ENSC: Multi-Resource Hybrid Scaling for Elastic Network Service Chain in Clouds

Hui Yu*, Jiahai Yang*, Carol Fung†, Raouf Boutaba‡, Yi Zhuang*

*Institute for Network Sciences and Cyberspace, Tsinghua University
†Department of Computer Science, Virginia Commonwealth University
‡David R. Cheriton School of Computer Science, University of Waterloo
yuhui7red@outlook.com, yang@cernet.edu.cn, cfung@vcu.edu
rboutaba@uwaterloo.ca, zhuangy17@mails.tsinghua.edu.cn

***Abstract*—Software-based network service chains in Network Function Virtualization (NFV) need to be dynamically allocated and scaled on hardware resources. This is because the resource demand of virtual network functions (VNFs) typically varies as a results of network flow volume. NFV elastic solutions by coarse-grained horizontal scaling or fine-grained vertical scaling have been investigated in recent years. However, none of the existing solutions can achieve both efficiency and scalability. To address this challenge, we propose elastic network service chain (ENSC), which utilizes a fine-grained hybrid scaling method to achieve both NFV efficiency and scalability. We systematically compare horizontal scaling with vertical scaling from six aspects and determine the priority within hybrid scaling. We formulate the resource allocation problem in the cloud datacenter as an integer linear programming (ILP) model and develop a heuristic algorithm called *Rubik*. Our evaluation results show that ENSC achieves higher acceptance ratios and resource utilization than horizontal scaling and vertical scaling methods.**

***Index Terms*—Middlebox; Network Function Virtualization; Service Chain; Resource Scaling; Traffic Steering.**

I. Introduction

Network Function Virtualization (NFV) [1] is an emerging technology that conquers the limitation of traditional proprietary middleboxes [2] by decoupling Network Functions (NFs) from dedicated hardware to standard commodity servers, in order to reduce cost and improve flexibility. The paradigm has been embraced by both academia and industry rapidly by many leading companies and organizations [3]. A primary challenge of NFV is to dynamically allocate and scale hardware resources for VNFs. The resource demand of virtual network functions (VNFs) typically varies as a results of network flow volume. If the resource cap is too low, the VNF will experience degraded performance. On the other side, if the resource cap is too high, resource will be wasted. Therefore, an elastic resource scaling mechanism is called upon to determine the resource allocation dynamically based on the need at the time.

Several NFV studies [4], [5], [6], [7] have investigated NFV elastic solutions by creating and destroying (scaling out/in) VM replicas based on demand, while balancing the workload among VMs, called *horizontal scaling*. Split/Merge [4] allow the control over VNF state so that the VNFs can split or merge for elastic execution. At the same time, the system

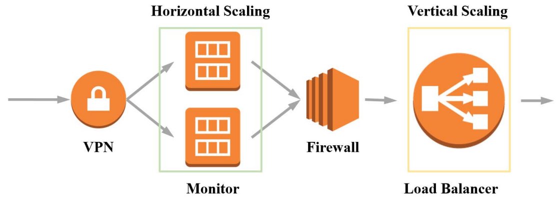


Fig. 1: A elastic network service chain with hybrid scaling.

partitions the network to ensure packets are routed to the appropriate replica. E2 [5] presents a scalable and application-agnostic scheduling framework for NFV packet processing. However, those solutions are coarse-grained which may cause resource over-provisioning and low resource utilization, because different VNFs can bottleneck on different resource [6]. Furthermore, the average VM startup time is costly [7] when scaling out new VNF instances, which may further deteriorate VNF performance. Therefore, we need a fine-grained scaling method to achieve NFV efficiency.

Compared to horizontal scaling method, *vertical scaling* method provides live resizing capability (scaling up/down) on VMs. For instance, ElasticNFV [8] provides a fine-grained cloud resource provisioning for VNFs. It can scale up/down three types of resources by adjusting their resource caps and uses a TPMM algorithm to solve scaling conflicts. Although the vertical scaling method is superior in terms of performance, it faces two problems: compatibility and scalability. A study from Cao et al [9] shows that some VNFs cannot improve their performance through resource vertical scaling, especially vCPU and memory scaling. For example, scaling up Snort [10] by allocating more vCPU resources is not an effective option, because Snort is a single-threaded application. On the other hand, due to the physical machine (PM) capacity, the vertical scaling method is limited to the capacity of a single node. Therefore, the vertical scaling method aims at all small and medium business and the horizontal scaling supports small, medium and big business. In this sense, coarse-grained horizontal scaling is more scalable than the vertical scaling.

In this paper, we present ENSC, which exploits a fine-grained *hybrid scaling* method to achieve NFV scaling efficiency and scalability. The goal of ENSC is to realize minimum performance degradation and minimum resource cost of cloud datacenter for every service chain. To design an

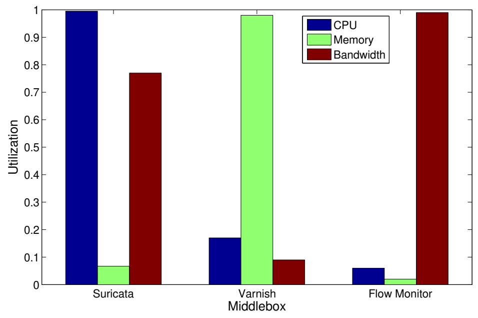


Fig. 2: Normalized resource usage of three middlebox.

elastic network service chain with hybrid scaling capability to meet the above goal, two challenges shall be addressed. The first is priority analysis for hybrid scaling. Hybrid scaling contains vertical scaling and horizontal scaling as shown in Fig. 1. We need a priority rule to determine when to scale vertically and when to scale horizontally for VNFs to gain optimal performance for every service chain. Second, when the new flows come or existing flows change, we also need a valid solution to determine how to schedule flows to optimize resource allocation in cloud datacenter.

