# MERIT: Model-driven Rehoming for VNF Chains

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 paper, 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]. For example, first responder services (EMS, police, fire) require very 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]. Further, network provider clouds are typically resource constrained as they primarily consist of cloud sites (including edge sites) that host anywhere between 10–500 servers [5]. 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.

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 [6], [7]. 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

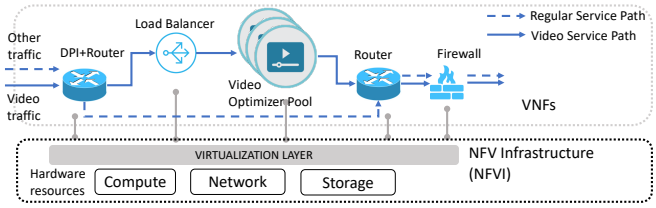



Fig. 1: An NFV ecosystem showing a video optimizing service chain, the VNFs of which are deployed on virtualized infrastructure (NFVI).

option. Based on this intuition, this paper 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 ([6], [7], [8], [9]) 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. 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 re-

homing 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 paper:

- ***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.
- ***Modeling of rehoming costs*** *(Section IV):* Using our measurements, we leverage supervised learning to model the delay and downtime for different rehoming actions.
- ***Implementation and evaluation on real-world service chains*** *(Sections V and VI):* We implement an (open-source [10]) autonomous rehoming system that leverages empirically trained models and interacts with OpenStack 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

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 that help with rehoming; we consider three such practical rehoming actions, as described below. While variants of rehoming actions are possible, our focus is on determining the optimal VNF rehoming actions 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. 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:** 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) [11]. 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 [11]. 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, the *rehoming delay under cold migrate can be high*, especially for a large disk size.
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 [6]. 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 migration can take a long time to complete*, especially for applications with a high page dirty rate [12]. In such cases, live migration can be aborted via a timeout feature [11].

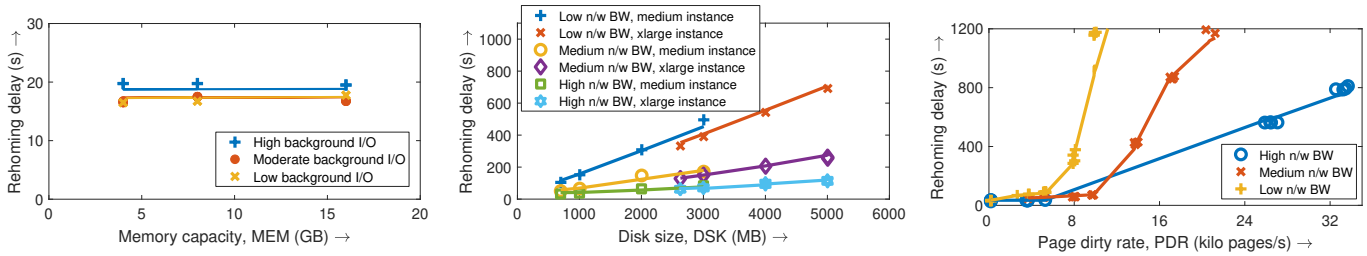
## III. Empirical analysis of rehoming

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. 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 [11], 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.

### 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. For shared storage, we employ GlusterFS (v4.1) as our shared network filesystem over ext4




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

using a single brick Gluster volume.

### *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, 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.

**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. 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 [12].

**Cloud Infrastructure parameters.** To study the impact of available bandwidth on rehoming, we employ WonderShaper [13] 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 [14] to generate different background I/O loads at the target host. We use mpstat and iostat to measure I/O statistics.

### *C. Empirical analysis of rehoming costs*

#### *1) Analysis of rehoming delay*

The markers in Figure 2 show our empirical results for the rehoming delay of all three actions under shared storage. Figure 2(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.

Figure 2(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.

Figure 2(c) shows the live migrate rehoming delay as a function of PDR for different network bandwidths (BW). To prevent live migration from getting stuck, we set a timeout of 20 minutes. 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 2(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.

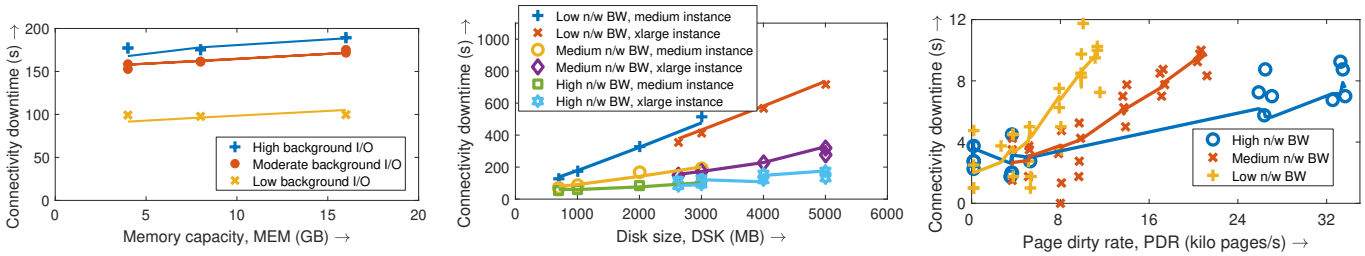
#### *2) Analysis of connectivity downtime*

The markers in Figure 3 show our empirical results for the connectivity downtime of all actions under shared storage. The results in Figure 3 are from the same experiments as Figure 2, and thus the VNF and system parameters are the same.

