Elastic Virtual Network Function Placement

Milad Ghaznavi, Aimal Khan, Nashid Shahriar, Khalid Alsubhi, Reaz Ahmed, Raouf Boutaba
David R. Cheriton School of Computer Science, University of Waterloo, ON, Canada
{eghaznav | a273khan | nshahria | kaalsubhi | r5ahmed | rboutaba}@uwaterloo.ca

***Abstract*— Nowadays, many cloud providers offer Virtual Network Function (VNF) services that are dynamically scaled according to the workload. Enterprises enjoy these services by only paying for the actual consumed resources. From a cloud provider's standpoint, the cost of these services must be kept as low as possible, while QoS is maintained and service downtime is minimized. In this paper, we introduce Elastic Virtual Network Function Placement (EVNFP) problem and present a model for minimizing operational costs in providing VNF services. In this model, the elasticity overhead and the trade-off between bandwidth and host resource consumption are considered together, while the previous works ignored this perspective of the problem. We propose a solution called Simple Lazy Facility Location (SLFL) that optimizes the placement of VNF instances in response to on-demand workload. Our experiments suggest that SLFL can accept two times more workload while incurring similar operational cost compared to first-fit and random placements.**

I. Introduction

Nowadays, many Cloud Providers (CPs) such as Amazon AWS, Microsoft Azure and IBM Bluemix offer VNF as a service (VNFaaS). An Enterprise Client (EC) can deploy all or part of its Network Functions (NFs) to cloud and enjoy the pay-per-use pricing model. Thus, an EC can eliminate the cost of provisioning NFs for peak-load on its own premises and only pay for the actually used resources in cloud. This can greatly reduce an EC's capital and operational expenditures.

From a CP standpoint, a core management problem to offer VNFaaS is placing VNF instances in the cloud infrastructure, and allocate resources to these instances elastically according to VNF service requests and workloads. The gaol is to utilize available bandwidth and host resources optimally without violating Service Level Agreements (SLAs). However, elastically allocating and releasing resources incur costs [21]. A VNF placement algorithm should consider these costs, as well as the overall consumption of bandwidth and host resources.

Existing works in the VM placement are not suitable for the placement of VNFs for the reasons described by Bouet et al. [6]. Moreover, most of these works consider only a part of the problem by optimizing either host or bandwidth resource [4], [10]. Additionally, the elastic VNF placement area has not been explored sufficiently, and the few recent works [6], [8], [17] do not address the challenges that may arise due to *conflicting objectives* and *elasticity*, as discussed below:

Conflicting Objectives: An optimal VNF placement algorithm should minimize bandwidth and host resource consumption. Host resource consumption can be minimized by serving the VNF request using minimum number of VNF instances, which may be many hops away from the source or target of the request resulting into high bandwidth consumption. Bandwidth consumption, on the other hand, can be minimized by placing dedicated VNF instance at the source or target of each request, but in this case host resource consumption will be high. Therefore, the challenge is to find a trade-off between host and bandwidth resources consumption.

Elasticity: Horizontal and *vertical* scaling are the prominent mechanisms for achieving elasticity. Horizontal scaling is *installation/removal* of VNF instances, whereas vertical scaling is *allocation/release* of host and bandwidth resources to/from a VNF instance. Further, live *migration* of VNF instances [7] and *reassignment* of partial workload to another VNF instance [13] can be employed to scale bandwidth and host resources. The challenge is to determine which of these elasticity mechanisms is the most appropriate for a given workload.

To address the above challenges, we introduce *Elastic Virtual Network Function Placement (EVNFP)* problem and propose an efficient solution called *Simple Lazy Facility Location (SLFL)*. Our major contributions are as follows:

1) We formulate EVNFP considering the challenges of conflicting objectives and elasticity.
2) We propose SLFL as a solution approach that optimizes placement of VNF instances in response to new service requests, and workload variation.
3) The effectiveness of our solution is examined through realistic experiments. The results suggest that SLFL can accept two times more workload with 5–8% less operational cost compared to first-fit and random placements.

The rest of this paper is organized as follows. In Section II, related works are studied. EVNFP model and our solution are presented in Sections III and IV, respectively. The proposed solution is evaluated in Section V. Finally, the paper is concluded in Section VI.

II. Related Works

Many mechanisms [11] have been proposed to support elasticity through horizontal scaling, vertical scaling, and migration techniques.

Horizontal Scaling: Amazon Auto Scaling Group [9] offers tenant controlled horizontal scaling based on tenant-defined thresholds. Microsoft Azure [1] adapts the number of instances based on time, history, or size of workload. Clayman et al. [8] focus on dynamic VNF placement to satisfy increasing demand by installing new virtual routers; however, no mechanism to release resources is supported. Another related area is elasticity in VNF service-chain embedding. Stratos [12] uses a simple packing technique to elastically scale resources.