To address the above challenges, we systematically analyzes the advantages and disadvantages of horizontal scaling and vertical scaling from six aspects and conclude that vertical scaling has higher priority than horizontal scaling. When a PM cannot satisfy a scaling up request from VNFs, we can also use flow migration to fix scaling up conflict and avoid horizontal scaling. We formulate ENSC deployment in cloud datacenter as an ILP model and propose a *Rubik algorithm* to balance the trade-off in between speed and accuracy.

The contributions of this work can be summarized as follows:

- We provide a priority analysis within hybrid scaling and introduce a elastic multi-resource provisioning mechanism of vertical scaling.
- We mathematically formulate the ENSC problem into an ILP problem and propose a Rubik algorithm to find a solution to optimize resource allocation in cloud datacenter.
- We demonstrate that ENSC outperforms the existing horizontal and vertical scaling solutions through experiments.

The rest of the paper is organized as follows. Section II analyzes the priority within hybrid scaling. Section III puts forward an ILP model to formulate the resource allocation problem in datacenter and a heuristic solution. Section IV demonstrates experimental results. Finally we conclude the work in Section V.

II. Priority Analysis of Hybrid Scaling

In this section, we analyze priority within hybrid scaling to achieve better performance for each service chain. The priority rule determines when to scale up/down existing VNF instances and when to scale out/in VNF instances for frequent workload changes.

Resource utilization: Existing studies have found that different VNFs can bottleneck on CPU, memory or link bandwidth [6], [11], [12]. To verify these findings, we measured three types of resource footprints of several canonical VNFs. Fig. 2 shows the results of three VNFs. Each VNF's maximum resource consumption was normalized to 1. We can see that the resource consumption varies across three types: the Suricata intrusion detection system [13] consumes a vast amount of CPU resource, the Varnish HTTP cache [14] bottlenecks on memory, and the flow monitoring implemented on Click [15] is link bandwidth-bound.

Thus, an efficient resource utilization is to provide fine-grained cloud resource provisioning for VNFs with the vertical scaling method than to enable elasticity by creating and destroying VM replicas with horizontal scaling.

Scaling period: Horizontal scaling method uses VM as the scaling unit, which is limited by the VM startup time in the cloud. Mao *et al.* [7] studied the startup time of cloud VMs across three real-world cloud providers: Amazon EC2, Microsoft Azure and Rackspace. The average VM startup time of EC2 is 96.6 seconds, Azure is 356.6 seconds and Rackspace is 44.2 seconds. As a vertical scaling method, KVM hypervisor can dynamically adjust the number of vCPUs and memory size of a VM at runtime. Performance measurements by Wind River [16] show that it is possible to hot plug a CPU in about 40 ms and unplug a CPU in about 20 ms. ElasticSwitch [17] shows that bandwidth allocation between VMs can be achieved in milliseconds. Compared to the VM startup time, the vertical scaling time is negligible.

We conclude that vertical scaling methods are more time efficient than horizontal scaling methods.

Response time: The horizontal scaling can be modeled as multiple single-server queues (*N M/M/1*) and the vertical scaling can be modeled as single queue with multiple servers (*1 M/M/N*). Let N denote the number of queues for horizontal elasticity or the number of servers for vertical elasticity. Let λ denote the mean arrival rate, and μ denote the departure rate. In normal conditions, λ is less than μ, otherwise there will be a large backlog in the queue. For horizontal scalability architecture, the average response time of the multiple single-server queues is:

$$E[R_{ms}] = \frac{N}{\mu - \lambda} \tag{1}$$

The single queue with multiple servers for elastic VM is:

$$E[R_{sm}] = \frac{N \cdot \mu}{\mu^2 - \lambda^2} \tag{2}$$

It is easily proven to be:

$$E[R_{ms}] - E[R_{sm}] = \frac{N \cdot \lambda}{\mu^2 - \lambda^2} > 0 \tag{3}$$

It can be strictly proved that the vertical scaling method has faster response time than horizontal scaling method for incoming tasks from the former VNFs.

Compatibility: Many researches are directed to parallelizing VNFs with horizontal scaling [4], [5]. Elastic NFV execution with horizontal scaling is relatively mature. On the other hand, one of the challenging issue in vertical scaling is that some VNFs cannot improve their performance through dynamic resources scaling, especially vCPU and memory scaling. As we mentioned, Cao *et al* [9] confirms the fact that Snort scales poorly on multi-core systems. A potential solution to this problem is to run multiple instances of Snort and configure them to handle partial traffic in the same VM (horizontal scaling).

As for the compatibility, horizontal scaling has more advantages than vertical scaling.

Scalability: Vertical scaling is limited to PM capacity, which limits vertical scaling to small to medium businesses. However, although horizontal scaling induces some performance overhead, such as data synchronization and load balance, it can supports all small, medium and big business.

Therefore, horizontal scaling has better compatibility than vertical scaling.

Robustness: The static load-balancer instance which schedules flows among VNF instances can be a single-point-of-failure in horizontal scaling. Compared to static VNF instances, vertical scaling can avoid overloading and thus makes it more resistant to failure. However, elastic VNF instances themselves could be a single-point-of-failure.

Thus, these two scaling methods have similar performance in robustness.

Through the above priority analysis from six aspects, we conclude that vertical scaling has better performance than horizontal scaling for elastic service chain, but horizontal scaling can be applied to more scenarios. Therefore, vertial scaling has higher priority than horizontal scaling.

As mentioned above, the average VNF instance startup time varies from a few seconds to a few minutes. At the same time, the flow migration can be completed in milliseconds. For example, OpenNF [18] can move, copy and share internal NF state alongside updates to network forwarding state. Therefore, we leverage flow migration to fix scaling up conflict and avoid scaling out new VNF instances, which can reduce the system overhead induced by hybrid scaling.

In conclusion, vertical scaling has the highest priority within hybrid scaling, then we can use flow migration to fix scaling up conflict and avoid horizontal scaling, and horizontal scaling has the lowest priority.

III. Resource Allocation in Cloud Datacenter

In this section, we introduce our solution to schedule flows to VNF instances and allocate service chains into physical datacenters. Given that current datacenters are mostly organized in oversubscribed tree-like topology [19], we consider a three-layer single root tree-shaped datacenter for simplicity, which is easy to expanded to multi-root topology. We start with the formal definitions, followed by a mathematical modeling and a heuristic solution.

