Provably Efficient Algorithms for VNF Routing Optimization

Shike Zhang*, Yuxiang Liu*, Xiaofeng Gao‡, Jiaqi Zheng†, Guihai Chen*
*‡Shanghai Key Laboratory of Scalable Computing and Systems,
Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China
† State Key Laboratory for Novel Software Technology, Nanjing University, China
Email: {zsk9020, liu-yuxiang}@sjtu.edu.cn, {gao-xf, gchen}@cs.sjtu.edu.cn, jzheng@nju.edu.cn

***Abstract*—Previous researches on (Virtualized Network Function)VNF deployment mainly focus on resource and VNF distribution given several candidate paths, whereas routing, though more practical, is not under consideration. Especially, resource distribution does not remain optimal given continuously coming flows with different characteristics. No quantitative approach is proposed to determine the timing to redistribute resources and functions while real-time adjusting network distribution is too expensive. In this paper, we claim an alternative approach to handle the resource-and-flow matching problem using the method of routing. We find a routing and evaluate its costs using our VNFs Routing Evaluation Model. We prove the NP-hardness of VNF-RE. While previous works focus on heuristic algorithms, whose performance cannot be proven, we propose a two phase approach and prove the approximation ratio. The problem is formulated as an optimization problem which jointly considers VNF covering, multi-resource consumption, and routing overhead. We prove that the problem is NP-hard and our solution $O((1+(kD_{fmax}/D-1)\beta))\log^3 N \log m)$ approximates the optimal where k, D_{fmax} and β are variables that depend on the input flow. The simulation compares our work with several representative heuristic algorithms and shows that VNF-RE has better performance.**

***Index Terms*—Network Function Virtualization, Virtualized Network Function, routing, Group Steiner Tree**

I. Introduction

Network Function Virtualization (NFV) is a recent trend of splitting hardware equipments into software pieces that serve the same functions as their hardware counterpart. Those hardware, known as middleboxes, play critical roles in the network. NFV helps to increase network scalability and flexibility. The scale of a specific function can be dynamically increased if network congestion happens [1]; it is possible to pursue network innovation by deploying new functions; the network functions can be greatly adjusted to ensure better overall performance. The popularity of NFV is achieved by the development of cloud computing, industry standard high volume servers and Software Defined Network(SDN) [2]. Cloud infrastructure enables proper orchestration mechanisms which can be applied to NFV implementation. Also, industry standard high volume servers satisfy the basic computational needs and serve for more general purpose than middleboxes. SDN is somehow complementary to NFV. Although they do not rely on each other, SDN turns out to be beneficial and efficient to NFV network management.

We focus on a scenario where several Network Functions (NFs) are on network servers and data flows are required to be processed by a subset of those NFs considering request priority and efficiency issue. Flows are unsplittable and must walk along a single path. NFs are mostly implemented for the reason of security and analytics [3]. For example, Intrusion Detection System (IDS) [4], [5], Firewalls, Intrusion Prevention System (IPS) [6], Deep Packet Inspection (DPI) are all duty NFs before flow reaches its destination under security consideration. In a general network, a working node has resources of multiple types such as CPU, memory and disks. Several VNF instances run on the working node and consume resources while they are processing packets. If necessary, the VNF instances are swapped in the memory, which induces overhead especially when the swap in/out operation frequently occurs.

Previous works [7]–[10] focus on VNF deployment given a set of predefined paths. The optimal placement of resource and VNFs is dependent on flow features such as sources, demand of flow and destinations. Flow characteristics have great impact on overall network performance. Real time NFs and resource deployment is expensive. It is argued by [1] that network requests are very similar in each time slot, and therefore VNFs implementation does not have to change a lot. Current solution divides a period of time T into several time slots with similar length. Therefore, the performance inside each time slot t_i cannot be guaranteed. Figure 1 clarify the situation. Consider a timeline with T/t time slots each with t duration. Red node represents new requests in the middle of time slot, green node represents new requests at the beginning of time slot and blue node represents a delayed request from last time slot. In each time slot the requests arrived randomly, which approaches the real world network condition. Green requests can be arranged properly because at the beginning of each time slot, the resource distribution algorithm will make adjustment and achieve a near-optimal performance for several flows. The requests are not processed in real time but are collected and released after a fixed, short period of time. By assuming a similar flow pattern within the time slot, the algorithm achieves the expected performance. This mechanism, however, requires pre-knowledge of flow pattern, so is fragile to peak hour with abrupt flow rate increase that

‡ Corresponding Author

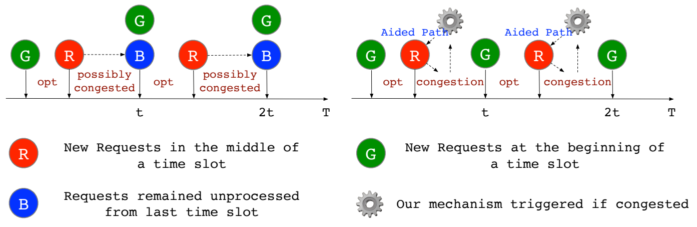
G
R
B
G
R
B
G
opt
possibly congested
opt
possibly congested
t
2t
T
Aided Path
Aided Path
G
R
G
R
G
opt
congestion
opt
congestion
t
2t
T
R New Requests in the middle of a time slot
G New Requests at the beginning of a time slot
B Requests remained unprocessed from last time slot
Our mechanism triggered if congested

Fig. 1. A timeline of traditional VNF networks

is not always predictable.

