Provably Efficient Algorithms for Joint Placement and Allocation of Virtual Network Functions

Yu Sang, Bo Ji, Gagan R. Gupta, Xiaojiang Du, and Lin Ye

***Abstract*—Network Function Virtualization (NFV) has the potential to significantly reduce the capital and operating expenses, shorten product release cycle, and improve service agility. In this paper, we focus on minimizing the total number of Virtual Network Function (VNF) instances to provide a specific service (possibly at different locations) to all the flows in a network. Certain network security and analytics applications may allow fractional processing of a flow at different nodes (corresponding to datacenters), giving an opportunity for greater optimization of resources. Through a reduction from the set cover problem, we show that this problem is NP-hard and cannot even be approximated within a factor of $(1 - o(1)) \ln m$ (where m is the number of flows) unless P=NP. Then, we design two simple greedy algorithms and prove that they achieve an approximation ratio of $(1 - o(1)) \ln m + 2$, which is asymptotically optimal. For special cases where each node hosts multiple VNF instances (which is typically true in practice), we also show that our greedy algorithms have a constant approximation ratio. Further, for tree topologies we develop an optimal greedy algorithm by exploiting the inherent topological structure. Finally, we conduct extensive numerical experiments to evaluate the performance of our proposed algorithms in various scenarios.**

I. Introduction

Network Function Virtualization (NFV) is emerging as a promising technology in the evolution of networking [1] to replace proprietary hardware appliances (e.g., middleboxes) with software modules running on general-purpose commodity servers. These modules provide one or multiple specific network services (such as Firewalls, WAN Optimizers, Load Balancers, and Network Address Translators) called Virtual Network Functions (VNFs). These services can be placed along the path of a network flow in a specific order (i.e., Service Function Chaining) [2], potentially in a dynamic manner. Software-Defined Networking (SDN) [3] is usually integrated with NFV to enable the centralized control as SDN is aimed to separate the control plane from the physical infrastructure. This results in a highly flexible architecture and has the potential to significantly reduce the capital and operating expenses, shorten product release cycle, and improve service agility. In this paper, we focus on the problem of optimal placement and allocation of VNF instances to provide a specific service to all the flows in the network.

Yu Sang, Bo Ji, and Xiaojiang Du are with the Department of Computer and Information Sciences, Temple University, Philadelphia, PA, and Lin Ye is with Harbin Institute of Technology, China. Emails: yu.sang@temple.edu, boji@temple.edu, gagan.gupta@iitdalumni.com, dxj@ieee.org, and hityelin@hit.edu.cn. This work was supported in part by the US NSF under grant CNS-1564128.

We focus on the scenario of one single network function that requires all the data packets of the flows to be processed before they leave the network. This is common for many network services related to security and analytics, such as Intrusion Detection Systems (IDSs) [4] [5], Intrusion Prevention Systems (IPSs) [6], Deep Packet Inspection (DPI)[1], and network analytics/billing services. Each VNF instance is implemented at a virtual machine with limited resources and processing capacity. A network node (corresponding to a datacenter) can dynamically grow or shrink its capacity by spinning up or spinning down VNF instances. While existing work commonly assumes that a flow is completely processed at a single node for one function (e.g., [7]), in our model we consider a more general setting where one flow may be fractionally processed at a network node and the network function can be completed at multiple nodes. This model is based on widely adopted technologies. For example, in the Enhanced Packet Core (EPC) of mobile networks, packets are commonly encapsulated in GTP tunnels. Packets in the same GTP tunnel can be fractionally processed at different network nodes (locations) based on the IPs of the GTP payload. Another way to do fractional processing is by computing hash functions on fields in the packet headers.

To the best of our knowledge, existing work on VNF placement is limited to the design of heuristic algorithms, and none of the proposed algorithms can provide provable performance guarantees. In [2], a scheduling algorithm is proposed, but this work assumes a special fat-tree topology and focuses on delay performance. In [8], an algorithm based on dynamic programming is proposed to attack large instances of VNF placement problem. In [7], the authors consider a model with a single type of VNF and present a heuristic algorithm towards solving the placement problem. In [9], the authors propose a new architecture, called Stratos, for orchestrating VNFs outsourced to a remote cloud through traffic engineering, horizontal scaling of VNFs, etc. In another open source project [10], OpenNF is proposed to extend the centralized SDN paradigm by integrating a control plane for VNFs. There are several other works that extend the VNF placement problem to more sophisticated applications, such as Service Function Chaining [11]–[13]. However, the main focus of [12] and [13] is on latency and physical resource usage, respectively. For [11], although a similar objective is considered, no performance guarantee is provided for their

[1] In some cases, it is not required to process all the packets of a flow. For example, some DPI functions only check a small percentage of packets of a flow. In our model, we only consider the fraction that needs to be processed.