A. Definitions

1) Physical Resource: $R = \{CPU, memory, \ldots\}$ represents different resource types of a physical machine.

2) Cloud Datacenter: $D = (\tilde{N}, \tilde{L})$ represents a cloud datacenter, where $\tilde{N}$ and $\tilde{L}$ are a set of physical machines and physical links, respectively. $c^r_{\tilde{n}}$ represents capacity of physical machine $\tilde{n} \in \tilde{N}$ for resource type $r \in R$. $b_{\tilde{l}}$ represents bandwidth capacity of physical link $\tilde{l} \in \tilde{L}$.

3) Service Chain: $C^i = (\bar{N}, \bar{L})$ represents a service chain, where $\bar{N}$ and $\bar{L}$ are a set of virtual links, respectively. I represents set of service chains, $i \in I$. $c^r_{\bar{n}}$ represents capacity of physical machine $\bar{n} \in \bar{N}$ for resource type $r \in R$. $b_{\bar{l}}$ represents bandwidth capacity of physical link $\bar{l} \in \bar{L}$.

4) Flow: $F^{ik} = (N, L)$ represents a flow, where N represents a set of VNF instances the flow goes through and L represents virtual links the flow goes through. K represents a set of flows, $k \in K$. c^{ikr}_{n} represents resource usage of flow in VNF instance $\bar{n} \in \bar{N}^i$ for resource type $r \in R$. b^{ik}_{l} represents bandwidth usage of flow in virtual link $l \in L^{ik}$.

TABLE I: Notations

Notations	**Definitions**
r	A resource
u	Buffer of VNF instance
e^{ik}_{F}	Flow size of flow F^{ik}
t^{ik}_{F}	Migration time of flow F^{ik}
$m^{ik}_{Fn} \in \{0,1\}$	A boolean variable that indicates whether flow F^{ik} is migrated to VNF instance $n \in N^{ik}$
$o^{i}_{\bar{n}} \in \{0,1\}$	A boolean variable that indicates whether $\bar{n}^i$ is a new scaling out instance
$x^{i}_{\bar{n}\tilde{n}} \in \{0,1\}$	A boolean variable that indicates whether VNF instance $\bar{n}^i$ is embedded in physical machine $\tilde{n}$
$y^{i}_{\bar{l}\tilde{l}} \in \{0,1\}$	A boolean variable that indicates whether virtual link $\bar{l}^i$ is embedded in physical link $\tilde{l}$
$y^{i}_{\bar{l}\tilde{n}} \in \{0,1\}$	A boolean variable that indicates whether virtual link $\bar{l}^i$ is embedded in physical machine $\tilde{n}$
$z^{ik}_{n\bar{n}} \in \{0,1\}$	A boolean variable indicates whether VNF instance n^{ik} is VNF instance $\bar{n}^i$
$w^{ik}_{l\bar{l}} \in \{0,1\}$	A boolean variable indicates whether virtual links l^{ik} is virtual link $\bar{l}^i$
$\tilde{s}_{\tilde{n}\tilde{l}} \in \{0,1\}$	A boolean variable indicates whether $\tilde{n}$ is the source of $\tilde{l}$
$\tilde{d}_{\tilde{n}\tilde{l}} \in \{0,1\}$	A boolean variable indicates whether $\tilde{n}$ is the destination of $\tilde{l}$

B. Mathematical Model

The challenge is how to meet resource demand of every service chain leveraging hybrid scaling with minimum resource cost of datacenter. We propose a corresponding model to formulate the hybrid scaling of service chains in datacenter. Table I lists important notations used in this paper.

1) Physical Machine Capacity Constraint: Equation 4 ensures no violation of the capacities of physical machine for resource r.

$$\sum_{i \in I} \sum_{\bar{n} \in \bar{N}^i} x^i_{\bar{n}\tilde{n}} \cdot c^{ir}_{\bar{n}} \le c^r_{\tilde{n}} \quad \forall \tilde{n} \in \tilde{N}, r \in R \tag{4}$$

2) Physical Link Capacity Constraint: Equation 5 ensures no violation of the capacities of physical link for bandwidth.

$$\sum_{i \in I} \sum_{\bar{l} \in \bar{L}^i} y^i_{\bar{l}\tilde{l}} \cdot b^i_{\bar{l}} \le b_{\tilde{l}} \quad \forall \tilde{l} \in \tilde{L} \tag{5}$$

3) VNF Instance Capacity Constraint: Equation 6 ensures no violation of the capacities of VNF instance for resource r.

$$\sum_{k\in K}\sum_{n\in N} z_{n\bar{n}}^{ik}\cdot c_{n}^{ikr}\le c_{\bar{n}}^{ir}\quad \forall i\in I,\bar{n}\in\bar{N}^i, r\in R \tag{6}$$

4) VNF Link Capacity Constraint: Equation 7 ensures no violation of the capacities of VNF link for bandwidth.

$$\sum_{k\in K}\sum_{l\in L} w_{l\bar{l}}^{ik}\cdot b_{l}^{ik}\le b_{\bar{l}}^{i}\quad \forall i\in I,\bar{l}\in\bar{L}^i \tag{7}$$

5) VNF Instance Location Constraint: Equation 8 ensures the embedding of every VNF instance $\bar{n}\in\bar{N}^i$.

$$\sum_{\tilde{n}\in\tilde{N}} x_{\bar{n}\tilde{n}}^{i}=1\quad \forall i\in I,\bar{n}\in\bar{N}^i \tag{8}$$

6) VNF Link Location Constraint: Equation 9 ensures the embedding of every VNF link $\bar{l}\in\bar{L}^i$.

$$\sum_{\tilde{l}\in\tilde{L}} y_{\bar{l}\tilde{l}}^{i}+\sum_{\tilde{n}\in\tilde{N}} y_{\bar{l}\tilde{n}}^{i}=\{1,2,4,6,\ldots\}\quad \forall i\in I,\bar{l}\in\bar{L}^i \tag{9}$$