Our work deals with potential problems within a time slot. We do not launch the resource redistribution mechanism but rather make an effort to fully explore the network potential by finding several candidate paths. Our approach avoids the redistribution overhead and acts as an alarm system that is triggered only if congestion happens. Comparing with our counterpart of redistributing NF, the plan is faster, more agile and less expensive with simple computational overhead versus a time delay of redistributing NFs caused by a large amount of spinning-in/out from/to memory. Potential use includes a more flexible combination of resource redistribution and routing but those topics are out of the scope of this paper.

In this paper, we emphasize the importance of routing in NFV network and propose a **VNF**s **R**outing **E**valuation model (VNF-RE). We would like to explore potentially better path by selecting those with minimum costs. The cost can be interpreted as the number of hops (unit cost), sum of delay time or the extra expenses traveling along the link. Note that even if we only have a single function on each server, which avoids the complexity of set covering, the problem is still non-trivial as shown in Section 1.

To the best of our knowledge, all the solutions in existing works [11]–[14] solve the VNF routing problem with heuristic algorithms such as greedy and tabu search, whose performance is not proved. We are the very first to propose an innovative approximation algorithm. Our contributions are as follows.

- We formulate the VNF routing problem as the VNF-RE model and prove that VNF-RE is NP-hard. We regard it as a combinatorial optimization problem, aiming to minimize the routing cost. In general, given k incoming flows and their required network functions, the algorithm determines a path that links the origin and destination and make sure that the packet is processed by all of demanding functions.
- Considering the complexity of VNF-RE problem, we propose an innovative two phase approach. In the first step we compute several feasible paths with bounded costs which $O(\log m \, \log\log N)$ approximates the path with optimal cost. m denotes number of functions and N denotes number of servers. In the second step we propose an effective greedy algorithm with approximation ratio $O(1 + (kD_{fmax}/D - 1)\beta)$ where D_{fmax}, D and β are related to the input requests.
- We simulate VNF-RE and compare the result with several proposed heuristic algorithms. The results show that our solution performs better than heuristics.

TABLE I
NOTATIONS

f_{ij}^k	flow k goes through edge e_{ij} or not
d_f^k	demand of flow k
c_{ij}	cost function of edge e_{ij}
B_{ij}	bandwidth of edge e_{ij}
R_{it}	amount of resource t on server i
r_{tm}	cost resource t per demand of flow corresponding to function m
y_{im}^k	flow k's required function m is processed by server i or not
u_i^k	a marker of visiting sequence of server i for flow k
f_p^k	flow k goes through path p or not
c_p	cost of path p
F_k	required functions of flow k
P_e^k	path sets of flow k that traverse through edge e
P_i^k	path sets of flow k that traverse through node i
N	number of servers
m	number of functions
I	request set
S	server set
R	resource set
Y	assignment set

II. Related Work

Various works focus on solutions for waypoint routing where flows are required to pass several NFs. What's more, a group of researches jointly optimizes both placement and routing in terms of different objective.

VNFs are regarded as a set of waypoints in [15]. It provides an approximation ratio for bidirectional networks by formulating the problem to be a TSP equivalence. But it fails to consider waypoints with several instances and network resource constraints. [11] offers solution for multicast where both placement and routing are considered. The formulation preserves order of NFs. In [16] the authors aim to jointly minimize user utility and ISP profit in NFV network. The two criteria are optimized using Pareto efficiency. [17] recognizes the VNF placement and chaining to be NP-hard and use an extension of Monte Carlo Tree Search method to address the problem. [18] provides NP analysis for waypoint routing with specific settings such as directed graph, Euler graph and changed flow size. It is stated that when flow size changes, the problem becomes strongly hard. [12] initiates the study of waypoint routing and proves that exact polynomial-time algorithm exists for trees with bounded depth while others remain NP-hard.

[19] explores NFV network routing model suitable for ISP operations. The authors formulate the model into a mixed integer linear program and provides a multi-objective math-heuristic resolution. [20] proposes an architecture to support IPV6 segment routing in a VNF network context [21] present a *steiner tree* based algorithm to reduce cost for both implementation and routing. The paper aims to minimize the bandwidth consumption. [22] proposes a proactive online

algorithm. The authors try to predict traffics and deploy NF instances before physical machines are overloaded. [23] proposes solution for multicast NFV placement and implementation and provides a two-approximation ratio. Further, the authors present a heuristic algorithm which shows less complexity. [24] uses genetic algorithms on five general VNF forward graph design and NF placement in hybrid network environment.

III. The VNF-RE Model

A. Model Statement

We consider the cluster on which m NFs are deployed. The NFs are described by a set $\{1, 2, ..., m\}$. For the sake of reliability, efficiency and flexibility, a single NF has multiple backups on different servers and each server holds at least one function. Each server has a limit amount of resources. There are r resources in each server. The amount of resource t in server i is denoted R_{it} and the set of resource is denoted $[R]$.

Several flows traverse through the network. They are assigned to servers to be processed. Each flow f_j has a required function set Fj and a demand of flow df_j. It has to hit all the functions in Fj before leaving. We introduce a convex cost function c_{ij} to denote the cost of utilizing edge e_{ij}. The total cost represents the efficiency of VNF deployment. The higher is the cost function, the worse is the deployment scheme. We aims to minimize the total cost function. Notations are summarized in Table I.

