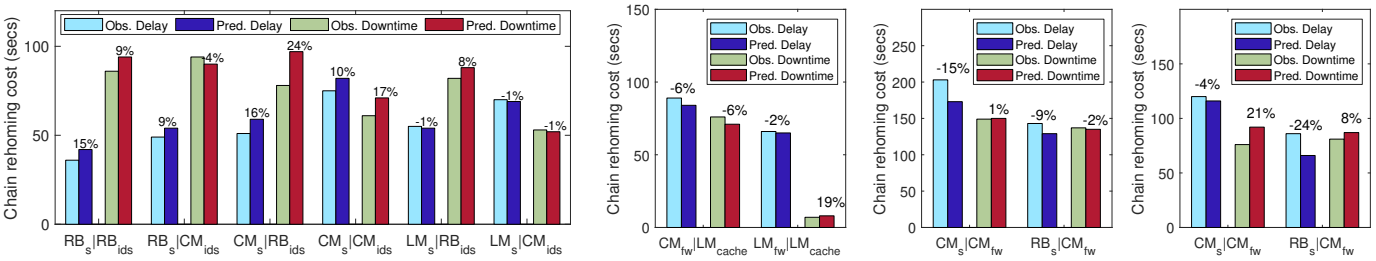
TABLE IV: Average absolute prediction error for the chain cost metrics across all feasible combinations of each chain.

MERIT accurately predicts the costs in each case, with a less than 10% prediction error in most cases. The average prediction error for chain delay and downtime across all eight combinations is 7.7% and 7.6%, respectively (see Table IV).

When optimizing for chain rehoming delay, MERIT rightly predicts the optimal (thus, Oracle) $LM_{fw}\, RB_{ffm}\, RB_{ids}\, LM_{ats}$ rehoming option; here, the subscripts, fw, ffm, ids, and ats, refer to the rehomable VNFs. If we instead employ the Homogeneous suggested $CM_{fw}\, CM_{ffm}\, CM_{ids}\, LM_{ats}$ (since ATS can only be live migrated), the chain delay cost would increase by 89%, as shown in Figure 10. When optimizing for the average (across both branches) chain downtime, MERIT predicts $LM_{fw}\, RB_{ffm}\, CM_{ids}\, LM_{ats}$ as the optimal combination, which is slightly different from the Oracle combination of $LM_{fw}\, CM_{ffm}\, CM_{ids}\, LM_{ats}$ (due to non-zero prediction errors, see Figure 11). Nonetheless, MERIT only incurs an additional 9% downtime cost relative to Oracle, whereas Homogeneous incurs an additional 41% downtime. Similar observations can be made for the other metrics in Figure 10.

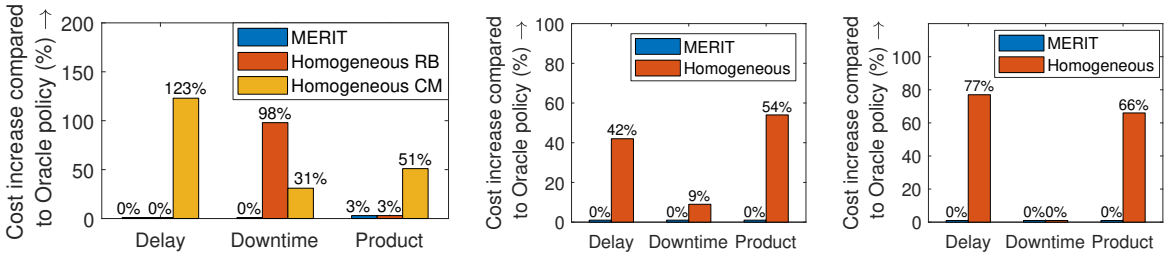
2) Rehoming the IDS (client-switch-IDS-server) chain

Figure 12(a) shows our results for the IDS chain, where the switch VNF and IDS VNF are subject to rehoming; we use shared storage for this experiment. We use a medium instance size (MEM=4GB) for both switch and IDS VNFs, with 919MB image and 700MB disk size for switch, and 1.7GB image and 510MB disk for IDS. Note that we did not train using a small instance size or a less than 500MB disk size during modeling. For the IDS chain, LM is infeasible for IDS VNF (due to high PDR) but there are no constraints for switch VNF. Consequently, due to its gray-box nature, MERIT considers $2 \times 3 = 6$ feasible action combinations, as opposed to the $3 \times 3 = 9$ combinations a black-box approach would consider. The IDS VNF uses Snort [44], which inspects every



(a) IDS chain (b) Cache chain (c) Large firewall VNF (d) Small firewall VNF

Fig. 12: Observed and model-predicted results for the chain rehoming costs for (a) the IDS chain, (b) the cache chain, (c) the firewall chain with large firewall VNF disk size, and (d) the firewall chain with small firewall VNF disk size.