978-1-5090-5336-0/17/$31.00 ©2017 IEEE

solution. One exception that provides provable performance guarantees is the work of [6]. However, network topology is not considered in their model.

We consider the problem of joint placement and allocation of VNFs (denoted by JPA-VNF) with an objective of minimizing the total number of VNF instances. Note that even under a simplifying assumption of one single network function, the formulated problem is non-trivial (see Section III). The main difficulty of this problem is to decide intelligently which flows, and more specifically, what fraction of the flows need to be processed at a given node so that the computing resource of the placed VNF instances is not left under-utilized.

We formulate JPA-VNF as a Mixed Integer Linear Programming (MILP) problem and prove its NP-hardness through a reduction from the set cover problem. Using a similar reduction, we also show that JPA-VNF cannot even be approximated within a factor of $(1 - o(1)) \ln m$ (where m is the number of flows) unless P=NP. Then, we design two simple greedy algorithms and rigorously prove that they can achieve an approximation ratio of $(1 - o(1)) \ln m + 2$, which is thus asymptotically optimal. In many NFV-based applications of practical interest, such as those in cellular networks, the flow rates are usually very large, and it typically requires multiple VNF instances at a single node (datacenter) to process the flows. In such scenarios, we can even prove a constant approximation ratio for our proposed greedy algorithms. Furthermore, for networks with tree topologies, we design an optimal greedy algorithm by exploiting the inherent topological structure. Finally, we conduct numerical experiments both on a randomly generated dense graph and on a realistic backbone network topology of InternetMCI [14]. We evaluate the performance of our proposed algorithms in various scenarios. The simulation results show that our proposed algorithms perform very well in all the scenarios we consider.

The remainder of the paper is organized as follows. In Section II, we describe our system model and formulate the JPA-VNF problem. In Section III, we prove the hardness of the formulated JPA-VNF problem, and in Section IV, we propose two simple greedy algorithms and prove that they are asymptotically optimal. Then, we consider tree topologies in Section V and propose an optimal algorithm. Finally, we conduct simulations in Section VI and make concluding remarks in Section VII.

II. System Model and Problem Formulation

We consider a network that can be modeled as a connected, undirected graph $G = (V, E)$, where V is the set of nodes and E is the set of edges. Let $n = |V|$ denote the number of nodes. Each node represents a possible location for a VNF instance, which could be a cluster or a private datacenter owned by a certain network operator or a service provider. Each edge denotes the link between two such locations. We assume that there are m data flows in the network. Let F denote the set of flows. Each data flow enters the network at a source node, traverses a sequence of links/nodes, and leaves the network after reaching the destination node. Let d_j and P_j be the flow rate and path of flow $f_j \in F$, respectively. We assume that all the flow paths are loop free. In particular, P_j is denoted by a sequence of distinct nodes $\{v_1^j, v_2^j, \ldots, v_{|P_j|}^j\}$ that are connected by a sequence of links. Let $L_i = \{f_1^i, f_2^i, \ldots, f_{|L_i|}^i\}$ denote the set of flows that pass node v_i. See Fig. 1 for an example network.

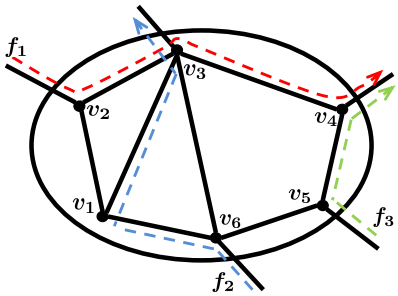

Fig. 1: An example network with six nodes and three flows.

While in the traditional networking paradigm, network functions are implemented on proprietary hardware and can only be placed at fixed locations, in a virtualized network environment VNF instances can be placed at any node as needed. A VNF instance is usually implemented on a standard virtual machine with limited amount of resource. However, a node can activate multiple VNF instances to increase its total processing capacity. In this paper, we focus on the scenario of single network function that requires all the data packets of a flow to be processed before they leave the network. We assume that one VNF instance has R units of computing resource. For ease of presentation, we assume that processing one unit of data requires one unit of computing resource. Let x_i be the number of VNF instances placed at node v_i. Then, the total processing capacity of node v_i is $x_i R$. The resource of one node could be shared by multiple flows that pass this node. Let r_{ij} denote the amount of computing resource allocated to flow f_j at node v_i.