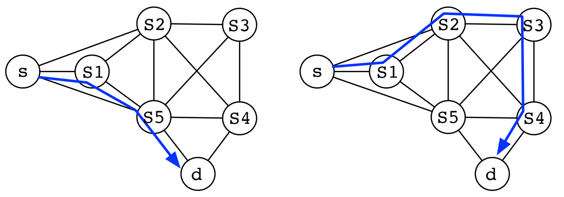

Fig. 2. Problem Illustration

Figure 2 illustrates how the flow satisfies the cover requirement. In figure 2's setting S_1 has the function $\{1, 2, 3\}$, S_2 has $\{1, 3\}$, S_3 has $\{4, 5\}$, S_4 has $\{2, 5\}$ and S_5 has $\{4, 5\}$.

Both of two images show feasible solutions to the problem. In the first image, the flow goes along the path $s - s_1 - s_5 - d$. Since $s_1 \cup s_5$ are together complete set, the flow path is a feasible one. The second image obeys a similar rule by going through $s - s_1 - s_2 - s_3 - s_4 - d$ and therefore is also feasible. If we assign unit cost to each of the edges on the graph, which can be interpreted as number of hops, the path on left-side image is a better solution with a cost of 3 while the other 5. As a result, in figure 2, a total cost of 3 is the optimal.

B. Problem Formulation

We formulate the problem considering the constraints of network links and server resources. Our optimization goal is to minimize total cost for all the flows. We assume that the order for NFs to be processed can be random.

$$\text{minimize} \quad \sum_{k\in[I]} \sum_{i\in[E]} \sum_{[j\in[E]} c_{ij} f^k_{ij} \tag{1}$$

$$\text{subject to} \quad \sum_{i\in[V]} f^k_{ij} - \sum_{j\in[V]} f^k_{ji} = src(j), \quad \forall k \in [I] \tag{1a}$$

$$\sum_{i\in[V]} y^k_{im} \geq 1, \quad \forall k \in [I], m \in [F_k] \tag{1b}$$

$$\sum_{k\in[I]} d^k_f f^k_{ij} \leq B_{ij}, \quad \forall i, j \in [E] \tag{1c}$$

$$\sum_{k\in[I]} \sum_{m\in[F_k]} d^k_f r_{tm} y^k_{im} \leq R_{it}, \quad \forall i \in [S], t \in [R] \tag{1d}$$

$$\sum_{j\in[V]} f^k_{ji} \geq y^k_{im}, \qquad \forall k \in [I], \forall i \in [V], \forall m \in [F_k] \tag{1e}$$

$$u^k_i - u^k_j + N f^k_{ij} \leq N - 1, \quad \forall i, j \in [S], \forall k \in [I] \tag{1f}$$

$$y^k_{im} \in \{0, 1\}, \forall i \in [S], \quad \forall k \in [I], \forall [m] \in [F_k] \tag{1g}$$

$$f^k_{ij} \in \{0, 1\}, \forall i, j \in [E], \quad \forall k \in [I] \tag{1h}$$

Constraint (1a) is a standard multi-commodity flow constraint that is applicable to all nodes, where $src^k(j)$ equals '1' if j is source of flow k, equals '-1' if it is a destination and equals '0' if it is neither of them. Constraint (1b) indicates that each function $i \in F_j$ has to be processed at least once. Constraint (1c) and constraint (1d) are the link capacity constraint and the server resource constraint respectively. Constraint (1e) indicates that a flow cannot be processed by functions on server S_i if it is not routed through S_i. Constraint (1f) is a cycle elimination equation.

An infeasible solution without constraint (1f) is indicated in figure 3. Both the two images show one unexpected flow that are not coming from the source s but still satisfies (1). The situation is likely to occur because of the covering requirement of (1b). In the first image shows the flow of size 1.0 that loops between S_2 and S_4. The loop aims to cover the function of {1,2,3} so that the flow starting from s can directly choose a minimum cost path. In the second graph it shows a fractional version of loop. There is a flow of size 0.8 unexpectedly looping between S_1 and S_5.

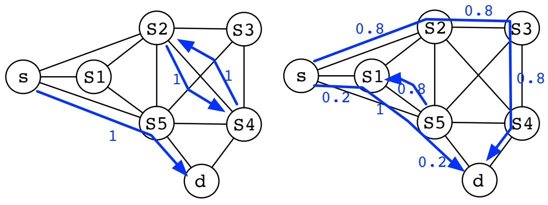

Fig. 3. The loop situation

C. Hardness Analysis

Theorem 1. *VNF-RE is $NP-hard$.*

Proof. We prove that VNF-RE problem is NP-hard by a reduction from the traveling salesman problem (TSP). The proof is trivial and is ignored here. □

IV. A Two-Phase Approach

To address the complexity and circumvent the dilemma of the problem, In this section we propose a two phase approach as a solution. Program (1) can be reformulated as following: we at first calculated a set of feasible paths for a flow satisfying constraints (1a)(1b)(1f)(1g)(1h). Next, we find a path-based solution of the problem. The first step of the two phase approach can be interpreted as pruning that narrows the search space of feasible path and reduce problem complexity.

However, there are potentially exponential number of feasible paths. In section IV-B we propose *the first step* to find a path set whose costs $O(\log^3 m \log\log N)$ approximate optimal path where m denotes the number of functions and N denotes the number of servers. In IV-A we propose *the second step*, in which we arrange the flow requests to the paths using a method of greedy by choosing the path with least cost.