7) Flow Node Location Constraint: Equation 10 ensures the embedding of every flow node $n\in N^{ik}$.

$$\sum_{\bar{n}\in\bar{N}} z_{n\bar{n}}^{ik}=1\quad \forall i\in I,k\in K,n\in N^{ik} \tag{10}$$

8) Flow Link Location Constraint: Equation 11 ensures the embedding of every flow link $l\in L^{ik}$.

$$\sum_{\bar{l}\in\bar{L}} w_{l\bar{l}}^{ik}=1\quad \forall i\in I,k\in K,l\in L^{ik} \tag{11}$$

9) Flow Constraint between Every Source and Destination Node Pair: In our model, we also require virtual link embedding to satisfy the flow constraint between every source and destination node pair in each service chain topology. Equation 13 essentially states that the total outgoing flow of a physical node $\tilde{n}$ is equal to the total incoming flow unless $\tilde{n}$ hosts either a source or a destination virtual node.

$$\sum_{\tilde{l}\in\tilde{L}}\tilde{s}_{\tilde{n}\tilde{l}}\cdot y_{\bar{l}\tilde{l}}^{i}-\sum_{\tilde{l}\in\tilde{L}} d_{\tilde{n}\tilde{l}}\cdot y_{\bar{l}\tilde{l}}^{i}=\sum_{\bar{n}\in\bar{N}} s_{\bar{n}\bar{l}}^{i}\cdot x_{\bar{n}\tilde{n}}^{i}-\sum_{\bar{n}\in\bar{N}} d_{\bar{n}\bar{l}}^{i}\cdot x_{\bar{n}\tilde{n}}^{i}$$
$$\forall i\in I,\bar{l}\in\bar{L},\bar{n}\in\bar{N} \tag{12}$$

10) VNF Instance Buffer Constraint: A safe flow migration requires buffering in-flight traffic and avoids buffer overflow. Equation 13 ensures no buffer overflow of the VNF instance when migrating the flow.

$$m_{Fn}^{ik}\cdot e_{F}^{ik}\cdot t_{F}^{ik}\le u\quad \forall i\in I,k\in K,n\in N^{ik} \tag{13}$$

11) Number of Flow Migrations: Equation 14 is the number of flow migrations.

$$T(M)=\sum_{i\in I}\sum_{k\in K}\sum_{n\in N^{ik}} m_{Fn}^{ik} \tag{14}$$

12) Scaling Out VNF Instances: Equation 15 is the number of scaling out VNF instances. Coefficient $\alpha\in\mathbb{R}^+$ identifies the relative importance of the cost of scaling out VNF instances.

$$S(O)=\alpha\cdot\sum_{i\in I}\sum_{\bar{n}\in\bar{N}^i} o_{\bar{n}}^{i} \tag{15}$$

13) Host Usage: Equation 16 is the average host usage. For each host, we take the lager usage between CPU and memory as the host usage.

$$H(X)=\frac{1}{|\tilde{N}|}\cdot\sum_{\tilde{n}\in\tilde{N}}(max_{r\in R}\sum_{i\in I}\sum_{\bar{n}\in\bar{N}^i}\frac{x_{\bar{n}\tilde{n}}^{i}\cdot c_{\bar{n}}^{ir}}{c_{\tilde{n}}^{r}}) \tag{16}$$

14) Bandwidth Usage: Equation 17 is the bandwidth usage. Coefficient $\beta\in\mathbb{R}^+$ identifies the relative importance of the usage of bandwidth resources.

$$B(Y)=\beta\cdot\frac{\sum_{\tilde{l}\in\tilde{L}}\sum i\in I\sum_{\bar{l}\in\bar{L}^i} y_{\bar{l}\tilde{l}}^{i}\cdot b_{\bar{l}}^{i}}{\sum_{\tilde{l}\in\tilde{L}} b_{\tilde{l}}} \tag{17}$$

15) Objective Function: Equation 18 is primary objective function, which minimizes the performance overhead of hybrid scaling for each service chain.

$$min(T(M)+S(O)) \tag{18}$$

Equation 19 is secondary objective function, which minimizes the aggregate cost of allocating host and bandwidth resources to improve the utilization of datacenter.

$$min(H(X)+B(Y)) \tag{19}$$

Equation 18 has higher priority than 19. Both two objective functions subject to equations 4 - 13.

C. Rubik: Heuristic Solution

Rubik algorithm is a heuristic solution for the objective function of mathematical model. As priority analysis in Section II, here, we design the Rubik algorithm to realize the following priorities: (1) the top priority is to satisfy a request; (2) the second priority is to minimize the number of scaling out VNF instances; (3) the third priority is to minimize number of flow migrations; (4) the last priority is to minimize the aggregate cost of allocating host and bandwidth resources as equation 19. We use a multi-rooted tree (or fat-tree) which is the predominant topology in today's data centers.

Algorithm 1 Rubik Algorithm

```
Require: Topology tree $T$
Input: Request $r :< t, f, c >$
 1: if $t == DECREASE$ then // an existing flow decreases
 2:     for each VNF $v_i$ in $c$ do
 3:         ScaleDown($f$, $v_i$)
 4:         ScaleIn($f$, $v_i$)
 5: else
 6:     if $t == NEW$ then // a new flow comes
 7:         Route($f$, $c$)
 8:     for each VNF $v_i$ in $c$ do
 9:         if ScaleUp($f$, $v_i$) == $success$ then
10:             continue
11:         if Migrate($f$, $v_i$) == $success$ then
12:             continue
13:         if ScaleOut($f$, $v_i$) == $success$ then
14:             continue
15:         return false
16: return true
```

Let t denote request type, which contains two situation: an existing flow change and a new flow arrival; f denotes a

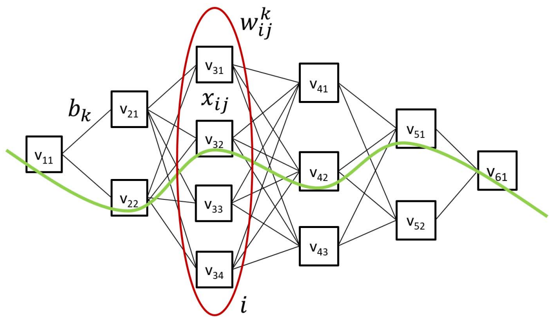


Fig. 3: Flow routing.