(a) IDS chain (b) Large firewall VNF (c) Small firewall VNF

Fig. 13: Percentage increase in cost, relative to the Oracle policy, for (a) IDS chain, (b) firewall chain with large firewall VNF disk size, and (c) firewall chain with small firewall VNF disk size.

incoming packet and writes to the intrusion log; this results in high PDR, making LM infeasible. The switch VNF does not have any constraints. We thus have $2 \times 3 = 6$ feasible action combinations, as opposed to the $3 \times 3 = 9$ actions we would have to consider with a black-box approach.

In Figure 12(a), the x-axis ticks are represented as $action_s|action_{ids}$, where $action_s$ and $action_{ids}$ are the rehoming action applied to the switch and IDS VNF, respectively. We refer to rebuild, cold migrate, and live migrate actions as RB, CM, and LM, respectively. The numbers above the prediction bars for chain delay and chain downtime show the prediction error percentage for MERIT. For both chain rehoming delay and chain downtime, MERIT accurately predicts the rehoming cost, with a mean error of 5% and 8% respectively (Table IV). However, for the CM_{switch} CM_{snort} action, our error is very high; this is because when both VNFs are being simultaneously cold migrated, there is significant network contention, which we are under-predicting with the linear regression technique, despite our network contention model in Section IV-D. For chain downtime, MERIT has good prediction accuracy, with a mean error of 11%. For the individual VNF-specific rehoming costs, MERIT's average prediction error is 13% and 16% for rehoming delay and connectivity downtime, respectively.

When optimizing for chain rehoming delay, MERIT accurately predicts the optimal rehoming combination of RB_s RB_{ids} (rebuild switch VNF and rebuild IDS VNF), thus resulting in the same cost as Oracle and Homogeneous (Figure 13(a)). In fact, the predicted ordering of the combinations, in terms of chain delay, is exactly in agreement with the empirically observed ordering for chain delay. When optimizing for chain downtime, MERIT again correctly predicts LM_s CM_{ids} as the best option. Instead, if we use the Homogeneous policy, we would either pick RB_s RB_{ids} or CM_s CM_{ids}, which would result in an increase in chain downtime of 98% and 31%, respectively. When minimizing the product of chain delay and downtime, MERIT incorrectly picks RB_s RB_{ids} as the optimal, instead of the Oracle LM_s CM_{ids} combination. However, the misprediction error is small, thus incurring only a 3% cost increase over Oracle. Across all feasible combinations, MERIT accurately predicts the rehoming costs with a mean prediction error of 5%–8%.

3) Rehoming the caching (client-firewall-cache-server) chain

Figure 12(b) shows our results for the Web caching chain, where the firewall VNF and cache server VNF are subject to rehoming; we use shared storage for this experiment. We use a medium instance size (MEM=4GB) for the firewall VNF with a 250MB image and 1.7GB disk. For the cache server VNF, we use another medium instance with 850MB image and 760MB disk size. We use a medium instance size for both firewall and cache VNFs, with 250MB image and 1.7GB disk size for firewall, and 850MB image and 760MB disk for IDS. For the caching chain, RB is infeasible for firewall As discussed in Section VI-A, the firewall VNF relies on IPTables, which are incrementally updated, making it infeasible to rebuild the firewall VNF. For the cache server VNF, it is stateful (so rebuild is infeasible) and the Apache Traffic Server caches contents in memory, making CM infeasible as well. We thus

experiment with the remaining two combinations.

Our average prediction error for chain rehoming delay and downtime across all cases is 3.8% and 12.5%, respectively; note that the downtime prediction error for LM_{fw} LM_{cache} is high since it is challenging to predict the small connectivity downtime under live migrate, as discussed in Section IV. Since the actual and predicted values are small (7s and 8s, respectively), the modeling error is inflated.