A. VNF-RE problem Reformulation

In this section, we propose a path based version of (1). We denote $[P_e^k]$ to be a feasible path set of flow k that goes through edge e. Similarly, we denote $[P_i^k]$ to be that of flow k going through node i. Variable f_{ij}^k and c_{ij} are both changed to base on path rather than edges. $f_p^k \in \{0,1\}$ represents whether flow k goes through path p and c_p is the cost of path p. We have $c_p = \sum_{e\in p} c_e$.

$$\text{minimize} \quad \sum_{k\in[I]} c_p f_p^k \tag{2}$$

$$\text{subject to} \quad \sum_{k\in[I]} d_f^k \sum_{p\in[P_e^k]} f_p^k \le B_e, \quad \forall e \in [E] \tag{2a}$$

$$\sum_{k\in[I]} \sum_{m\in[F_k]} d_f^k r_{tm} y_{im}^k \le R_{it}, \quad \forall i \in [S] \quad \forall t \in [R] \tag{2b}$$

$$\sum_{p\in[P_i^k]} f_p^k \ge y_{im}^k, \qquad \forall k \in [I] \tag{2c}$$

$$\sum_{i\in[S]} y_{im}^k \ge 1, \qquad \forall m \in F_k \tag{2d}$$

$$\sum_{p\in[P_k^*]} f_p^k = 1, \qquad \forall k \in I \tag{2e}$$

$$y_{im}^k \in \{0,1\}, \qquad \forall i \in [S] \quad \forall m \in [I] \tag{2f}$$

$$f_p^k \in \{0,1\}, \qquad \forall p \in [P^*] \tag{2g}$$

B. l-Feasible Path Calculation

We propose an approximation algorithm to compute a set of at least l feasible paths. Given a graph, where the functions are fixed on node servers, we aim to compute l feasible paths that traverse through all the required functions corresponding to a single pair of source and destination. It is assumed that the flows have same required functions. Functions that are not required by flows can be simply ignored on the graph. We consider that l can be dynamically adjusted according to the flow demand in terms of performance.

Instead of calculating the paths starting from source node and heading to the destination, our method finds a tree that has at least one path going from root to each function. The formulation is based on the Group Steiner Tree (GST) problem. It is trivial that although we have to reach the destination finally, we can simply assign a distinct function (e.g. function 6 in figure 2, 3) to the destination node so that if we cover all the function in F, we reach the destination.

Theorem 2. *Let $c(T)$ be the cost of an optimal group steiner tree, cost of the path can be no more than $2c(T)$.*

Proof. The proof is similar to a TSP to minimum spanning tree formulation [25]. □

Theorem 3. *A single path derived from the group steiner tree algorithm has a cost no more than $O(\log^3 m \log\log N)$ times the optimal path.*

Proof. We ignore the process of group steiner tree algorithm because it has been fully discussed in [26]. □

Theorem 2 and 3 inspire the idea of finding several GST to preserve the approximation ratio. Two paths are considered to be different if at least one node or one edge are different. We find l different paths by discovering a node or an edge that can distinguish the path from existing path set. We enumerate nodes and edges to find the minimum cost paths. Alg. 1 indicates our idea.

GST is a random algorithm. In Alg. 1 We assume that if we use the random procedure for a bounded number of time, the algorithm returns at least l different paths. To accelerate the process and to avoid returning the same path all the time, Alg. 1 makes assumption for each node and edge (line 3-16) that they are in $p \in PathSet$ but not next path p^*. By deleting edges or nodes from G, we can still find p^* with a bounded cost. From theorem 4 we show that the i^{th} path is optimal and thus by induction Alg. 1 is valid.

Theorem 4. *(Validity of Alg. 1) If we find a new path p^* with Alg. 1 in the i^{th} iteration, cost(p^*) is at most the i^{th} expensive.*

Proof. We denote the i^{th} path chosen by Alg. 1 p_i. By contradiction, if $cost(p^*) \ge cost(p_i)$ and p_i is a new path different from p^*, it must has at least one node or edge that is different from any of the path in $PathSet$. Therefore in the iteration, p_i must have been evaluated but has a cost greater than p^*, which turns out to be a contradiction because $cost(p^*) \ge cost(p_i)$, the theorem is proved. □

Algorithm 1: LFP: l-Feasible Path Calculation

```
Input  : G(V,E), $l$, $s$, $d$
Output: $l$ feasible paths
1  Initialize: $MinPath \leftarrow \infty$, $PathSet \leftarrow \phi$, $count \leftarrow 0$.
2  for $count = 1$ to $l$ do
3      for Each Node $\in [V]$ do
4          $G_N = DeleteNode(G, PathSet)$;
5          $p = GST(G_N)$;
6          if $cost(p) \leq MinPath$ then
7              $p^* \leftarrow p$;
8              $MinPath \leftarrow cost(p)$;
9      for Each Edge $\in [E]$ do
10         $G_E = DeleteEdge(G, PathSet)$;
11         $p = GST(G_E)$;
12         if $cost(p) \leq MinPath$ then
13             $p^* \leftarrow p$;
14             $MinPath \leftarrow cost(p)$;
15     Add $p^*$ to $PathSet$;
16 return $PathSet$;
```

C. A Greedy Algorithm

Flows tend to move along the lowest cost path as long as it satisfies multiple resource constraints. The complexity lies in the fact that several paths may share same nodes or links, and thus share the resources on them. The limited resources should be scheduled among those paths.