Our goal is to minimize the total number of VNF instances used in the network, subject to the constraint that all the data flows need to be fully processed before leaving the network. This is a joint problem of placement and allocation of VNF instances: not only do we need to decide how many VNF instances to place at each node, but also need to determine how to allocate the computing resource of each VNF instance to process the flows passing each node. We formulate JPA-VNF as the following MILP problem:

$$\min_{x_i} \sum_{i=1}^{n} x_i$$

$$\text{subject to} \sum_{i:v_i \in P_j} r_{ij} \geq d_j, \text{ for all } 1 \leq j \leq m, \tag{1}$$

$$\sum_{j=1}^{m} r_{ij} \leq x_i R, \text{ for all } 1 \leq i \leq n, \tag{2}$$

$$x_i \in \{0, 1, 2, \ldots\}.$$

We assume that a flow can be processed at multiple nodes along its path. Constraint (1) means that the total resource that flow f_j receives from all the nodes on its path should be no less than its rate d_j. Constraint (2) means that the total demand for node v_i cannot exceed its processing capacity.

III. Hardness of JPA-VNF

In this section, we show that JPA-VNF is NP-hard through a reduction from the set cover problem, which is a well-known NP-hard problem. Similarly, we show that JPA-VNF cannot be approximated within a factor of $(1 - o(1)) \ln m$ unless P=NP.

We start by introducing the classic set cover problem. Consider a set $U = \{e_1, e_2, \ldots, e_m\}$ of m elements and a collection $\Phi = \{u_1, u_2, \ldots, u_n\}$ of n subsets of U. The union of all the subsets in Φ equals U, i.e., $\cup_{i=1}^{n} u_i = U$. The objective is to find the minimum number of subsets in Φ such that their union equals U. We state the hardness of JPA-VNF in Theorem 1.

Theorem 1: JPA-VNF is NP-hard.

Proof: Given an arbitrary instance (U, Φ) of the set cover problem as described above, we construct an instance (G, F, R) of the JPA-VNF problem. We show that there is a feasible solution with k VNF instances for the JPA-VNF problem if and only if there exists a feasible solution of k subsets in Φ for the set cover problem.

First, we set R to be any positive value. Then, we construct a graph G with n nodes and a set F of m flows. For each subset $u_i \in \Phi$, we add one node v_i to G. For each element $e_j \in U$, we construct a flow f_j. The path P_j of flow f_j contains v_i if e_i is an element of u_j, i.e., $P_j = \{v_i | e_i \in u_j\}$. We create the links such that every flow can traverse the nodes in its path (e.g., making G a complete graph). We set the rate of every flow to R/m. This ensures that one VNF instance is sufficient to fully process all the flows. Then, we calculate the set of passing flows L_i for each node v_i. The objective of this JPA-VNF problem is to find the minimum number of nodes such that the union of their passing flows equals F. It is easy to see that this is equivalent to finding the minimum number of subsets in Φ whose union equals the universe set U. ■

We use a simple example to illustrate the above construction process. Consider a set cover problem with $U = \{1, 2, 3\}$ and $\Phi = \{u_1, u_2, \ldots, u_6\}$, where $u_1 = \{2\}$, $u_2 = \{1\}$, $u_3 = \{1, 2\}$, $u_4 = \{1, 3\}$, $u_5 = \{3\}$, and $u_6 = \{2\}$. Based on this set cover problem instance and the construction in the proof of Theorem 1, the corresponding flows would be the same as those shown in Fig. 1. One difference is that the corresponding network topology is a complete graph and the flow rates are all 10/3.

In Theorem 2, we state a stronger inapproximability result. The detailed proof is omitted since a reduction method similar to that in the proof of Theorem 1 can be applied.

Theorem 2: The JPA-VNF problem cannot be approximated within a factor of $(1 - o(1)) \ln m$ unless P=NP.

IV. Asymptotically Optimal Greedy Algorithms

In this section, we propose two greedy algorithms and show that they can achieve an approximation ratio of $(1 - o(1)) \ln m + 2$, which is thus asymptotically optimal due to the result of Theorem 2.