Vertical Scaling: CloudScale [22] and PRESS [15] scale by releasing or allocating CPU resources while ignoring network resources. Vertical mechanisms are limited to individual physical machines [20]. Furthermore, changing compute or memory resources on-the-fly is not supported in most cases. Moreover,

Table I: Comparison of EVNFP to the Most Related Works

Paper	Host	Bandwidth	Elasticity
EVNFP	✓	✓	✓
Elasticity in cloud [15], [21], [22]	✓	✗	✓
Dynamic VM Placement [21], [23]	✓	✗	✓
Network-Aware VM Placement [5], [18], [19]	✓	✓	✗
vDPI Placement [6]	✓	✓	✗

it requires rebooting the system causing SLA violations. Therefore, CPs do not encourage vertical scaling mechanisms.

Migration: Migration is a popular technique to achieve elastic VM placement and better server and network consolidation. pMapper [23] considers VM migration cost in its greedy heuristics to solve the optimal VM placement problem. Entropy [16] models the optimal VM placement as a variant of the vector bin-packing. However, both pMapper and Entropy ignore network requirement and locality in placement decisions. Kingfisher [21] employs both vertical and horizontal elasticity mechanisms by allocating more host resources, installing and migrating VM instances. It optimizes host resources from tenants standpoint while considering delay of these mechanisms. Although, bandwidth resources are ignored.

MCRVMP [5] addresses the static VM placement problem to satisfy the time-varying traffic demands of the VMs in addition to CPU and memory requirements. TVMPP [19] strives to reduce the aggregate traffic. However, it can lead congested links by ignoring link capacity constraints. The authors of NAVP [18] focus on consolidating as much traffic demands as possible over the same set of network links in order to reduce the total energy consumption.

Recently, Bouet et al. [6] studied the placement of virtual DPI engines to optimize both the number of installed engines and their network footprint. However, the formulated problem is static and cannot handle elastic VNF placement.

Table I briefly compares the most related works mentioned above. It is evident from the table that none of the existing works consider all three aspects considered by EVNFP.

III. Elastic Virtual Network Function Placement

From a cloud provider perspective, a VNF request can be modeled as follows: traffic (stream of packets) originating from a source is routed to a VNF instance, where the packets are processed and forwarded to a target. If the source or target is outside the datacenter, we can assume a border router as the source or target of the traffic. VNF instances reside at hosts within datacenter.

We are interested in an optimal cost-aware elastic placement of VNF instances involving (i) selecting an optimal set of hosts on which VNF instances are placed (ii) optimally allocating bandwidth resources to route service traffic, and (iii) applying the most cost-effective elasticity mechanism.

When to elastically scale? Resources must be scaled in response to two events: i) *Demand arrival*: when workload increases or a new service request is received; and ii) *Demand departure*: when one unit of service traffic drops. In addition, for the sake of simplicity, we assume that each demand consists of one unit of traffic transmitted from its source to a VNF instance and delivered to its target, and requires a unit of VNF resource to be served.

How to elastically scale? Due to the inflexibility in vertical scaling, we opt for horizontal scaling. We also assume one VNF instance-type. A small instance-type is lightweight, can be easily distributed over the network and instantiated on-demand. Moreover, having multiple instance-types unnecessary complicates management tasks. In summary, we consider following elasticity mechanisms: *Installing* a new VNF instance, *Removing* an existing VNF instance, *Migrating* a VNF instance, and *Reassigning* a demand to another VNF instance.

Fig. 1 depicts the above elasticity mechanisms. Initially in Fig. 1a, traffic from 3 requests is served by a single VNF instance v. After sometime the traffic of the first request increases. To accommodate this new workload, a new VNF v^* is instantiated and the first request is reassigned to v^* (Fig. 1b). Then, traffic of the second request terminates, and allocated resources for this request are released. Because, v is still serving the third request, it is migrated to a more optimal location to reduce bandwidth usage (Fig. 1c). Next, third request terminates, and VNF instance v is removed to save host resources (Fig. 1d).

A. Notation

1) Data Center Network: The data center network is denoted as a graph $G = (N, A)$, where N is a set of switches and hosts, while A is a set of links (arcs). We identify host nodes by N_H. Capacity of host $n \in N_H$ is denoted by the maximum number of VNF instances that can be installed on n. Let $c_n(t)$ denote the available capacity of $n \in N_H$ at time t. Let w_{mn} and $c_{mn}(t)$ represent weight and capacity at time t of arc $(m, n) \in A$, respectively.

2) Demand: $D(t)$ is the set of demands at time t. A demand $d \in D(t)$ is identified by its two endpoints, source $m_d \in N$ and target $n_d \in N$. Let *demand nodes* refer to sources and targets. A demand needs a unit of traffic b and a unit VNF resource to be served.