We observe a greedy nature of the problem: it is profitable to route flows on the lowest cost path, and therefore the limited resource should be first assigned to the minimum cost path. There is a dilemma as how to determine whether the resources are adequate to hold a particular flow with a df demand of flow. This leads to our first algorithm - the Bottleneck based Greedy (BNG) algorithm. The BNG is provided in Alg.2. It takes the bottleneck d_f for each path as input by calling a bottleneck detection subroutine.

In Alg. 2 we first sort flow requests in terms of df and all the path in P_r in terms of costs c_p in ascending order (line 2-4) . To determine the path for a particular request, for each request we enumerate all the feasible path $p \in P_r$ (line 5-6) and check whether they are qualified or not. Since the path is already sorted, we find a first-fit path that has the smallest possible cost and does not violate resource constraints (line 7-10). If the algorithm found a path p that could not fit current flow requests, it will distribute all the resources on p to other paths and update their bottleneck d_f values (line 11-16).

a) A Bottleneck Detection Subroutine: To accomplish the idea of greedy algorithm, we firstly find the bottleneck of a single path, which is defined to be the maximum demand of flow that can traverse through the path. It is restricted by two limitations: link capacity and server resource capacity. By gradually increasing the demand of flow, we can finally reach a point that no more flow is able to be added.

$$\text{maximize} \quad D_f^p \tag{3}$$

$$\text{subject to} \quad D_f^p \leq B_e \quad \forall e \in [P_e] \tag{3a}$$

$$\sum_{m \in [F_k]} D_f^p r_{tm} y_{im} \leq R_{it}, \quad \forall i \in [S] \cap [p], \forall t \in [R] \tag{3b}$$

$$\sum_{i \in [S]} y_{im} \geq 1, \qquad \forall m \in F \tag{3c}$$

$$0 \leq y_{im} \leq 1, \qquad \forall i \in [S] \quad \forall m \in [I] \tag{3d}$$

constraint (3a) indicates link capacity requirement, constraint (3b) indicates resource capacity requirement and constraint (3c) follows a cover constraint. (3) includes a resource scheduling problem. The subroutine is used in line 34 of Alg.2.

b) CalRestResrc: specifies the procedure of updating bottleneck d_f for each path. Node resources are distributed to the minimum cost path that traverse through the node and link resources are distributed to the minimum cost path that traverse through the link. Resources are delivered from the lower-cost path to the higher-cost path who are direct predecessor and successor from an ordered perspective.

D. Theoretical Analysis

In this section we analyze the performance of our algorithm in terms of approximation ratio. For simplicity we introduce several notations. They are only used in this section.

- A denotes the number of requests.
- n_i denotes the number of flows arranged to p_i under Alg. 2 where $cost(p_1) \leq cost(p_2) \leq ... \leq cost(p_k)$
- k denotes the number of distinct paths used by Alg. 2.
- $T = \min_t\{\sum_{i=1}^{t} Df^p \geq D\}$ denotes a minimum number of paths that must be used by Alg. 2.
- $D = \sum_{k \in [I]} d_f^k$ denotes the sum of demand of flow.
- $cost(Alg.2)$ denotes the cost calculated by Alg. 2.
- $cost_{max}^p$ denotes maximum cost of path.
- $cost_{min}^p$ denotes minimum cost of path.

lemma 1. *If there are no remanent resource for each path except the last one, Alg.2 returns an optimal scheduling.*

Proof. Consider that there is no remanent resource for each path and that Alg.2 is greedy, it is trivial that

$$\sum_{i=1}^{p} n_i \geq \sum_{i=1}^{p} opt(n_i), \quad i \neq k, \forall p \leq k$$

The total cost for Alg.2 will be

$$\sum_{i}^{k-1} n_i cost(p_i) + (A - \sum_{i}^{k-1} n_i) cost(p_k)$$

opt however is the sum of costs for first $A - n_k$ requests and the last n_k requests. It is direct that the sum of the last n_k requests are at least $n_k cost(p_k)$ and thus

$$opt \geq \sum_{i}^{k-1} opt(n_i) cost(p_i) + (A - \sum_{i}^{k-1} opt(n_i)) cost(p_k)$$

Algorithm 2: BNG : the Bottleneck based Greedy Algorithm

```
Input  : A bottleneck demand of flow set $BN\{D_f^p\}$,
         request set $I$, path set $P_r$ for request $r$
Output: A set of arrangement tuple $\{request, path\}$ $[R^l]$
Initialize: $CurrentDf[path] \leftarrow \{0\}$
$I = SortAscent(I)$;
for request $r \in [I]$ do
    $P_r = SortAscentCost(P_r)$;
for request $r \in [I]$ do
    for request $p \in [P_r]$ do
        if $df_r + CurrentDf[p] \leq D_f^p$ then
            $CurrentDf[p] \leftarrow df_r + CurrentDf[p]$;
            $R^l[r].path \leftarrow p$;
            $break$;
        if $df_r + CurrentDf[p] > D_f^p$ then
            $[BN] \leftarrow CalRestResrc(p, P^*, R, B, Y)$;
            $P_r.delete(p)$;
            $P^*.delete(p)$;
            for request $r \in [I]$ do
                $P_r = SortAscentCost(P_r)$;
return $[R^l]$;

function CalRestResrc$(p, P^*, R, B, Y)$
Initialize: $curMin \leftarrow \infty$, $UpdatedSet \leftarrow \phi$
$I = SortAscent(I)$;
for each node $i \in p$ do
    for $p' \in [P_i]$ and $p' \neq p$ do
        if $cost(p') < curMin$ then
            $MinPath \leftarrow p'$
        for $R_{it} \in [R]$ do
            $R_{it} \leftarrow R_{it} - \sum_{m \in [F_p]} CurrentDf[p] r_{tm} y_{im}^p$;
        Add $p'$ to $UpdatedSet$;
for $p' \in [P_e]$ and $p' \neq p$ do
    if $cost(p') < curMin$ then
        $MinPath \leftarrow p'$
    $B_e \leftarrow B_e - D_f^p$;
    Add $p'$ to $UpdatedSet$;
for $p' \in [UpdatedSet]$ and $p' \neq p$ do
    Update $BN[p']$ with program (3);
return $[BN]$;
end function
```