A. Intuitions

For the set cover problem, a greedy algorithm is known to attain the best possible approximation ratio that a polynomial-time algorithm can achieve. It chooses a subset u_i^* with the largest number of uncovered elements in an iterative manner until all the elements in Φ are covered. Inspired by this greedy algorithm for the set cover problem, we develop two greedy algorithms for JPA-VNF. As shown in Theorem 1, every subset u_i in a set cover problem corresponds to a node v_i in a JPA-VNF problem. An intuitive approach is that we treat each flow as an element in the set cover problem and do not consider the flow rates. This leads to our first algorithm - the Flow Number based Greedy (*FNG*) algorithm. The FNG algorithm iteratively chooses a node with the largest number of unprocessed flows passing it. Another similar greedy strategy is to choose a node that has the largest amount of unprocessed data in each iteration. Based on this intuition, we propose our second greedy algorithm - the Flow Rate based Greedy (*FRG*) algorithm. At first glance, FRG seems to work better since it uses additional information of flow rates. However, in our technical report [15], we provide two examples to show that neither of them dominates. More interestingly, we prove that both greedy algorithms are asymptotically optimal.

Note that two obvious factors distinguish the JPA-VNF problem from the set cover problem. In the set cover problem, an element is covered as long as it is included in one of the chosen subsets. However, in our problem it matters which nodes are used to process a flow. If a flow is fully processed at a node, then there is no need to allocate computing resource to this flow at the other nodes along its path. This resource allocation problem also leads to the second difference. In the set cover problem, each subset has two states: either selected or not selected. However, in JPA-VNF a node could have multiple VNF instances, which leads to a much larger state space. Considering these two factors, when a node is selected, we choose to process all the flows that pass this node. We design the algorithms in such a manner instead of splitting the flow rates among multiple nodes due to the following reason. As the graph becomes larger and the flows interact with each other in a more complex way, the number of possible combinations increases exponentially. Our strategy is appealing because it leads to low-complexity algorithms with only minimal drop in the performance. In addition, our proposed algorithms tend to place VNF instances at as few nodes as possible, which is preferred from the application point of view.

Algorithm 1 $FNG(G, F, R)$

Let U be the set of all unprocessed flows. Initially, set $U = F = \{f_1, f_2, \ldots, f_m\}$.
Let S_i denote the set of unprocessed flows that pass node v_i. Initially, set $S_i = L_i$ for all i.
while U $\neq \emptyset$ **do**
    Find v_{i^*} such that $i^* \in \text{argmax}_i |S_i|$. Choose the node with the smallest index i when there is a tie.
    Process all the flows in S_{i^*} at node v_{i^*}, i.e., place $x_{i^*} = \left\lceil \frac{\sum_{j:f_j \in S_{i^*}} d_j}{R} \right\rceil$ VNF instances at node v_{i^*}.
    Allocate the computing resource to these unprocessed flows according to their flow rates, i.e., $r_{i^*j} = d_j$ for all j such that $f_j \in S_{i^*}$.
    Set $U = U \backslash S_{i^*}$.
    Set $S_i = S_i \backslash S_{i^*}$ for all i.
end while

B. Flow Number based Greedy Algorithm

We first introduce the FNG algorithm. FNG iteratively chooses a node that covers the largest number of unprocessed flows and places just sufficient VNF instances at this node such that all the flows passing this node can be fully processed. We describe the details of FNG in Algorithm 1.

Note that VNF instances are usually deployed on virtual machines with a limited amount of computing resource. In many applications like cellular networks, the flow rates can be very large, whereas there are only a limited number of datacenters in the network. A datacenter may need a large number of VNF instances to provide certain network function or service. Therefore, the number of VNF instances at each datacenter is typically large, which leads to an approximation ratio close to 1. We first state Lemma 1 based on this observation. Then, we will use it to prove the main result about FNG.

Lemma 1: Consider the FNG algorithm. Suppose a total of h VNF instances are placed at t different nodes. Let $A = h/t$ be the average density of the solution. Suppose $A \neq 1$. Then, FNG guarantees an approximation ratio smaller than $A/(A-1)$.