3) VNF Instance: $V(t)$ is the set of installed VNF instances at time t. $c_v(t)$ denotes the number of demands that VNF instance v can serve at time t. As we use one instance-type, we assume the maximum capacity is C. To show assignments of demands to VNF instances, we use two maps at time t: $D_v(t)$ denoting a set of demands assigned to v and $v_d(t)$ representing the VNF instance to which demand d is assigned.

B. Simplified Model

We refine the above model from two aspects in order to simplify our formulation. First, host and VNF instance constraints are transformed to arc bandwidth capacity constraints; second, demands are simplified:

1) Constraints Transformation: For each host $n \in N_H$, $c_n(0)$ nodes are added, and we assume that VNF instances are installed on these nodes. These nodes are called *VNF nodes*. Imagine a VNF instance v is installed on n at time t, and n still has capacity of $c_n(t) = 2$. As shown in Fig. 2a, three nodes are added. $F_n(t)$ denotes these nodes, and $F(t) = \bigcup_{n \in N_H}(F_n(t))$. Each $m \in F_n(t)$ is connected to n via arc (m, n). The capacity of arcs initially are set to $2 \times b \times C$,

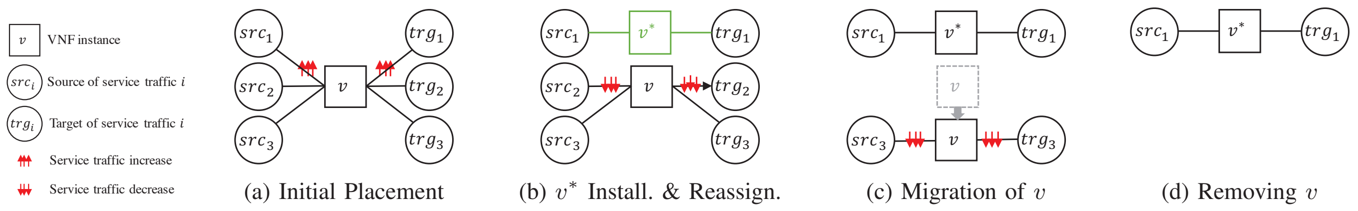


Figure 1: Elasticity Mechanisms

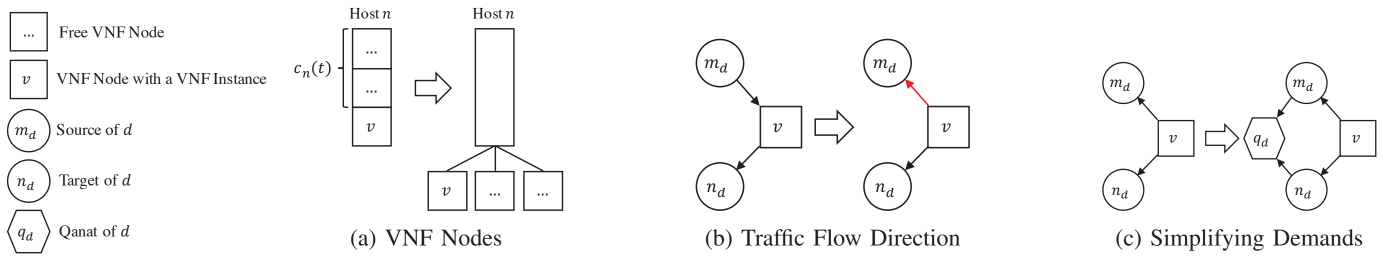


Figure 2: Simplified Model

as at most, traffic of C number of demands enters and leaves m. These arcs force that no traffic can enter a VNF instance node; however, this does not change the problem, because we can assume that the demand traffic is sent to a demand source from the VNF instance node, instead of the opposite direction (Fig. 2b). Let $A^F_n(t)$ represent the arcs connecting VNF nodes to the node n, and $A_F(t) = \bigcup_{n \in N_H}(A^F_n(t))$. Finally, let $n_v(t) \in F(t)$ be the VNF instance node hosting VNF instance v at time t, and $N_V(t) = \bigcup_{v \in V(t)}\{n_v(t)\}$.

2) Simplifying Demands: By previous transformation, we assume that VNF instances send the traffic to demand nodes. We add a node q_d called Qanat to the graph for each demand $d \in D(t)$. Traffic is now received in q_d instead of m_d and n_d (Fig. 2c). Let $Q(t) = \bigcup_{d \in D(t)}\{q_d\}$ denotes all Qanat nodes at time t. A q_d is connected to m_d and n_d via arcs (m_d, q_d) and (n_d, q_d). The capacity of these arcs initially are set to b, and their weights are set to 0. These two arcs ensure that if traffic reaches q_d, it has met m_d and n_d earlier.

C. Mathematical Model

A discrete-time system is considered to model the problem in which time is divided into equal slots $0 \leq t \leq T$.