Figure 3(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.

Figure 3(b) shows the connectivity downtime for cold migrate. The trends are very similar to those in Figure 2(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 3(c) shows the downtime for live migrate. In stark contrast to rebuild and cold migrate, the downtime




(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. 3: Empirical results for connectivity downtime of different rehoming actions under shared storage.

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; we refer interested readers to our technical report for full results [15]. Note that live migrate is infeasible under non-shared storage.

## IV. Modeling the Rehoming Costs

This section first presents our modeling results for a single VNF, and then, in Section IV-C, we show how MERIT leverages the single VNF models to predict the optimal simultaneous rehoming actions for the entire service chain.

### A. Modeling methodology

We consider the simple client-switch-server VNF chain and focus on modeling the rehoming costs for the switch VM.

*1) 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. For NN, we consider the sigmoid and the rectified linear unit (ReLU) activation functions for the hidden layer. We experiment with multiple, diverse techniques so we can evaluate their accuracy and determine the best model for each rehoming action. While LR is easy-to-use and quick to train, it cannot model non-linearities or dependencies between features. SVR can model non-linearities, but cannot easily model dependencies between features. NN can model both non-linearities and dependencies, but is more complex and slow to train [16].

*2) Features used for model training*

Since different rehoming actions work differently, we choose features based on the specific action being modeled. However, to model a given action, we employ the same features for all learning techniques and rehoming costs (delay and downtime).

For rebuild, we use the following features: (i) image size of the original VM , (ii) instance size, denoted by its memory capacity (MEM), (iii) VM CPU usage, (iv) available bandwidth (BW), (v) mean I/O wait time, and (vi) standard deviation of I/O wait time. The intuition for including these 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 impact the rehoming cost under rebuild. For cold migrate, we additionally use the disk size 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, prior works have investigated the set of useful features for modeling [6], [12], 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, (v) modified words per page, (vi) working set entropy (WSE), and (vii) non-working set entropy (NWSE). To capture I/O contention, as before, we also use the IOPS and the read and write kBps as 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.

### 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.

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 I and II, 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.

Based on the final results in Tables I and II, *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. However, for live migrate, LR has poor accuracy, making it an impractical choice for MERIT. Instead, *for live migrate, we choose NN*

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

*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 2(a), 2(b), 3(a), and 3(b). For live migrate rehoming costs, the NN with ReLU model is shown as the solid lines in Figures 2(c) and 3(c).

### C. Modeling network contention when applying MERIT

At run-time, due to simultaneous rehoming of VNFs in the chain, the available network bandwidth (BW) is shared among them. Thus, *BW for each VNF rehoming action must be estimated at run-time* when applying the rehoming models.

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. 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 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.

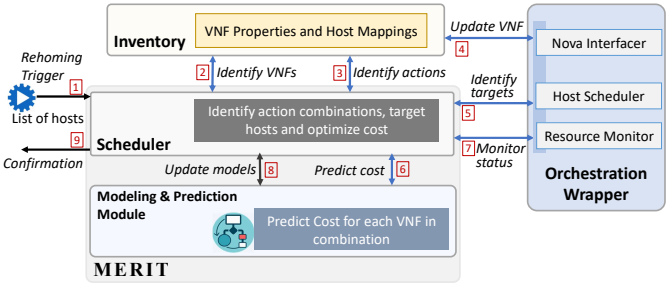



Fig. 4: Illustration of our MERIT system implementation.

## V. System Design and Implementation

Figure 4 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 [10].

The NFV Infrastructure (NFVI) monitoring systems (e.g, Ceilometer in OpenStack [11]) 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). 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 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 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 (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).

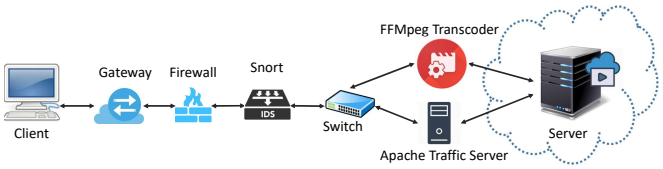



Fig. 5: Gi-LAN service chain used in our evaluation. VNFs are based on real-world reference implementations from OPNFV [17].

## VI. Evaluation Results

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 [17].