Proof: Let D be the total amount of data rates of all the flows, i.e., $D = \sum_{j=1}^{m} d_j$. Due to the way FNG functions, *each node has at most one VNF instance whose computing resource is not fully used.* Therefore, the total resource waste should be less than tR, i.e., $hR - D < tR$. Hence, we have

$$D > (h-t)R = \frac{A-1}{A} hR. \tag{3}$$

Now, we consider an optimal solution that uses a total number of O^* VNF instances. It must be satisfied that the total computing resource is no smaller than the total flow rate due to Constraint (1), i.e.,

$$O^* R \geq D. \tag{4}$$

By combining Eqs. (3) and (4), we derive $h < \frac{A}{A-1} O^*$. This implies that FNG guarantees an approximation ratio smaller than $A/(A-1)$. ■

Lemma 1 implies that if the average number of VNF instances at a node is greater than 1 (i.e., $A \geq 2$), the approximation ratio will be smaller than 2. Note that in practice, the density of a solution could easily be much larger. Consider a cellular network with 100 datacenter nodes and 10 million flows (users) with an average flow rate of 1 Mbps. If the processing capacity of one VNF instance is 1 Gbps, then it is guaranteed that there will be at least 10,000 VNF instances over 100 datacenters. Hence, the density will be at least 100, and thus, the solution computed by FNG is guaranteed to be within 1% of the optimal. Through Lemma 1, we can see that processing a flow entirely at a single node not only simplifies NFV orchestration but also ensures a minimal performance loss. $A = 1$ implies that there is exactly one VNF instance at every node that hosts VNF instances. In such cases, the JPA-VNF problem can be as difficult as the set cover problem.

The intuition behind Lemma 1 is as follows. When the algorithm gives a dense solution, in which all the VNF instances are placed at a small number of nodes, rather than sparsely spreaded over the entire network, the solution is typically very close to optimal. This is because resource waste at a single node cannot exceed R. Placing the VNF instances at fewer nodes will lead to less resource waste. We will prove the performance guarantee of FNG (stated in Theorem 3) based on the insight obtained from Lemma 1.

Theorem 3: The approximation ratio of FNG is no greater than $(1 - o(1)) \ln m + 2$.

Proof: We first divide the original JPA-VNF problem into two subproblems. Then, we use FNG algorithm to solve these two new problems and get two solutions, a dense one and a sparse one. We will prove that the combination of these two solutions is equivalent to the FNG solution to the original JPA-VNF problem. The dense solution achieves a constant approximation ratio as shown in Lemma 1, and for the sparse one, we prove that the approximation ratio is no greater than $(1 - o(1)) \ln m$.

Consider a JPA-VNF problem $I = (G, F, R)$. Assume that the total number of VNF instances used by the FNG algorithm and an optimal algorithm are H and O^*, respectively. We divide the flow set F into two subsets F_1 and F_2 and construct two new JPA-VNF problems. The partition is done is the following way. Recall that in every iteration of the FNG algorithm, we choose one node v_i and allocate VNF instances to process all the unprocessed flows that go through node v_i. If we only place one VNF instance at v_i, then all the flows processed by this VNF instance belong to F_1; otherwise, they belong to F_2. By doing this in each iteration, we construct flow sets F_1 and F_2. Note that under the FNG algorithm, every flow is completely processed at a single node. Therefore, F_1 and F_2 are disjoint, and their union equals F. We keep the original network topology and obtain two subproblems:

$I_1 = (G, F_1, R)$ and $I_2 = (G, F_2, R)$. We will compare the performance of FNG and the optimal algorithm for these two subproblems. Let H_1 and H_2 denote the number of VNF instances in the solutions given by FNG for these two subproblems, respectively, and let O_1^* and O_2^* denote the number of VNF instances in the optimal solutions, respectively. Since I_1 and I_2 have fewer flows than I, we have

$$O_1^* \leq O^* \quad \text{and} \quad O_2^* \leq O^*. \tag{5}$$

Now, we characterize the relation between the solutions of the new problems (i.e., I_1 and I_2) and the original problem (i.e., I) under FNG. We will first show the following:

$$H = H_1 + H_2. \tag{6}$$

Then, we will show that the following two inequalities hold:

$$H_1 \leq (1 - o(1))O_1^* \ln m, \tag{7}$$

$$H_2 \leq 2O_2^*. \tag{8}$$

Finally, using these intermediate results, we compare our greedy solution with the optimal solution.

We can prove Eq. (6) by induction. We provide the detailed proof in our online technical report [15] and briefly describe the idea here. Consider the solutions given by FNG for problem instances I, I_1, and I_2. Suppose that in the solution of I, node v_i hosts x_i VNF instances. Then, the following must be true: *in one of the two solutions of the subproblems, node v_i also hosts x_i VNF instances, and in the other one, node v_i does not host any VNF instance.* Apparently, this implies Eq. (6). Consider all the nodes chosen in I, we can show that the above claim holds for all the nodes one by one through induction. Hence, we have $H_1 = \sum_{i|x_i=1} x_i$ and $H_2 = \sum_{i|x_i \geq 2} x_i$. Therefore, we can get $H_1 + H_2 = \sum_i x_i = H$.