When optimizing for chain rehoming delay, MERIT rightly predicts the optimal LM_{fw} LM_{cache} rehoming option; similarly for the chain downtime metric and the product of chain rehoming delay and downtime metric, which have the same optimal action of LM_{fw} LM_{cache}. Since rebuild is not a feasible option for either VNF of this chain, we identify live migrate as the optimal action for this shared storage scenario.

4) Rehoming the Firewall (client-switch-firewall-server) chain

We now consider the Firewall chain, where the switch VNF and firewall VNF are subject to rehoming. This time, we use non-shared storage for this experiment, so the feasible actions are RB and CM. As before, RB is infeasible for the firewall VNF, resulting in two action combinations for the chain, with cold-migrate–all being the Homogeneous combination. We use a small instance size (1 core, 2GB memory) for the switch VNF with a 250MB image and 400MB disk. For the firewall VNF, we use a medium instance with 250MB image. We experiment with two different disk sizes for the firewall VNF in this chain, 2GB (Large) and 700MB (Small); the corresponding results are shown in Figures 12(c) and 12(d), respectively. Note that we did not train using a small instance size.

When optimizing for chain rehoming delay under firewall with large disk (Figure 13(b)), MERIT rightly predicts the optimal $RB_s\,CM_{fw}$ rehoming option, achieving the same cost as Oracle. If instead, the Homogeneous suggested $CM_s\,CM_{fw}$ was employed, the chain rehoming delay would increase by about 42%. When considering the chain downtime or the product of chain delay and downtime metrics, MERIT again rightly predicts the optimal combination. By contrast, Homogeneous incurs a cost increase of 9% and 54%, respectively, over MERIT, for the chain downtime and the product of chain delay and downtime metrics.

When using the small disk for firewall VNF (Figure 13(c)), MERIT rightly predicts the heterogeneous $RB_s\,CM_{fw}$ action as the optimal when optimizing chain rehoming delay, resulting in the same cost as Oralce. If instead the Homogeneous $CM_s\,CM_{fw}$ was employed, the chain rehoming delay would increase by about 77%. When optimizing for chain downtime, all three policies, MERIT, Homogeneous, and Oracle, pick the optimal $CM_s\,CM_{fw}$ combination. Finally, when optimizing for the product of chain delay and downtime, MERIT again rightly predicts the heterogeneous $RB_s\,CM_{fw}$ action combination, achieveing the same cost as Oracle. By contrast, Homogeneous incurs a 66% increase in product cost.

VII. Related Work

Prior work on modeling rehoming costs has largely focused only on live migration. Nathan et al. [11] perform a thorough evaluation of existing models to predict VM live migration time and propose a new model that takes into account important factors such as the writable working set size (WSS) and page dirty rate (PDR). Akoush et al. [13] provide simulation models based on historical observations of page dirtying rate in Xen-based VMs to predict the total live migration and service interruption times. Wu et al. [14] develop regression models that capture the impact of CPU resource availability on the performance of live migration. MERIT's modeling of rehoming costs also considers rebuild and cold migrate, which are viable alternatives to live migrate, and also takes into account the underlying cloud storage framework. Further, MERIT leverages the models to provide optimal rehoming recommendations, that include live migration.

Wang et al. [47] provide virtual router migration as a network management primitive allowing (virtual) routers to move across physical nodes. Mistral [48] considers a model-based approach to estimate the cost of single VM migrations in the context of improving the power efficiency and resource utilization in cloud infrastructures. We take a more holistic approach, with information provided by the service, and model different techniques for a wider range of rehoming actions such as rebuild and migration with a focus on its impact on the service chain. Mistral [48] optimizes the overall *data center utility* by choosing adaptation actions such as increasing the CPU allocation, migrating VMs, and restarting hosts. Hence, Mistral may lead to sub-optimal decisions from the perspective of each service chain. By contrast, we focus on the rehoming actions for VNFs to optimize *chain-specific* metrics such as rehoming delay and connectivity downtime. Wood et al. [12] espouses a graybox approach for VM migration taking into account OS and application-level statistics. Our graybox rehoming involves simple user information such as the nature of the VMs (stateful/stateless) in the chain.