1) Decision Variables: $x^d_{mn}(t) \in \mathbb{R}$ is the amount of traffic for demand d on arc $(m,n) \in A$ at time t. We derive two variables $y^d_n(t)$ and $z_n(t)$ from $x^d_{mn}(t)$. $y^d_n(t)$ denotes if demand d gets traffic from VNF instance node n. $z_n(t)$ represents if a VNF instance is installed in node n. They are defined as follows.

$$\forall (n,m) \in A_F(t) : y^d_n(t) = \begin{cases} 1 & \text{if } x^d_{nm}(t) > 0 \\ 0 & \text{otherwise} \end{cases}$$

$$\forall n \in F(t) : z_n(t) = \begin{cases} 1 & \text{if } \sum_{d \in D(t)} y^d_n(t) > 0 \\ 0 & \text{otherwise} \end{cases}$$

2) Capacity Constraint: Eq. 1 ensures that arcs capacities are not violated.

$$0 \leq \sum_{d \in D(t)} x^d_{mn}(t) \leq c_{mn}(t) \text{ for } \forall (m,n) \in A \tag{1}$$

3) Flow Conservation Constraint: For each node $n \in N$, Eq. 2 guarantees that the amount of traffic entering and leaving n are equal, where n is not a VNF node or a Qanat.

$$\sum_{(m,n) \in A} x^d_{mn}(t) - \sum_{(n,m) \in A} x^d_{nm}(t) = \begin{cases} -2by^d_n(t) & \text{if } n \in F(t) \\ 2b & \text{if } n \in Q(t) \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

4) Pair Connectivity Constraint: Eq. 3 ensures that a Qanat receives traffic from a VNF instance, so both source and target of a demand are connected to the same VNF instance.

$$\forall d \in D(t) : \sum_{n \in F(t)} y^d_n(t) = 1 \tag{3}$$

5) Installations Cost ($\mathcal{C}_{ins}(t)$): The cost of installed VNF instances at time t is defined by Eq. 4. f is the cost of host resources consumed by a VNF instance for each time slot.

$$\mathcal{C}_{ins}(t) = f \sum_{n \in F(t)} z_n(t) \tag{4}$$

6) Transportation Cost ($\mathcal{C}_{tr}(t)$): The cost of delivering demands traffic at time t is denoted by Eq. 5. g is the cost of a unit of bandwidth usage for each time slot.

$$\mathcal{C}_{tr}(t) = g \sum_{d \in D(t)} \sum_{(m,n) \in A} x^d_{mn}(t) w_{mn} \tag{5}$$

7) Reassignment Cost ($\mathcal{C}_{re}(t)$): Eq. 6 is the cost of reassigning a set of demands at time t. In this equation, $h_d(t)$ is the penalty of reassigning demand d. Here, $|y^d_n(t-1) \neq y^d_n(t)|$ is 1 if $y^d_n(t-1) \neq y^d_n(t)$ otherwise is 0.

$$\mathcal{C}_{re}(t) = \sum_{n \in F(t) \cap F(t-1)} \sum_{d \in D(t) \cap D(t-1)} \big(h_d(t) \; |y^d_n(t-1) \neq y^d_n(t)|\big) \tag{6}$$

8) Migration Cost ($\mathcal{C}_{mig}(t)$): This is the cost of migrating a set of VNF instances at time t. In Eq. 7, $k_v(t)$ is the penalty of migrating VNF instance v. Here, $|z_n(t-1) \neq z_n(t)|$ is 1 if $z_n(t-1) \neq z_n(t)$, otherwise is 0.

$$\mathcal{C}_{mig}(t) = \sum_{n \in F(t) \cap F(t-1)} \big(k_v(t) |z_n(t-1) \neq z_n(t)| \big) \quad (7)$$

9) Objective Function: The objective is to minimize Eq. 8.

$$\lim_{T \to \infty} \frac{1}{T} \sum_{t=0}^{T} \big(\mathcal{C}_{ins}(t) + \mathcal{C}_{tr}(t) + \mathcal{C}_{re}(t) + \mathcal{C}_{mig}(t) \big) \quad (8)$$

The static version of EVNFP generalizes to the NP-Hard location routing problem (LRP) [3]. An optimal solution needs solving a dynamic version of LRP for each time slot. Due to its intractability, it is not possible to solve the problem for large datacenters. Thus, we break down the problem and solve it independently for each demand arrival or departure. Let $\hat{T}$ be a time system, and at each $t \in \hat{T}$, an arrival or departure occurs. We rewrite the objective function as Eq. 9. Here, λ_t is a weight factor to balance the transportation and installation costs with the migration and reassignment costs.

$$\min \Big(\sum_{t \in \hat{T}} (\mathcal{C}_{ins}(t) + \mathcal{C}_{tr}(t)) + \sum_{t \in \hat{T}} \lambda_t (\mathcal{C}_{re}(t) + \mathcal{C}_{mig}(t)) \Big) \quad (9)$$

Although, this problem is easier than the original one, the static version still generalizes to LRP. Hence, we propose a heuristic algorithm for solving this problem in the next section.