Subproblem $\mathbf{I_1}$

We now prove Eq. (7). We first construct another JPA-VNF problem based on I_1. Let $m_1 = |F_1|$. We change the rate of every flow in F_1 to a very small value. Let d_{min} be the lowest flow rate of all the flows in F_1. We set the rate of all the flows to $\min\{d_{min}, R/m_1\}$. This modification ensures that every node needs at most one VNF instance to process all the flows that pass the node under any feasible algorithm, and that the rate of a flow does not increase. We use F_3 to denote the set of flows with new flow rates. Then, we construct a new JPA-VNF problem instance $I_3 = (G, F_3, R)$. Now, we apply FNG to I_3. Assume that FNG uses H_3 VNF instances to process all the flows. FNG does not consider the rate of the flows, and every node requires only one VNF instance to process all the flows passing it. Therefore, the solution should be the same as that for I_1, i.e., $H_3 = H_1$. For this new instance, let O_3^* be the optimal solution. The only difference between I_1 and I_3 is that the later one has lower flow rates. Therefore, it is easy to see $O_3^* \leq O_1^*$. Note that in this case, I_3 can be exactly mapped to a set cover problem, and FNG also becomes equivalent to the well studied greedy algorithm for the set cover problem. Hence, we have

$$H_1 = H_3 \leq (1 - o(1))O_3^* \ln m_1 \leq (1 - o(1))O_1^* \ln m,$$

Algorithm 2 $GFT(T, F, R)$

1: **for** p from the largest to the smallest **do**
2: **for** $q = 1 \to l_p$ **do**
3: **if** there are flows leaving the network through $v_{p,q}$ **then**
4: put $\lceil d_{p,q}/R \rceil$ VNF instances at $v_{p,q}$ to process all the flows leaving the network through $v_{p,q}$.
5: **while** there is computing resource left **do**
6: Allocate the computing resource to process the first flow f_j in $F_{p,q}$.
7: Update the rate of f_j if it is not fully processed. Otherwise, move it out of $F_{p,q}$.
8: **end while**
9: Let F' be the set of flows that are fully processed in this loop (line 3 $\sim$ line 8). Update the waiting list and unprocessed flow set of all other nodes, $F_{s,k} = F_{s,k}/F'$, $D_{s,k} = D_{s,k}/F'$ for all s and k.
10: **end if**
11: **end for**
12: **end for**

where the first inequality is due to the achievable approximation ratio of the greedy algorithm for the classic set cover problem [16], and the second inequality is due to $O_3^* \leq O_1^*$ and $F_1 \subseteq F$.

Subproblem $\mathbf{I_2}$

Next, we prove Eq. (8). The proof follows immediately from Lemma 1. Recall that all the nodes in I_2 has either no VNF instance or at least two VNF instances. Assume that FNG places VNF instances at t_2 nodes in I_2. Each one of these t_2 nodes has at least two VNF instances. Hence, we have $H_2/t_2 \geq 2$. Therefore, we have $H_2 \leq 2O_2^*$ from Lemma 1.

Combining Eqs. (5), (6), (7), and (8), we have

$$\begin{aligned} H &= H_1 + H_2 \\ &\leq (1 - o(1))O_1^* \ln m + 2O_2^* \\ &\leq ((1 - o(1)) \ln m + 2)O^*. \end{aligned}$$

Therefore, the approximation ratio of FNG is upper bounded by $(1 - o(1)) \ln m + 2$. This, along with Theorem 2, implies that FNG is asymptotically optimal. ■

C. Flow Rate based Greedy Algorithm

We briefly introduce the FRG algorithm. Similar to FNG, FRG iteratively chooses the node with the largest unprocessed flow rate. Therefore, FRG is different from FNG only in line 4. For FRG, it chooses v_i^* such that $i^* = \text{argmax}_i \sum_{j \in S_i} d_j$. We can show that FRG also achieves an approximation ratio of $(1-o(1)) \ln m + 2$. The proof is similar to that of Theorem 3. We also divide the original problem into two subproblems. Every subproblem consists of the original topology and a subset of flows. The only difference is about the proof of Eq. (6). Eq. (6) shows that if we run the algorithm on these two subproblems, we will still place the same number of VNF instances at the nodes as the original problem. The basic