Prior work on holistic models for service chains focus on various factors that influence *initial placement* such as the hardware and resource constraints [49], [50]. MERIT specifically focuses on identifying the (possibly heterogeneous) optimal rehoming actions to be taken for service chain rehoming. Prior work on improving the performance of specific rehoming actions [15], [16], [17], [18] are orthogonal to our focus in this report, which is on identifying the optimal rehoming actions for a chain, from among available actions. Our model-driven approach can be extended to incorporate other rehoming actions, such as variants of live migrate and cold migrate.

VIII. Conclusion

This report identifies a practical problem in network provider clouds – how to optimally rehome a VNF service chain in response to hotspots, upgrades or failures. We demonstrate the importance of considering multiple, alternative rehoming actions, such as rebuild and cold migrate, in addition to the existing de-facto option of live migrate. We empirically

analyze the rehoming costs of various rehoming actions and identify the features which facilitate the modeling of rehoming costs. Finally we present the design and implementation of the MERIT system that leverages our models to rehome service chains by estimating, at run time, the impact of single-VNF rehoming on other simultaneous rehoming actions. Our experimental evaluations on OpenStack using real-world VNF chains highlight the superiority of MERIT over the existing practice of applying the same rehoming action across all VNFs, and also illustrates the importance of considering the impact of one rehoming action on another.

ACKNOWLEDGMENT

This work was supported by NSF grants 1717588 & 1750109.