flow; c denotes a service chain; v_i denotes the i_{th} VNF of a service chain; v_{ik} denotes the k_{th} VNF instance of VNF i and p denotes the set of physical machines. Rubik algorithm is shown in Alg. 1. If the request decreases an existing flow, we iteratively scale down each corresponding VNF instance v_{ik} and decide whether to scale in each VNF v_i (lines 1-4) or not. If the request is from a new flow, then we route the flow among different VNF instances in the same VNF with a flow routing algorithm (lines 6-7). For each scaling up request (an existing flow increases or a new flow with path), we iteratively scale up each corresponding VNF instance v_{ik} (lines 9-10). When a PM cannot satisfy a scaling up request, we use a flow migration algorithm to move some flows out of the overloaded machine in order to accommodate the new request (lines 11-12). Finally, if the flow migration algorithm still cannot solve the problem, we create a new VNF instance by triggering the scaling out process (lines 13-14).

In the rest of this subsection, we provide a detailed overview of Rubik algorithm. We present the flow routing between two VNFs, the flow migration among different VNF instances in the same VNF, and how to scale out/in VNF instances.

1) Flow Routing: Procedure *Route(.)* is used to compute a valid path for a new flow that minimizes the combined cost of host allocation and bandwidth resources as shown in equation 19. *Route(.)* computes the solution by solving the multi-choice multidimensional knapsack problem (MMKP) [20] as shown in Fig. 3. MMKP is a more complex variant of the classical 0-1 knapsack problem (KP), an NP-hard problem. Given a set of knapsacks with limited resources and some disjoint groups of items, where each item has a profit value and requires a certain amount of resources, MMKP aims to fill the knapsacks by picking exactly one item from each group, such that total profit value of the collected items is maximized and no resource constraint of the knapsack is violated.

Suppose there are l knapsacks and n groups of items, where each group (denoted by i) has r_i items. Let $L = \{1, 2, \cdots, l\}$ be the knapsack set, $N = \{1, 2, \cdots, n\}$ be the group set, $R_i = \{1, 2, \cdots, r_i\}$ be the set of items in group i, v_{ij} be the profit value of item j in group i, w_{ij}^k be the amount of resource k required by item j in group i, and b_k be the resource amount available on knapsack k. Formally, the objective function of MMKP can be written as,

$$max \sum_{i\in N} \sum_{j\in R_i} v_{ij} \cdot x_{ij}, \tag{20}$$

which subjects to

$$\sum_{i\in N} \sum_{j\in R_i} w_{ij}^k \cdot x_{ij} \le b_k, \quad k \in L, \tag{21}$$

$$\sum_{j\in R_i} x_{ij} = 1, \quad i \in N, \tag{22}$$

$$x_{ij} \in \{0, 1\}, \quad i \in N, \quad j \in R_i. \tag{23}$$

In the route algorithm, each VNF has several VNF instances, which represent n groups of items and each group has r_i items. For each VNF in service chain, the algorithm should choose one VNF instance to process the flow. x_{ij} = 1 denotes VNF instance v_{ij} has been chosen for the flow. b_k contains all the CPU and memory constraint of PM and bandwidth constraint of PL. w_{ij}^k be the amount of resource k required by VNF instance v_{ij} when processing the flow. The objective function of routing algorithm is equation 19.

Algorithm 2 Procedure *Route*(.)

```
1: for each VNF $v_i$ in $c$ do
2:     for each VNF instance $v_{ik}$ in $v_i$ do
3:         choose $v_{ik}$ for $f$ that meets $min(H(x) + \beta \cdot B(Y))$
4: while true do
5:     for each VNF $v_i$ in $c$ do
6:         for each VNF instance $v_{ik}$ in $v_i$ do
7:             adjust $v_{ik}$ for $f$ that meets $min(H(x) + \beta \cdot B(Y))$
8:     if no VNF instance adjustment then
9:         break
```

The routing algorithms developed for solving MMKP can be divided into two classes, namely, exact algorithms and heuristics. Exact algorithms, mainly based on Branch-and-Bound [21], strive to produce the optimal solution for MMKP. However, all these exact algorithms can only be used to solve small-sized problems due to the nature of NP-hard problems. Therefore, many research efforts [22], [23] have been devoted to the development of heuristics that can find good or near optimal solutions within an acceptable computation time. The routing algorithm is also a heuristic solution, which constructs an initial feasible solution by a fast greedy procedure and then improve the quality of the initial solution by swapping items from each class. Route algorithm is shown in Alg. 2 and works as follows. For each VNF v_i in service chain, we choose the VNF instance v_{ik} for flow f that meets the requirement specified in equation 19, which constructs an initial feasible path (lines 1-3). We iteratively swap VNF instances from each VNF until there is no significant change in equation 19 (lines 4-9).

2) Flow Migration: When a scale-up conflict occurs, we try to use a flow migration algorithm to migrate some flows to avoid scaling out a new VNF instance as the priority analysis of hybrid scaling. Procedures *Migrate(.)* and *Place(.)* determine which flows should be migrated and where the flows should be migrated to. At the same time, the algorithm minimizes the number of migrated flows, and then also minimize the aggregated cost of host allocation and bandwidth resources as equation 19.