IV. Simple Lazy Facility Location

In this section, we describe our solution, Simple Lazy Facility Location (SLFL), including two novel heuristics to handle arrival and departure events. For simplicity, we omit time variable t. Also, we assume function $flow(n, D^*, R^*)$ that finds the optimal routing of traffic between node n and demands D^*, and stores the routes in R^*. This function finds R^* in polynomial time by solving the single-commodity min-cost flow problem [14].

A. Demand Arrival

Upon new demand d arrival, a combination of installation, migrations and reassignments can be applied to optimize the placement. Since in most cases the arrival affects its locality, SLFL locally optimizes the placement. Three possible actions are considered: (i) assignment to an existing VNF instance, (ii) migration and (iii) installing a new VNF instance followed by a set of reassignments. The first action assigns d to an existing VNF instance with the minimum transportation cost. For the two other actions, *migration potential* and *installation potential* metrics are defined as follows:

Migration potential is the difference between current transportation cost of D_v and transportation cost of $D_v \cup \{d\}$ after possible migration of v to node n. Function pot_{mig}, defined in algorithm 1, finds this potential and stores routes in R^*.

Installation potential is the difference between the operational cost before and after installing a VNF instance in node n. We consider a set of reassignments during installation. Function pot_{ins}, as defined in Algorithm 2, computes the installation potential for a node n, finds candidate demands D^* for reassignment, and stores routes in R^*.

Algorithm 1 Migration Potential

```
1: function $pot_{mig}(v, n, R^*)$
2:     $\mathcal{C} \leftarrow$ Transportation cost of $D_v$;
3:     $\mathcal{C}^* \leftarrow g \times flow(n, D_v \cup \{d\}, R^*) + \lambda \times k_v$;
4:     return $(\mathcal{C} - \mathcal{C}^*)$ if ($\mathcal{C}^*$ is not $\infty$), otherwise $-\infty$;
5: end function
```

Algorithm 2 Installation Potential

```
1: function $pot_{ins}(n, R^*)$
2:     $e^* \leftarrow -g \times flow(n, \{d\}, R^*)$; $D^* \leftarrow \emptyset$;
3:     for $j = 1$ to $C - 1$ do
4:         $d^* \leftarrow$ Find best next demand to reassign;
5:         $e \leftarrow C - (g \times flow(n, D^* \cup \{d, d^*\}, \mathcal{R}) + \lambda \times h_{d^*})$;
6:         break if ($e \leq 0$ or $e \leq e^*$);
7:         $e^* \leftarrow e$; $D^* \leftarrow D^* \cup \{d^*\}$; $R^* \leftarrow \mathcal{R}$;
8:     end for
9:     return $(e^* - f, D^*)$;
10: end function
```

Alg. 3 shows how SLFL handles a demand arrival. The cost of the best assignment is found (lines 3-4). The best migration and installation potential are computed (lines 4-8). If the best installation potential is greater than the best migration potential, a new VNF v is instantiated and demands D_{re} are reassigned to v (lines 9-14). Otherwise, VNF instance v_{mig} is migrated to n_{mig} (lines 14-17). Finally, in the lack of potential to change, d is assigned to VNF instance v_{asn}.

Algorithm 3 SLFL-Demand Arrival

```
1: function DemandArrival($d$)
2:     $D \leftarrow D \cup \{d\}$;
3:     $v_{asn} \leftarrow \arg\min_{v \in V} \{flow(n_v, \{d\}, \emptyset)\}$
4:     $p_{asn} \leftarrow g \times flow(n_{v_{asg}}, \{d\}, R^*_{asg})$;
5:     $(v_{mig}, n_{mig}) \leftarrow \arg\max_{v \in V : n \in F} \{pot_{mig}(v, n, \emptyset)\}$;
6:     $e_{mig} \leftarrow pot_{mig}(v_{mig}, R^*_{mig})$;
7:     $n_{ins} \leftarrow \arg\max_{n \in F/N_V} \{pot_{ins}(n, \emptyset)\}$;
8:     $(e_{ins}, D_{re}) \leftarrow pot_{ins}(n_{ins}, R^*_{ins})$;
9:     if $(e_{ins} > -p_{asn})$ and $(e_{ins} \geq e_{mig})$ then
10:        $u \leftarrow$ install a facility at $n_{ins}$;
11:        Reassign $\forall d \in D_{re}$ and assign $d$ to $u$;
12:        Route related traffic based on $R^*_{ins}$;
13:        $V \leftarrow V \cup \{u\}$;
14:    else if $e_{mig} > -p_{asn}$ then
15:        Migrate $v_{mig}$ to node $n_{mig}$;
16:        Assign $d$ to $v_{mig}$;
17:        Route traffic based on $R^*_{mig}$;
18:    else
19:        Assign $d$ to $v_{asn}$;
20:        Route traffic based on $R^*_{asg}$;
21:    end if
22: end function
```