subtract the lower bound of opt from $cost(Alg.2)$, together with $cost(p_i) < cost(p_k)$ yields

$$\sum_{i}^{k-1} cost(p_i)(n_i - opt(n_i)) - \sum_{i}^{k-1} cost(p_k)(n_i - opt(n_i)) \leq 0$$

Therefore the cost of Alg.2 is even less than opt and thus Alg.2 equals optimal. □

lemma 2. *(Lower Bound of opt) $opt \geq \sum_{i=1}^{T} cost(p_i)(n_i + 1)$*

Proof. Consider a situation that we reduce demand of flow of some requests. Although the path cannot hold even one more flow, we reduce some demand for it and put it in the path to make it full. The reduction satisfies the prerequisite of lemma 1. If we use Alg. 2 to calculate an assignment, we can retrieve a cost at least less than the optimal cost. Since that we have to prepare at least T boxes that hold all the requests, opt has at least the optimal cost, which is $\sum_{i=1}^{T} cost(p_i)(n_i + 1)$. □

Theorem 5. *The cost retrieved by Alg. 2 has an approximation ratio of $O(1 + (kD_{fmax}/D - 1)\beta)$ to (2), where D_{fmax} denotes the maximum demand of flow among all the paths and $\beta = cost_{max}^p / cost_{min}^p$.*

Proof. Consider a condition that T has to satisfy, which is the minimum number of paths to hold all the requests corresponding to their demand of flows. It is intuitive that from lemma 2 we have

$$T \geq D/D_{fmax}$$

Since in Alg.2 we use k paths to fulfill the requirements, we have

$$cost(Alg.2) = \sum_{i=1}^{k} cost(p_i) n_i$$

Considering lower bound on opt, by multiplying it by a factor of $(k/T - 1)\beta$, where $\beta = cost_{max}^p / cost_{min}^p$.

$$(k/T - 1)\beta opt \geq (k/T - 1)\beta \sum_{i=1}^{T} cost(p_i)(n_i + 1)$$
$$\geq \sum_{i=1}^{k} cost(p_i)(n_i + 1)$$

together with lemma 2, we have

$$cost(Alg.2) - opt \leq \sum_{i=T+1}^{k} cost(p_i)(n_i + 1) - \sum_{i=1}^{k} cost(p_i)$$
$$\leq \sum_{i=1}^{k} cost(p_i)(n_i + 1)$$
$$\leq (k/T - 1)\beta opt$$

To sum up,

$$cost(Alg.2) \leq opt + (k/T - 1)\beta opt$$
$$\leq (1 + (kD_{fmax}/D - 1)\beta) opt$$

And the ratio is proved. □

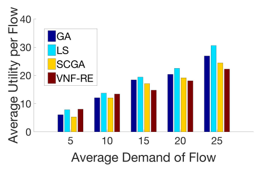

Fig. 4. CPU

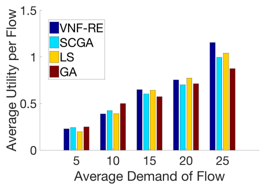

Fig. 5. Memory

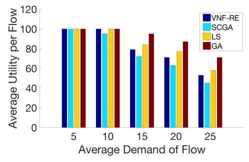

Fig. 6. Throughput

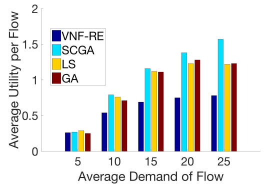

Fig. 7. Bandwidth

V. Performance Evaluation

A. Simulation Setup

a) Settings: We simulate an NFV cloud vendor that spans over a total time T of $6000s$. T is divided into several time slots Δt when at the beginning of each time slot, we redistribute NFs. The flow request input contain a source ip, a destination ip and a timestamp. In each time slot Δt, we consider those with different timestamps to be same single-path flows, but consider requests in different time slot to be different flows.