Algorithm 3 Procedure *Migrate*(.)

```
1: find flow set $F$ allocated to $v_{ik}$ which process $f$
2: sort $F$ in ascending order by average resource demand
3: for $m$ = 1 to size($F$) do
4:     for $j$ = 1 to size($F$)-$m$+1 do
5:         if $v_{ik}$ is enough after migrating $F_j, \cdots, F_{j+m-1}$ then
6:             if Place($F_j, F_{j+1}, \cdots, F_{j+m-1}$, $v_i$) == success then
7:                 return true
8: return false
```

Algorithm 4 Procedure *Place*(.)

```
1: for $j$ = 1 to SHUFFLE_TIMES do
2:     shuffle($F$)
3:     for each flow $f$ in $F$ do
4:         choose $v_{ik}$ for $f$ that meets $min(H(x) + \beta \cdot B(Y))$
5:     if all $f$ in $F$ have been placed then
6:         return true
7: return false
```

This flow migration problem is also an NP-hard problem. Correspondingly, the flow migration algorithm is also a heuristic solution. Procedure *Migrate(.)* is shown in Alg. 3. Firstly, the algorithm finds a flow set F allocated to VNF instance v_{ik} which processes flow f and then sorts flow set F in ascending order by average resource demand (lines 1-2). Then it iteratively add the number of migrated flows. If the PM of VNF instance v_{ik} has sufficient resources after migrating the adjacent flows $F_j, F_{j+1}, \cdots, F_{j+m-1}$, it then migrates these flows using the procedure *Place(.)* (lines 3-8). Procedure *Place(.)* is shown in Alg. 4. The procedure shuffles the migrated flow (line 2). For each flow f in F, it chooses a VNF instance v_{ik} for flow f that meets the requirement in equation 19 (lines 3-4). If all f in F have been placed, the flow migration is successful (5-6). Otherwise, the process is repeated.

3) Scaling Out New VNF Instances: If the flow migration can not solve the scaling conflict, we use procedure *ScaleOut(.)* to create a new VNF instance to accommodate the flows. Procedure *ScaleOut(.)* determines where a new VNF instance should be created and which flows should be migrated to the new VNF instance; It minimizes the number of new VNF instances, and then the number of migrated flows, in order to meet the requirement specified in equation 19.

Algorithm 5 Procedure *ScaleOut*(.)

```
1: for each PM $p$ in $P$ do
2:     if $p$ is enough for $f$ and meets $min(H(x) + \beta \cdot B(Y))$ then
3:         create a new VNF instance $v_{ik+1}$ on $p$
4:         choose $v_{ik+1}$ for $f$
5:         return true
6: return false
```

The scaling out problem is also NP-hard. Therefore, the scaling out algorithm is a heuristic solution. Procedure *ScaleOut(.)* is shown in Alg. 5. Let P denote all the PMs in cloud datacenter. For each PM p in P (line 1), if PM p has sufficient resource for flow f and satisfy equation 19 (line 2), we create a new VNF instance v_{ik+1} on PM p and choose this VNF instance v_{ik+1} to process flow f (lines 3-4).

The time complexity of Rubik algorithm is bounded by $O(L \cdot (F^2 \cdot V + S \cdot \log S))$ in the worst-case, where L is the size of flow, F is the number of flows on a service chain, V is the max number of VNF instances of a particular function of a service chain, S is the number of PM. More specifically, the Route algorithm is $O(L \cdot V)$, the Migrate algorithm is $O(L \cdot F^2 \cdot V)$, the ScaleOut algorithm is $O(L \cdot S \cdot \log S)$.

IV. Evaluation

In this section, we use a simulation approach to evaluate the performance of ENSC. Specifically, our experiments evaluate the acceptance ratio and resource utilization of ENS, and we compare ENSC with horizontal scaling method and vertical scaling method respectively.

A. Experimental Setup

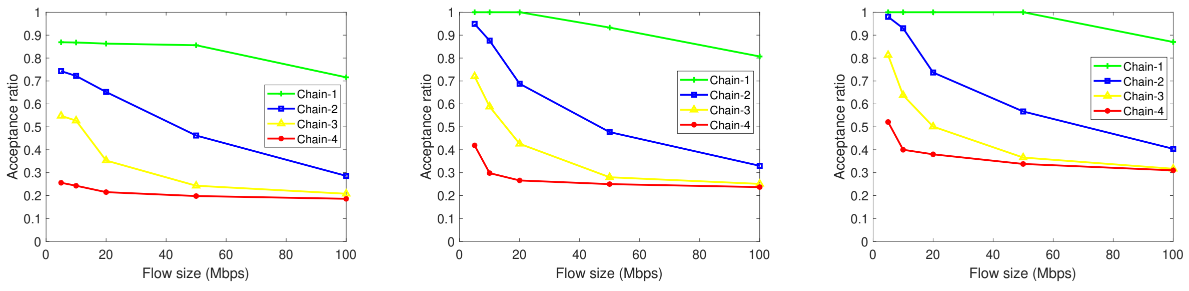
1) Simulated Cloud Datacenter: We developed a simulator of a cloud datacenter. We used a 3-level tree topology to simulate the datacenter network. The cloud datacenter has one core switch, four aggregation switches, 80 ToR switches and 1600 servers. The oversubscription rate of the physical network is 2. For simplicity, we used a homogenous PM type with identical CPU (12 vCPUs), memory (32GB), and bandwidth capacity (1Gbps). It is straightforward to extend the model to a heterogeneous environment. The upper-bound bandwidth of PM, ToR switches, and aggregation switches are 1Gbps, 10Gbps, and 100Gbps, respectively.

2) Service Chain and Parameters: We selected the firewall (Level 1 [24]), IDS (Suricata [13]), IPSec (Click [15]), and WAN-opt (CCX770M [25]) as VNFs and interconnected the four VNFs to achieve a service chain. We assessed ENSC in respect to the size of flows and the length of chains. In each experiment, the size of flows was selected from the set {5, 10, 20, 50, 100} Mbps, and one of the following chains was selected as experimental scenarios.

- Chain-1: {firewall},
- Chain-2: {firewall $\rightarrow$ IDS},
- Chain-3: {firewall $\rightarrow$ IDS $\rightarrow$ IPSec}, and
- Chain-4: {firewall $\rightarrow$ IDS $\rightarrow$ IPSec $\rightarrow$ WAN-opt}.

We measured the CPU, memory and bandwidth demand of the above four VNF chains under the five flow sizes. The Poisson distribution with an average inter-arrival rate (λ) of 1-request per 5-milliseconds was used to simulate the arrival process. The request lifetime followed an exponential distribution with an average of one hour, which also determines the departure process with departure rate μ. We did not consider the bandwidth usage from sources to the first VNFs and from the last VNFs to the targets.