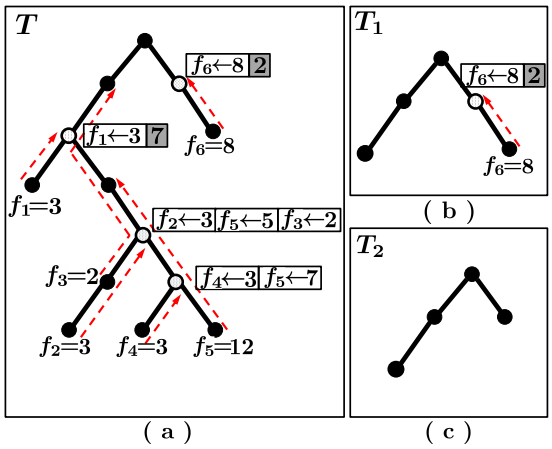

Fig. 2: An optimal solution generated by GFT for JPA-NFV with tree topology. Dashed lines denote the flow paths. A node is denoted by a hollow cycle if there is at least one VNF instance placed at the node. The rectangles next to the hollow nodes present the allocation of computing resource to the flows. Assume that the computing capacity of each VNF instance is $R = 10$.

idea is that the partition of flows will not affect which nodes we choose in every round. Note that Lemma 1 is also true for FRG. The time complexity of FNG and FRG are both $O(n^2 + mn)$. We provide the proof details and complexity analysis in the technical report [15] due to lack of space.

V. An Optimal Algorithm for Tree Topology

As described in our general model, for network operators who have their own datacenters within the core network, they may choose to implement their VNF instances that are scattered over different locations [17] [18]. This general model leads to an NP-hard problem as we described in Section III. However, some network services may require the network to have special topologies. Tree topologies are widely used for streaming services and Content Delivery Networks (CDNs) [19]. In such cases, by harnessing the properties of tree topologies, we propose an optimal solution for JPA-VNF under some simplifying assumptions.

A. An Optimal Algorithm for JPA-VNF with Tree Topology

We consider a tree network topology, denoted by T. Let l_p denote the number of nodes at the p-th level of the tree. We assume that the root is at level 1, which is the highest level. From left to right, all these l_p nodes are denoted by $\{v_{p,1}, v_{p,2}, \ldots, v_{p,l_p}\}$. We assume that all the flows are upstream flows (i.e., from a lower-level node to a higher-level node in the tree). We make this assumption for ease of presentation only; our results can be immediately generalized to cases where the flows are either upstream or downstream. Let $T_{p,q}$ be the subtree rooted at node $v_{p,q}$.

Our algorithm is based on a key observation: if we check all the nodes on the path of a flow in a bottom-up manner, we should not process the flow until it intersects other flows or it is about to leave the network. This is because processing a flow at a lower-level node may lose the opportunity to combine it with other flows at a higher level. Hence, a good strategy would be to not process the flow until it reaches the highest-level node along its path (i.e., at the node through which the flow leaves the network). Now, we propose our greedy algorithm based on this key idea. We call this algorithm Greedy For Tree *(GFT)*, which traverses all the nodes in the tree from the lowest level to the root node. Let $D_{p,q}$ be the set of all unprocessed flows leaving the network through node $v_{p,q}$ and $d_{p,q}$ be the total rate of all the flows in $D_{p,q}$. Once we reach a node $v_{p,q}$ through which a flow leaves the network, we place VNF instances at this node to process all the flows in $D_{p,q}$. Then, the problem would be how to allocate the remaining computing resource if the flow does not consume all the resource. For every node $v_{p,q}$, we create a waiting list $F_{p,q}$, which consists of all the unprocessed flows going through $v_{p,q}$. These flows are sorted in a nonincreasing order of the level of the node through which a flow leaves the network. Then, we allocate all the available computing resource to the first flow in the waiting list. The detailed operations of GFT are provided in Algorithm 2. To help understand the operations of GFT, we provide an example in Fig. 2a and present the detailed steps of this example in Table I. The time complexity of GFT is $O(n + m \log n)$. We provide the complexity analysis in our technical report [15].

Step	Node	Leaving flow	$F_{p,q}$	# of VNFs	Resource allocation
1	$v_{6,2}$	f_4	$f_4 = 3$ $f_5 = 12$	$\lceil \frac{3}{10} \rceil = 1$	$3 \rightarrow f_4$ $7 \rightarrow f_5$
2	$v_{5,1}$	f_2	$f_2 = 3$ $f_5 = 5$ $f_3 = 2$	$\lceil \frac{3}{10} \rceil = 1$	$3 \rightarrow f_2$ $5 \rightarrow f_5$ $2 \rightarrow f_3$
3	$v_{3,1}$	f_1	$f_1 = 3$	$\lceil \frac{3}{10} \rceil = 1$	$3 \rightarrow f_1$
4	$v_{2,2}$	f_6	$f_6 = 8$	$\lceil \frac{8}{10} \rceil = 1$	$8 \rightarrow f_6$

TABLE I: This table shows how to allocate VNF instances to the network shown in Fig. 2a under GFT.