TABLE II
NFs resource consumption

VNF types / Parameters	FW	Proxy	NAT	IDS
CPU	20%	13.5%	2%	20%
Mem	1.9%	0.8%	10%	4.5%

Requests are required to be processed by a random subset of functions shown in table II. We assume that all the data packets have to be processed by all the NFs before leaving. The demand of flow is set randomly between $[0.1, 3]$ (Mbps). We randomly generate a cloud topology containing 40 nodes and 500 links to simulate the cloud situation [3] and their link bandwidth is set between 100 to 3000 based on an average number of flows. Only CPU and memory resources are considered. Servers are set to have the same configuration of $4GB$ memory. We assume that the coming number of flows follow a Gaussian distribution where $n \sim N(\mu, \sigma)$. μ equals 1000 under current setting.

b) Comparison Schemes: We compare VNF-RE with three heuristic algorithms: Greedy Algorithm (GA), Local Search (LS) and a Set Cover based Greedy Algorithm (SCGA). (i) GA: for each flow it finds a minimum cost path to reach a server holding a function that remains unprocessed. Meanwhile the algorithm should not violate link and node capacity constraints until the data request is processed by all the required functions. (ii) LS: find a minimal cost path. Enumerate all the required functions. If found a function that cannot be processed on the path, the algorithm take a detour and find a minimum cost circle, extending the path until all the functions are collected. (iii) SCGA: consider a set $|\Delta S_i|$ representing number of non-collected functions on server i. At any location the request flows through a path entering server with lowest average function costs, which is $\frac{c_e}{|\Delta S_i|}$. If on some locations $|\Delta S_i|$ equal zero, SCA randomly chooses a path to go. In (i)-(iii), we arrange flow requests following a smaller first strategy. All the algorithms consider the same graph and input flow requests.

B. Performance Analysis

a) Load balancing in the network: Load balancing among multiple resources indicates the potential and effectiveness of a network. We evaluate the CPU, memory and network link bandwidth utility corresponding to each algorithm.

Figure 4, 5, 7 depict the average CPU utility per flow, average memory utility per flow and average bandwidth utility respectively. We observe that SCGA has the smallest average CPU utility among the three when average demand of flow is relatively low, but VNF-RE outperforms others when average df increases when it maintains a low utility. In our setting CPU resource is the bottleneck resource and therefore CPU utility demonstrates the ability to adapt to peak hour flows.

b) Throughput: Throughput is under our concern because VNF-RE focus on increasing throughput by finding aided paths. High throughput indicates an ability of finding potential feasible paths. The three comparison schemes find paths by searching while VNF-RE obtains path set using GST randomize algorithm. Figure 6 indicates the throughput of all the schemes when the total number of requests is 100. We observe that when the average df increases, the throughput decreases. VNF-RE performs the best on finding paths because it calculates path set from an overall perspective. Besides, SCGA also has great ability of increasing throughputs while its time consumption is less than the other three.

c) Total cost for all the flows: Total cost is our optimization goal. In Figure 8 We evaluate the average cost for each scheme. We observe that LS performs well with small average df but it reacts badly when the df increases. On the other hand, SCGA has better performance with large df but it has no advantage over other schemes when df is low. GA performs the worst among the three. VNF-RE however, is always superior to the other schemes. Therefore we draw a conclusion that VNF-RE well adapts to traffic variations.

In figure 9 and 10 we evaluate the cost increase pattern in ordinary hour and peak hour respectively. We observe that VNF-RE is rather smooth because of the greedy nature while that of GA, LS and SCGA is rugged because they search and cannot guarantee that all the resources are assigned to the lowest cost path first. In figure 10 we consider a peak hour

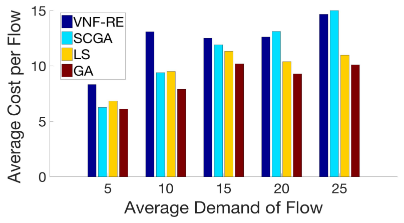

Fig. 8. Total Link Cost

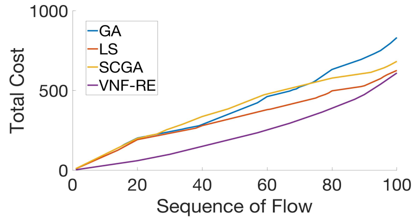

Fig. 9. Ordinary Cost

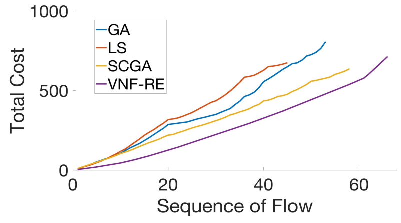

Fig. 10. Peak Hour Cost

situation. During the peak hour VNF-RE still has advantage over the heuristics.

VI. Conclusions

In this paper, we emphasize the importance of fully exploring potential in NFV network by developing algorithms for routing. We formulate the NFV routing problem and present our VNF-RE model. Previous works mostly focus on heuristics while we propose an innovative approximation algorithm to solve it. We prove that the problem is NP-hard and address the complexity of it with a two phase approach. In the first step we find several candidate paths using a Group Steiner Tree (GST) method with $O(\log^3 m \log\log N)$ approximation ratio. In the second step we calculate a minimum cost assignment using the approach of greedy by assigning requests with lower demand of flow at first. We prove the approximation ratio to be $O((1+(kD_{fmax}/D-1)\beta))\log^3 m \log logN)$. Finally, we conduct experiments comparing VNF-RE with several representative heuristics. We show that VNF-RE has better performance because it has higher throughput, better resource utility and lower average total cost per flow.