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 [18]), cache VM (Apache Traffic Server [ATS] [19]), and a server VM. This chain, illustrated in Figure 5, 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 [20]), and server VM, and is representative of a network intrusion detection system.

We also evaluate MERIT for a Firewall chain and a Web caching chain; due to lack of space, we defer the discussion and results for these chains to our technical report [15].

### A. Evaluation methodology

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.

We refer to rebuild, cold migrate, and live migrate actions as $RB$, $CM$, and $LM$, respectively. 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 where the *entire chain* needs to be rehomed in response to maintenance or failures. 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).

For each chain, we compare MERIT with the following:

- The **Oracle** policy applies the optimal rehoming actions at each VNF. We "implement" the unrealistic Oracle by experimenting with all feasible combinations and then labeling the minimum-cost optimal combination as the Oracle policy.
- 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.

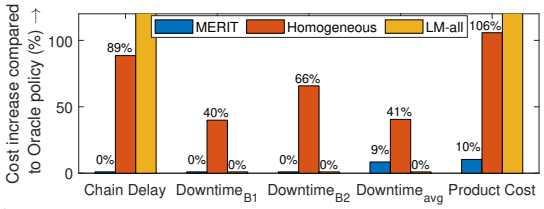



Fig. 6: Percentage increase in cost, relative to the Oracle policy, for MERIT and the best homogeneous policy for the Gi-LAN chain. Also shown, for completeness, is the infeasible live-migrate all policy.

### B. Rehoming evaluation results

#### 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 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 6 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.

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 6. 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. Across all experiments, the average prediction error for chain delay and downtime is 7.7% and 7.6%, respectively; a detailed analysis of prediction accuracy and MERIT-predicted rehoming actions can be found in our technical report [15].

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

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.

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 7). 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%.

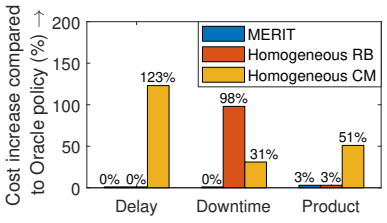



Fig. 7: Percentage increase in cost, relative to Oracle policy, for IDS chain.

## VII. Related Work

Prior work on modeling rehoming costs has largely focused only on live migration. Nathan et al. [6] 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. [8] 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. [9] 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.

Mistral [21] 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. [7] 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.

## VIII. Conclusion

This paper 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.

## 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] H. Nguyen *et al.*, "AGILE: Elastic Distributed Resource Scaling for Infrastructure-as-a-Service," in *ICAC 2013*, San Jose, USA, pp. 69–82.
[5] "AT&T DataCenter locations," https://www.business.att.com/solutions/Service/cloud/colocation/data-center-locations/.
[6] S. Nathan *et al.*, "Towards a comprehensive performance model of virtual machine live migration," in *SoCC 2015*. ACM, pp. 288–301.
[7] T. Wood *et al.*, "Black-box and gray-box strategies for virtual machine migration," in *NSDI'07*, Cambridge, MA, USA, 2007.
[8] S. Akoush *et al.*, "Predicting the performance of virtual machine migration," in *IEEE/ACM MASCOTS'10*, Miami, FL, 2010, pp. 37–46.
[9] Y. Wu and M. Zhao, "Performance modeling of virtual machine live migration," in *IEEE CLOUD*, Washington, D.C., 2011, pp. 492–499.
[10] M. Wajahat, "MERIT System Implementation with Openstack," https://github.com/PACELab/merit_system.
[11] "Open source software for creating private and public clouds," https://www.openstack.org.
[12] C. Jo *et al.*, "A Machine Learning Approach to Live Migration Modeling," in *SoCC 2017*, Santa Clara, CA, USA, 2017, pp. 351–364.
[13] B. Hubert *et al.*, "WonderShaper: Command-line utility for limiting an adapter's bandwidth," https://github.com/magnific0/wondershaper.
[14] "Stress-ng," http://kernel.ubuntu.com/~cking/stress-ng.
[15] M. Wajahat *et al.*, "Model-driven rehoming for vnf chains," https://tr.cs.stonybrook.edu/tr/sbcs-tr-2020-13, Tech. Rep., 2020.
[16] P. Cunningham *et al.*, "Stability problems with artificial neural networks and the ensemble solution," *Artificial Intelligence in Medicine*, vol. 20, no. 3, pp. 217–225, 2000.
[17] "Open Platform for NFV (OPNFV)," https://wiki.opnfv.org/display/functest/List+Of+VNFs.
[18] "FFmpeg," https://www.ffmpeg.org/.
[19] The Apache Software Foundation. Apache Traffic Server. http://trafficserver.apache.org.
[20] Cisco. Snort - Network Intrusion Detection and Prevention System. https://www.snort.org.
[21] G. Jung *et al.*, "Mistral: Dynamically Managing Power, Performance, and Adaptation Cost in Cloud Infrastructures," in *ICDCS 2010*, Genoa, Italy, pp. 62–73.