B. Demand Departure

Similar to an arrival event, SLFL locally optimizes the placement of VNF instances upon departure of demand d. Assume that d was assigned to VNF instance v. Two actions are considered: (i) migration of v, and (ii) removal of v. We define *emigration potential* and *removal potential* metrics for migration and removal of v as follows:

Emigration potential defined in Alg. 4 is the difference in transportation cost of D_v before and after migration of v to node n.

Algorithm 4 Emigration Potential

```
1: function $pot_{emg}(v, n, R^*)$
2:     $\mathcal{C} \leftarrow$ Transportation cost of $D_v$;
3:     $\mathcal{C}^* \leftarrow g \times flow(n, D_v, R^*) + \lambda \times k_v$;
4:     return $(\mathcal{C} - \mathcal{C}^*)$ if $\mathcal{C}^*$ is not $\infty$, otherwise $-\infty$;
5: end function
```

Removal potential is the difference in operational-cost before and after the removal of v. Similar to installation potential, a set of reassignments are considered. Function pot_{rmv}, defined in Alg. 5, computes the removal potential of v, finds candidate VNF instances V_{re} for reassignment of D_v, and stores routes in R^*.

Algorithm 5 Removing Potential

```
1: function $pot_{rmv}(v, R^*)$
2:     $\{D_{re}, V_{re}\} \leftarrow \{\emptyset, \emptyset\}$; $e^* \leftarrow f$;
3:     $\mathcal{C} \leftarrow$ Transportation cost of $D_v$;
4:     $\mathcal{C}^* \leftarrow 0$; $U \leftarrow V/\{v\}$;
5:     for all $d_v \in D_v$ do
6:         $v_{re} \leftarrow$ best VNF instance for reassignment of $d_v$;
7:         $\mathcal{C}^* \leftarrow \mathcal{C}^* + g \times flow(n_{v_{re}}, \{i\}, R^*_{re}) + \lambda \times h_{d_v}$;
8:         if $\mathcal{C}^*$ is $\infty$ then
9:             $e^* \leftarrow 0$; $\{D_{re}, V_{re}\} \leftarrow \{\emptyset, \emptyset\}$;
10:            break;
11:        end if
12:        $\{V_{re}, D_{re}\} \leftarrow \{V_{re} \cup v_{re}, D_{re} \cup d\}$;
13:        $R^* \leftarrow R^* \cup R^*_{re}$;
14:    end for
15:    return $\big(e^* + (\mathcal{C} - \mathcal{C}^*), \{D_{re}, V_{re}\}\big)$;
16: end function
```

Finally, algorithm 6 defines how SLFL handles a demand departure. First, v's resources assigned to d are released (line 2). Then, the best node to migrate and emigration potential are computed (lines 3-4). The removal potential and possible reassignments are determined (line 5). If removal potential is positive and greater than emigration potential, v is removed and its demands are reassigned to other VNF instances (lines 6-12). Otherwise, if emigration potential is positive, v is migrated to a more optimal node (lines 13-16).

V. Evaluation

A. Experimental Setup

We have implemented SLFL[1] and evaluated its performance by simulations on a data center topology with 99 nodes (45 switches and 54 hosts). We used a 6-ary Fat-tree topology [2] providing full bisection bandwidth. Each host has 8 CPU cores, 8GB of memory, and contains a 1Gbps network adapter. A host CPU consumes 140W of power at electricity cost of

[1]https://github.com/miladghaznavi/Elastic-VNF-Placement

Algorithm 6 SLFL-Demand Departure

```
1: function DemandDeparture$(d, v)$
2:     Release $v$'s resources assigned to $d$;
3:     $n_{emg} \leftarrow \arg\max_{n \in F/N_V} \{pot_{emg}(v, n, \emptyset)\}$;
4:     $e_{emg} \leftarrow pot_{emg}(v, n_{emg}, R^*_{emg})$;
5:     $(e_{rmv}, \{D_{re}, V_{re}\}) \leftarrow pot_{rmv}(v, R^*_{rmv})$;
6:     if $(e_{rmv} > 0)$ and $(e_{rmv} > e_{emg})$ then
7:         $V \leftarrow V/\{v\}$;
8:         Remove facility $v$;
9:         for all $\{d_{re}, v_{re}\} \in \{D_{re}, V_{re}\}$ do
10:            Reassign $d_{re}$ to $v_{re}$;
11:        end for
12:        Route traffic based to $R^*_{rmv}$;
13:    else if $e_{emg} > 0$ then
14:        Migrate $v$ to the node $n_{emg}$;
15:        Route traffic based to $R^*_{emg}$;
16:    end if
17: end function
```