REFERENCES

[1] "Unraveling AT&T's and Verizon's Virtualization Vendors," https://www.sdxcentral.com/articles/news/unraveling-att-and-verizons-virtualization-vendors/2016/08/.
[2] B. Han *et al.*, "Network function virtualization: Challenges and opportunities for innovations," *IEEE Communications Magazine*, vol. 53, no. 2, pp. 90–97, 2015.
[3] "First Responder Network," https://www.firstnet.gov.
[4] R. Nathuji *et al.*, "Q-clouds: Managing performance interference effects for QoS-aware clouds," in *EuroSys 2010*, Paris, France, pp. 237–250.
[5] H. Nguyen *et al.*, "AGILE: Elastic Distributed Resource Scaling for Infrastructure-as-a-Service," in *ICAC 2013*, San Jose, USA, pp. 69–82.
[6] G. Ananthanarayanan *et al.*, "Reining in the Outliers in Map-reduce Clusters Using Mantri," in *OSDI 2010*, Vancouver, BC, Canada, 2010, pp. 1–16.
[7] A. Verma *et al.*, "Large-scale Cluster Management at Google with Borg," in *EuroSys 2015*, Bordeaux, France, 2015.
[8] "Amazon Elastic Compute Cloud (EC2)," http://aws.amazon.com/ec2.
[9] "Google Cloud Platform," https://cloud.google.com.
[10] "AT&T DataCenter locations," https://www.business.att.com/solutions/Service/cloud/colocation/data-center-locations/.
[11] S. Nathan *et al.*, "Towards a comprehensive performance model of virtual machine live migration," in *SoCC 2015*. ACM, pp. 288–301.
[12] T. Wood *et al.*, "Black-box and gray-box strategies for virtual machine migration," in *NSDI'07*, Cambridge, MA, USA, 2007.
[13] S. Akoush *et al.*, "Predicting the performance of virtual machine migration," in *IEEE/ACM MASCOTS'10*, Miami, FL, 2010, pp. 37–46.
[14] Y. Wu and M. Zhao, "Performance modeling of virtual machine live migration," in *IEEE CLOUD*, Washington, D.C., 2011, pp. 492–499.
[15] H. Liu *et al.*, "Live migration of virtual machine based on full system trace and replay," in *ACM HPDC 2009*. ACM, 2009, pp. 101–110.
[16] E. Park *et al.*, "Fast and Space-efficient Virtual Machine Checkpointing," in *ACM VEE '11*, Newport Beach, CA, USA, 2011, pp. 75–86.
[17] H. Jin *et al.*, "Live virtual machine migration with adaptive, memory compression," in *Proceedings of the 2009 IEEE International Conference on Cluster Computing and Workshops*, New Orleans, LA, USA, 2009, pp. 1–10.
[18] P. Svärd *et al.*, "Evaluation of Delta Compression Techniques for Efficient Live Migration of Large Virtual Machines," in *ACM SIGPLAN/SIGOPS VEE 2011*, Newport Beach, CA, USA, 2011, pp. 111–120.
[19] M. Wajahat, "MERIT System Implementation with Openstack," https://github.com/PACELab/merit_system.
[20] Openstack Org. Nova Rebuild. https://docs.openstack.org/python-novaclient/latest/cli/nova.html.
[21] ——. Migrate instances. https://docs.openstack.org/nova/queens/admin/migration.html.
[22] C. Jo *et al.*, "A Machine Learning Approach to Live Migration Modeling," in *SoCC 2017*, Santa Clara, CA, USA, 2017, pp. 351–364.
[23] Openstack.org, "Configure live migrations," https://docs.openstack.org/nova/pike/admin/configuring-migrations.html.
[24] "Open source software for creating private and public clouds," https://www.openstack.org.
[25] Openstack Org, "Neutron: OpenStack project's "network connectivity as a service"," https://docs.openstack.org/neutron/latest.
[26] "Allowed address pairs," https://docs.openstack.org/dragonflow/latest/specs/allowed_address_pairs.html.
[27] "Mutilate: high-performance memcached load generator," https://github.com/leverich/mutilate.
[28] B. Hubert *et al.*, "WonderShaper: Command-line utility for limiting an adapter's bandwidth," https://github.com/magnific0/wondershaper.
[29] "Stress-ng," http://kernel.ubuntu.com/~cking/stress-ng.
[30] "High availability of live migration," http://superuser.openstack.org/wp-content/uploads/2017/06/ha-livemigrate-whitepaper.pdf, 2017.
[31] "Configure live migrations," https://docs.openstack.org/nova/pike/admin/configuring-migrations.html.
[32] D. A. Freedman, *The Regression Line*. Cambridge University Press, 2009, ch. 2, p. 26.
[33] "Support Vector Machines and Kernel Methods," https://www.cs.cmu.edu/~ggordon/SVMs/new-svms-and-kernels.pdf.
[34] R. Schaback, "A Practical Guide to Radial Basis Functions," https://num.math.uni-goettingen.de/schaback/teaching/sc.pdf.
[35] S. Hochreiter, "The vanishing gradient problem during learning recurrent neural nets and problem solutions," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 6, no. 02, pp. 107–116, 1998.
[36] A. Krizhevsky *et al.*, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.
[37] "Advantages of ReLU vs other Activation Functions," https://datascience.stackexchange.com/questions/23493/why-relu-is-better-than-the-other-activation-functions.
[38] S. Haykin, *Neural Networks: A Comprehensive Foundation*, 3rd ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2004.
[39] H. Liu *et al.*, "Performance and energy modeling for live migration of virtual machines," in *Proceedings of the 20th international symposium on High performance distributed computing*. ACM, 2011, pp. 171–182.
[40] "scikit-learn: Machine Learning in Python," https://scikit-learn.org.
[41] "Open Platform for NFV (OPNFV)," https://wiki.opnfv.org/display/functest/List+Of+VNFs.
[42] "FFmpeg," https://www.ffmpeg.org/.
[43] The Apache Software Foundation. Apache Traffic Server. http://trafficserver.apache.org.
[44] Cisco. Snort - Network Intrusion Detection and Prevention System. https://www.snort.org.
[45] D. Novakovic *et al.*, "Deepdive: Transparently identifying and managing performance interference in virtualized environments," in *ATC 2013*, San Jose, USA, 2013, pp. 219–230.
[46] J. Mars *et al.*, "Bubble-Up: Increasing Utilization in Modern Warehouse Scale Computers via Sensible Co-locations," in *MICRO 2011*, Porto Alegre, Brazil, 2011, pp. 248–259.
[47] Y. Wang *et al.*, "Virtual routers on the move: live router migration as a network-management primitive," in *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 4. ACM, 2008, pp. 231–242.
[48] G. Jung *et al.*, "Mistral: Dynamically Managing Power, Performance, and Adaptation Cost in Cloud Infrastructures," in *ICDCS 2010*, Genoa, Italy, pp. 62–73.
[49] B. Addis *et al.*, "Virtual network functions placement and routing optimization," in *IEEE CloudNet 2015*. IEEE, pp. 171–177.
[50] H. Moens and F. D. Turck, "VNF-P: A model for efficient placement of virtualized network functions," in *CNSM 2014*, Nov 2014, pp. 418–423.