Acknowledgement

This work is supported by the China 973 project (2014CB340303), the National Natural Science Foundation of China (61872238, 61472252, 61672353, 61802172), the Shanghai Science and Technology Fund (17510740200), the CCF-Tencent Open Research Fund (RAGR20170114), and Huawei Innovation Research Program (HO2018085286)

References

[1] T. Wang, H. Xu, and F. Liu, "Multi-resource load balancing for virtual network functions," in *Inproceedings of the IEEE International Conference on Distributed Computing Systems*, 2017, pp. 1322–1332.
[2] M. C. et.al, "Network functions virtualisation," in *SDN and OpenFlow World Congress*, 2012.
[3] Y. Sang, B. Ji, G. R. Gupta, X. Du, and L. Ye, "Provably efficient algorithms for joint placement and allocation of virtual network functions," in *Inproceedings of the IEEE INFOCOMM*, 2017.
[4] A. Olteanu, Y. Xiao, K. Wu, and X. Du, "An optimal sensor network for intrusion detection," in *Inproceedings of the IEEE International Conference on Communications*, 2009.
[5] J. Lv1, W. Yang, L. Gong, D. Man, and X. Du, "Robust wlan-based indoor fine-grained intrusion detection," in *Inproceedings of GLOBECOM*, 2016.
[6] R. Cohen, L. Lewin-Eytan, J. S. Naor, and D. Raz, "Near optimal placement of virtual network functions," in *Inproceedings of the IEEE INFOCOMM*, 2015.
[7] H. Feng, J. Llorca, M. Antonia, D. Raz, and A. F. Molisch, "Approximation algorithms for the nfv service distribution problem," in *Inproceedings of the IEEE INFOCOMM*, 2017.
[8] C. Ghribi, M. Mechtri, and D. Zeghlache, "A dynamic programming algorithm for joint vnf placement and chaining," in *CoNEXT*, 2016.
[9] M. Ghaznavi, A. Khan, N. Shahriar, K. Alsubhi, R. Ahmed, R. Boutaba, and D. R. Cheriton, "Elastic virtual network function placement," in *CLOUDNET*, 2015.
[10] F. Carpio, S. Dhahri, and A. Jukan, "Vnf placement with replication for load balancing in nfv networks," in *arXiv preprint*, vol. 1610.08266, 2017.
[11] V. Eramo, A. Tosti, and E. Miucci, "Server resource dimensioning and routing of service function chain in nfv network architectures," in *Journal of Electrical and Computer Engineering*, 2016.
[12] S. A. Amiri, K.-T. Foerster, and S. Schmid, "Walking through waypoints," in *arXiv:1708.09827 [cs.DS]*, 2017.
[13] S. Balasubramaniam, D. Botvich, R. Carroll, J. Mineraud, W. Donnelly, T. Nakano, and T. Suda, "Adaptive dynamic routing supporting service management for future internet," in *Inproceedings of the IEEE Global Telecommunications Conference*, 2009.
[14] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. D. Turck, and S. Davy, "Design and evaluation of algorithms for mapping and scheduling of virtual network functions," in *Inproceedings of the IEEE Conferenct on Network Softwarization*, 2015.
[15] K.-T. Foerster, M. Parham, and S. Schmid, "A walk in the clouds: Routing through vnfs on bidirected networks," in *ALGOCLOUD*, 2017.
[16] ChaoBu, XingweiWang, HuiCheng, MinHuang, KeqinLi, and S. K.Das, "Enabling adaptive routing service customization via the integration of sdn and nfv," in *Journal of Network and Computer Applications*, 2017.
[17] O. Soualah, M. Mechtri, C. Ghribi, and D. Zeghlache, "Energy efcient algorithm for vnf placement and chaining," in *International Symposium on Cluster, Cloud and Grid Computing*, 2017.
[18] S. A. Amiri, K.-T. Foerster, R. Jacob, and S. Schmid, "Charting the complexity landscape of waypoint routing," in *arXiv:1705.00055 [cs.NI]*, 2017.
[19] B. Addis, D. Belabed, M. Bouet, and S. Secci, "Virtual network functions placement and routing optimization," in *Inproceedings of the IEEE International Conference on Cloud Networking (CloudNet)*, 2015.
[20] A.Abdelsalam, F. Clad, C. Filsfils, S. Salsano, G. Siracusano, and L. Veltri, "Implementation of virtual network function chaining through segment routing in a linux-based nfv infrastructure," in *Inproceedings of the IEEE Conference on Network Softwarization (NetSoft)*, 2017.
[21] Y. Cheng and L. Yang, "Vnf deployment and routing for nfv-enabled multicast: A steiner tree-based approach," in *Inproceedings of the IEEE International Conference on Wireless Communications and Signal Processing (WCSP)*, 2017.
[22] X. Fei, F. Liu, H. Xu, and H. Jin, "Adaptive vnf scaling and flow routing with proactive demand prediction," in *Inproceedings of the IEEE INFOCOMM*, 2018.
[23] S. Q. Zhang, A. Tizghadam, B. Park, H. Bannazadeh, and A. Leon-Garcia, "Joint nfv placement and routing for multicast service on sdn," in *Inproceedings of the IEEE Network Operations and Management Symposium*, 2016.
[24] J. CAO, Y. ZHANG, W. AN, X. CHEN, J. SUN, and Y. HAN, "Vnf-fg design and vnf placement for 5g mobile networks," in *SCIENCE CHINA*, 2017.
[25] V. V. Vazirani, *Approximation Algorithms*. London Milan Paris: Springer, 2001.
[26] N. Garg, G. Konjevod, and R. Ravit, "A polylogarithmic approximation algorithm for the group steiner tree problem." in *SODA*, 1998.