¢11 per kWh[2]. We select Bro IDS [3] as a representative VNF providing a capacity of 80 Mbps. We assume that Bro can be installed on a VM which requires 1 vCPU and, 1GB of memory. In regards to bandwidth, we set the cost of using a unit of bandwidth for a link to 20% of the power consumption cost. Regrading the migration penalty $k_v(t)$, the full memory (1GB) of VNF instance v is transported from original VNF node $n_v(t-1)$ to new VNF node $n_v(t)$. The reassignment penalty $h_d(t)$ involves transporting a fraction of memory of VNF instance $v_d(t-1)$ to new VNF instance $v_d(t)$. This fraction is relative to the VNF instance's maximum capacity.

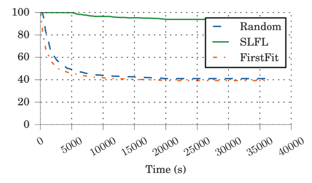
We model demand arrival using Poisson distribution with an average rate of 1 demand per second. The lifetime of a demand follows an exponential distribution with an average of 1800 seconds. Demand nodes are uniformly distributed in the data-center network. We set $b = 20$Mbps and $\lambda = 1$.

We compare SLFL with *Random* and *First-Fit* placements. Upon a demand arrival, Random placement randomly selects a VNF instance with the sufficient residual capacity and bandwidth. Otherwise, a VNF instance is installed in a random not-saturated host with enough bandwidth. First-Fit selects the first not-saturated VNF instance with adequate residual bandwidth. If not, a VNF instance is installed in the first not-saturated host with adequate available bandwidth. Upon demand departure, both algorithms remove a VNF instance if this instance has no assigned demand.

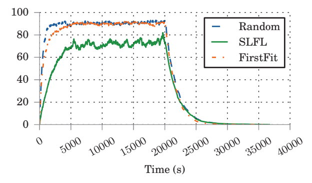
B. Acceptance and Utilization

Fig. 3 depicts workload acceptance and resources utilization. As shown in Fig. 3a, SLFL accepts 97% of workload whereas Random and First-Fit accept 48% and 45% of workload, respectively. Bandwidth, host, and VNF resource utilization are depicted in Fig. 3b, Fig. 3c and Fig. 3d, respectively. Random and First-Fit quickly exhaust bandwidth resources (utilization of 94% and 92%, respectively) causing low host resource utilization (45% and 44%, respectively).

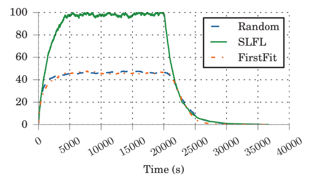
[2]Electric Power Monthly, www.eia.gov/electricity/monthly/pdf/epm.pdf.
[3]The Bro Network Security Monitor, www.bro.org



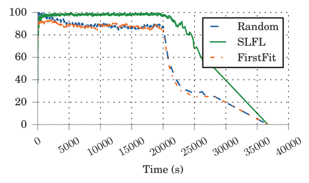
(a) Workload Acceptance



(b) Bandwidth Res. Utilization

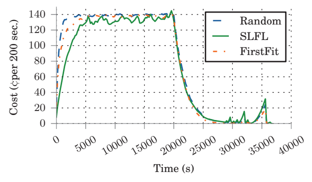


(c) Host Res. Utilization

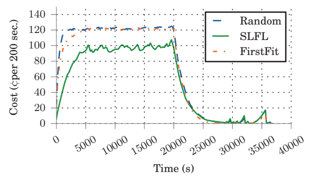


(d) VNF Res. Utilization

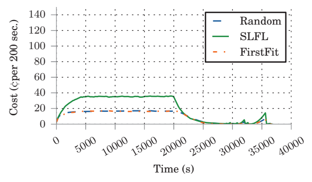
Figure 3: Workload Acceptance and Resource Utilization



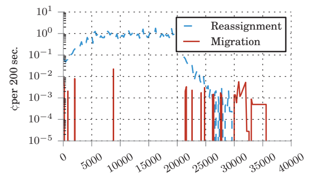
(a) Total Operational Cost



(b) Transportation Cost



(c) Installation Cost



(d) Overhead

Figure 4: Operational Costs

Moreover, these placements utilize 88% and 87% of VNF resources, respectively. SLFL achieves bandwidth, host and VNF resource utilization of 82%, 91% and 98%, respectively.