3) Evaluation Method: To understand the performance of ENSC, we compared ENSC with some horizontal scaling methods such as FreeFlow [4] and some vertical scaling methods such as ElasticNFV [8]. For each parameters setting, we repeated the experiment 20 times for every 50000 requests generated, and reported the average afterwards. Two types of requests are considered in our experiments: the arrival of new flows and the update of existing flows.



(a) FreeFlow. (b) ElasticNFV. (c) ENSC.

Fig. 4: Acceptance ratio: horizontal scaling vs. vertical scaling vs. hybrid scaling.

B. Acceptance Ratio

Fig. 4 depicts the acceptance ratios of FreeFlow, ElasticNFV and ENSC, respectively. The values are the average of acceptance ratios from the 20 experiments. As we have expected, the larger flows in longer chains are less likely to be accepted. The low acceptance ratio for Chain-4 is due to the resource starvation from those chains, especially from WAN-opt which has a high demand of CPU and memory resources.

Fig. 4(a) demonstrates the range of the numbers of requests accepted by FreeFlow: 72%-87% for Chain-1, 29%-74% for Chain-2, 21%-55% for Chain-3, and 19%-26% for Chain-4. As shown in Fig. 4(b), the acceptance ratios for ElasticNFV are 81%-100% for Chain-1, 33%-95% for Chain-2, 25%-73% for Chain-3 and 24%-42% for Chain-4. The results of ENSC are shown in Fig. 4(c). The acceptance ratios are: 87%-100%, 40%-98%, 32%-81% and 31%-52% for the four type of chains respectively. We can see that ENSC has similar acceptance ratio as ElasticNFV in terms of small flow sizes and short chains. In both cases very few scaling up conflicts occurred in this situation. As the length of the chains and the size of flows increases, ElasticNFV has a higher acceptance ratio than FreeFlow, and ENSC has a higher acceptance ratio than ElasticNFV. Recalling from Section II, the goal of ENSC is to improve the efficiency and scalability on top of horizontal scaling and vertical scaling methods. Overall, ENSC has a more competitive acceptance ratio than FreeFlow and ElasticNFV.

C. Resource Utilization

Fig. 5 shows the comparison results of CPU, memory and bandwidth utilization of ENSC and FreeFlow. The resource utilization is the ratio of allocated and used resource over physical resource capacities in the cloud datacenter. For ENSC, we provided a fine-grained hybrid scaling for resource provisioning. Therefore, the used resource is equalalent to the allocated resource.

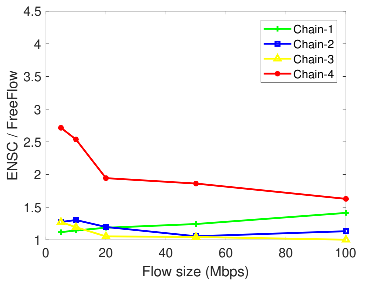
The CPU utilization ratios for ENSC/FreeFlow, as depicted in Fig. 5(a), are 112%-141% for Chain-1, 106%-130% for Chain-2, 100%-127% for Chain-3, and 163%-272% for Chain-4. The memory utilization ratios for ENSC / FreeFlow are 121%-135% for Chain-1, 106%-134% for Chain-2, 100%-157% for Chain-3, and 169%-274% for Chain-4 (Fig. 5(b)). Fig. 5(c) shows that the bandwidth utilization ratios for ENSC / FreeFlow are 100%-100% for Chain-1, 122%-151% for Chain-2, 101%-132% for Chain-3, and 163%-429% for Chain-4. Fig. 5 shows that ENSC efficiently utilizes the CPU, memory and bandwidth resources for all chains under various throughput demands. Regarding Chain-4, ENSC's efficiency in utilizing CPU and memory resources increases due to the high CPU/memory demand from WAN-opt. Regarding Chain-1, there is only one VNF without bandwidth cost between VNFs. Therefore, the bandwidth utilization of ENSC is 0, which is the same as FreeFlow in this situation.

Fig. 6 shows the comparison results of CPU, memory and bandwidth utilization of ENSC with ElasticNFV. As we know, ElasticNFV provides a fine-grained vertical scaling for resource provisioning. Therefore, the used resource from ElasticNFV is also the same as the allocated resource.

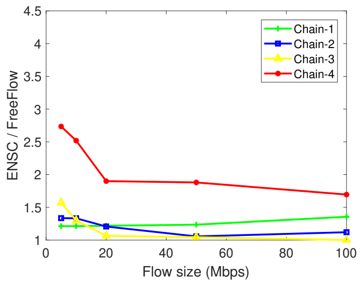
The CPU utilization ratios for ENSC / ElasticNFV, as shown in Fig. 6(a), are 99%-108% for Chain-1, 99%-107% for Chain-2, 101%-105% for Chain-3, and 102%-110% for Chain-4. The memory utilization ratios for ENSC / ElasticNFV are 99%-107% for Chain-1, 99%-106% for Chain-2, 101%-106% for Chain-3, and 103%-109% for Chain-4 (Fig. 6(b)). As shown in Fig. 6(c), the bandwidth utilization ratios for ENSC / ElasticNFV are 100%-100% for Chain-1, 104%-156% for Chain-2, 108%-116% for Chain-3, and 139%-169% for Chain-4. Fig. 6 shows that ENSC has a higher CPU, memory and bandwidth resources utilization for all chains under various throughput-demands than ElasitcNFV. In terms of Chain-2, Chain-3 and Chain-4, ENSC's efficiency in utilizing bandwidth resources is higher than ElasticNFV.

V. Conclusion

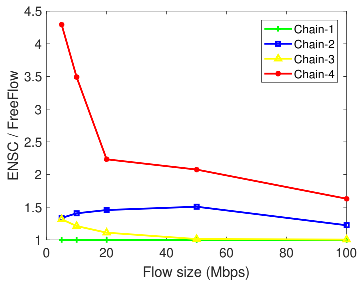
In this paper, we present ENSC - an elastic network service chain solution that exploits fine-grained hybrid scaling to achieve NFV efficiency and scalability. We analyzed the priority within hybrid scaling to improve the performance of ENSC. For resource allocation in cloud datacenter, we formulated the ENSC deployment in cloud datacenter using an ILP model. We proposed and evaluated a heuristic algorithm called Rubik for larger scale networks. The experimental



(a) Average CPU utilization.

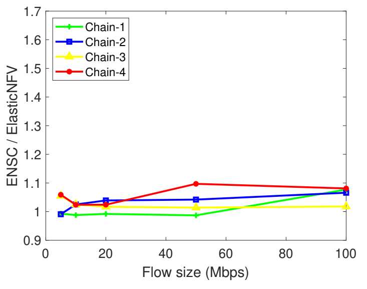


(b) Average memory utilization.

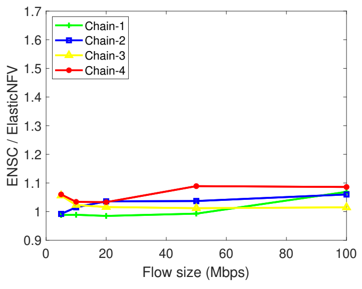


(c) Average bandwidth utilization.

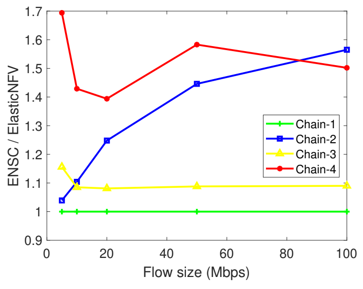
Fig. 5: Average utilization: hybrid scaling vs. horizontal scaling.



(a) Average CPU utilization.



(b) Average memory utilization.



(c) Average bandwidth utilization.

Fig. 6: Average utilization: hybrid scaling vs. vertical scaling.

results under various chain lengths and throughput demands demonstrate that ENSC achieves higher acceptance ratios and resource utilization than horizontal scaling and vertical scaling methods.

VI. Acknowledgment

This work was supported by the National Natural Science Foundation of China (No.61432009).

References

[1] R. Mijumbi, J. Serrat, J. Gorricho, N. Bouten, F. Turck, and R. Boutaba, "Network function virtualization state-of-the-art and research challenges," in *IEEE COMMUNICATIONS SURVEYS & TUTORIAL*, 2016.
[2] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, and V. Sekar, "Making middleboxes someone elses problem: network processing as a cloud service," in *ACM SIGCOMM Computer Communication Review*, 2012.
[3] C. Matsumoto, "Leading lights 2017 finalists: Most innovative nfv product strategy (vendor)," 2017. http://www.lightreading.com/nfv/nfv-strategies/leading-lights-2017-finalists-most-innovative-nfv-product-strategy-(vendor)-/d/d-id/732749.
[4] S. Rajagopalan, D. Williams, H. Jamjoom, and A. Warfield, "Split/merge: System support for elastic execution in virtual middleboxes," in *USENIX NSDI*, 2013.
[5] S. Palkar, C. Lan, S. Han, K. Jang, A. Panda, S. Ratnasamy, L. Rizzo, and S. Shenker, "E2: A framework for nfv applications," in *ACM SOSP*, 2015.
[6] A. Ghodsi, V. Sekar, M. Zaharia, and I. Stoica, "Multi-resource fair queueing for packet processing," in *ACM SIGCOMM*, 2015.
[7] M. Mao and M. Humphrey, "A performance study on the vm startup time in the cloud," in *IEEE International Conference on Cloud Computing*, 2012.
[8] H. Yu, J. Yang, and C. Fung, "Elastic network service chain with fine-grained vertical scaling," in *IEEE IWQoS*, 2018.
[9] L. Cao, P. Sharma, S. Fahmy, and V. Saxena, "Nfv-vital: A framework for characterizing the performance of virtual network functions," in *IEEE NFV-SDN*, 2017.
[10] "Snort," 2017. https://www.snort.org/.
[11] H. Dreger, A. Feldman, V. Paxson, and R. Sommer, "Predicing the resource consumption of network intrusion detection systems," in *RAID*, 2008.
[12] A. Anand, V. Sekar, and A. Akella, "Smartre: An architecture for coordinated network-wide redundancy elimination," in *ACM SIGCOMM*, 2016.
[13] "Suricata," 2016. https://suricata-ids.org/.
[14] "Varnish," 2009. https://varnish-cache.org/.
[15] "Click," 2009. http://read.cs.ucla.edu/click/.
[16] "Wind river," 2017. https://www.windriver.com/.
[17] L. Popa, P. Yalagandula, S. Banerjee, and J. Mogul, "Elasticswitch: Practical work-conserving bandwidth guarantees for cloud computing," in *ACM SIGCOMM*, 2013.
[18] A. Jacobson, R. Viswanathan, C. Prakash, and R. Grandl, "Opennf: Enabling innovation in network function control," in *ACM SIGCOMM*, 2014.
[19] N. Farrington and A. Andreyev, "Facebooks data center network architecture," in *IEEE Optical Interconnects Conf*, 2013.
[20] Z. Ren, Z. Feng, and A. Zhang, "Fusing ant colony optimization with lagrangian relaxation for the multiple-choice multidimentional knapsack problem," in *Information Sciences*, 2012.
[21] A. Sbihi, "A best first search exact algorithm for the multiple-choice multidimensional knapsack problem," in *Journal of Combinatorial Optimization*, 2007.
[22] M. Hifi, M. Michrafy, and A. Sbihi, "Heuristic algorithm for the multiple-choice multidimensional knapsack problem," in *JORS*, 2004.
[23] R. Hernandez, D. Jokanovic, and N. Shiratori, "A new heuristic for solving the multichoice multidimensional knapsack problem," in *IEEE Transactions on System*, 2005.
[24] "Barracuda wap," 2017. https://www.barracuda.com/.
[25] "Steelhead product family," 2017. https://www.riverbed.com/.