C. Operational Cost

Operational costs are reported in Fig. 4. Compared to Random and First-Fit, SLFL incurs 9% and 4% less operational cost (Fig. 4a), and pays 22% and 19% less bandwidth cost (Fig. 4b), respectively. However, SLFL incurs two times more installation cost (Fig. 4c) compared to the Random and First-Fit. The reason is that SLFL accepts two times more workload than the other approaches. Finally, Fig. 4d shows the overhead of SLFL. SLFL does reassignments more frequently than migrations. The reason is that a reassignment requires moving of a part of VNF instance state, whereas a migration requires moving of the entire memory of the VNF instance.

VI. Conclusion

We introduced Elastic Virtual Network Function Placement (EVNFP) problem presenting a model to minimize operational costs in providing VNF as a service. This model considered the elasticity overhead and the trade-off between bandwidth and host resource consumption. We developed and evaluated an algorithm, named SLFL, to solve this problem in polynomial time. Our experiments suggest that by taking both bandwidth and host resources into consideration, and by carefully selecting the right elasticity mechanism, SLFL accepts $\sim 2\times$ more workload in comparison to first-fit and random placements. Additionally, SLFL incurs 5–8% less operational cost.

VII. Acknowledgement

This work was supported by the Natural Science and Engineering Council of Canada (NSERC) under the Smart Applications on Virtual Infrastructure (SAVI) Research Network.

References

[1] Microsoft azure. http://www.microsoft.com/azure/default.mspx.

[2] M. Al-Fares, A. Loukissas, and A. Vahdat. A Scalable, Commodity Data Center Network Architecture. In *ACM SIGCOMM 2008*.

[3] M. Albareda-Sambola, J. A. Dıaz, and E. Fernández. A compact model and tight bounds for a combined location-routing problem. *Computers & Operations Research*, 32(3):407–428, 2005.

[4] A. Beloglazov et al. A taxonomy and survey of energy-efficient data centers and cloud computing systems. Academic Press, 2011.

[5] O. Biran et al. A stable network-aware vm placement for cloud systems. In *CCGRID*, pages 498–506, 2012.

[6] M. Bouet, J. Leguay, and V. Conan. Cost-based placement of vdpi functions in nfv infrastructures. In *NetSoft*, 2015.

[7] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live migration of virtual machines. In *NSDI*, 2005.

[8] S. Clayman, E. Maini, A. Galis, A. Manzalini, and N. Mazzocca. The dynamic placement of virtual network functions. In *IEEE NOMS, 2014*.

[9] Amazon EC2. http://aws.amazon.com/ec2/.

[10] W Ahmad et. al. A survey on virtual machine migration and server consolidation frameworks for cloud data centers. *Journal of Network and Computer Applications*, 52(0):11 – 25, 2015.

[11] Guilherme Galante et al. A survey on cloud computing elasticity. UCC 2012, pages 263–270. IEEE Computer Society, 2012.

[12] A. Gember, R. Grandl, A. Anand, T. Benson, and A. Akella. Stratos: Virtual middleboxes as first-class entities. *UW-Madison TR1771*, 2012.

[13] A. Gember-Jacobson et al. Opennf: Enabling innovation in network function control. In *ACM SIGCOMM, 2014*.

[14] A. V. Goldberg and R. E. Tarjan. Finding minimum-cost circulations by canceling negative cycles. *JACM*, 36(4):873–886, 1989.

[15] Z. Gong, X. Gu, and J. Wilkes. Press: Predictive elastic resource scaling for cloud systems. In *IEEE CNSM, 2010*.

[16] F. Hermenier, X. Lorca, J. M. Menaud, G. Muller, and J. Lawall. Entropy: A consolidation manager for clusters. In *ACM SIGPLAN/SIGOPS*.

[17] M. A. Lopez and O. Duarte. Providing elasticity to intrusion detection systems in virtualized software defined networks.

[18] V. Mann, A. Kumar, P. Dutta, and S. Kalyanaraman. Vmflow: Leveraging vm mobility to reduce network power costs in data centers. In *IFIP NETWORKING, 2011*.

[19] X. Meng, V. Pappas, and L. Zhang. Improving the scalability of data center networks with traffic-aware virtual machine placement. In *IEEE INFOCOM, 2010*.

[20] M. Sedaghat, F. Hernandez-Rodriguez, and E. Elmroth. A virtual machine re-packing approach to the horizontal vs. vertical elasticity trade-off for cloud autoscaling. In *ACM CAC, 2013*.

[21] U. Sharma, P. Shenoy, S. Sahu, and A. Shaikh. A cost-aware elasticity provisioning system for the cloud. In *IEEE ICDCS 2011*.

[22] Z. Shen, S. Subbiah, X. Gu, and J. Wilkes. Cloudscale: Elastic resource scaling for multi-tenant cloud systems. In *ACM SoCC, 2011*.

[23] A. Verma, P. Ahuja, and A. Neogi. pmapper: Power and migration cost aware application placement in virtualized systems. In *ACM/IFIP/USENIX Middleware, 2008*.