B. Main Result: Optimality

In Theorem 4, we state our main result of optimality.

Theorem 4: GFT is optimal for tree topologies.

A key insight from our investigations for general graphs is to minimize the waste of resource by processing multiple flows together at the same node. Therefore, in order to prove the optimality of GFT, we need to check the nodes where resource waste happens. We define such nodes as *Breaking Points*. Consider a JPA-VNF problem and a feasible solution for this problem. A node is called a *Breaking Point* if it hosts at least one VNF instance whose computing resource is not fully utilized. Breaking points have a very important property in one particular type of solutions, which we define as *conservative solutions*. A solution is called conservative if every breaking point in the solution hosts at most one VNF

instance that is not fully utilized. Apparently, the solution given by GFT is conservative. We further introduce another notion called *external flows* and then state the property of breaking points in Lemma 2, which will be used to prove Theorem 4. For a node $v_{p,q}$, if the path of a flow has exactly one end within the subtree $T_{p,q}$ (including $v_{p,q}$), we call this flow an *external flow* of node $v_{p,q}$.

Lemma 2: Consider a conservative solution for a JPA-VNF problem with tree topology. Let $v_{p,q}$ be a breaking point. Suppose that node $v_{p,q}$ is the only breaking point within subtree $T_{p,q}$ and that $v_{p,q}$ does not have any external flows. Then, no other feasible algorithm can use fewer VNF instances in $T_{p,q}$ than the conservative solution.

Proof: Let N_T be the number of VNF instances in $T_{p,q}$ under the conservative solution. Note that $T_{p,q}$ only has one breaking point which is the root node. In this case, the VNF instances within $T_{p,q}$ only process the flows whose full paths are within the subtree. Let F_t be the set of all such flows. Note that all the flows in F_t must be processed within $T_{p,q}$. The total rate of the flows in F_t is $\sum_{f_j \in F_t} d_j$. Since the solution is conservative, computing resource waste occurs for at most one VNF, which must be at node $v_{p,q}$. This implies $\sum_{f_j \in F_t} d_j > (N_T - 1)R$. Therefore, no other feasible algorithm can place fewer than N_T VNF instances in $T_{p,q}$ to process all the flows in F_t. ■

The key idea of the proof is as follows. If none of the breaking points have external flows, we can iteratively remove the subtrees rooted at breaking points in a bottom-up manner. In each iteration, we remove a subtree with only one breaking point, which is its root. The solution given by GFT is conservative. Lemma 2 implies that no feasible algorithm can use fewer VNF instances in the subtrees. Every time we remove such a subtree, we construct a new JPA-VNF problem instance based on the remaining topology. Since there is no external flow for the subtree's root (i.e., the breaking point), the VNF instances left in the network can still form a conservative solution for the new instance. By doing this repeatedly, we can show that the algorithm achieves optimality after all breaking points are removed. However, breaking points can have external flows. In the following, we show that processing these external flows actually do not increase the number of VNF instances. We can simply remove all the external flows of the breaking points iteratively and get the simpler case as described above. The detailed proof is provided in the following.

Proof of Theorem 4: Consider a JPA-VNF problem $I = (T, F, R)$ on a tree topology T. Let N_g and N_o be the number of VNF instances used by GFT and an optimal algorithm. Our goal is to show the following:

$$N_g \leq N_o. \tag{9}$$

We first remove all the external flows of the breaking points such that none of the breaking points have external flows. After removing external flows, we do not change the placement of VNF instances. In this case, the computing resource allocated to process these flows will be wasted and may create new breaking points. To assist the analysis, we create a priority queue that consists of all the breaking points. The breaking points are sorted in a nondecreasing order of the level. The breaking points at the same level are sorted from left to right to break the tie. In each iteration, we check the breaking point at the head of the queue. We remove all the external flows of this breaking point if there is any and then remove it from the queue. If there are new breaking points generated in this process, those new breaking points are inserted into the priority queue. We repeat the procedure until the priority queue becomes empty. Note that the external flows of a breaking point must already be fully processed within the subtree rooted at this breaking point. Otherwise, the remaining resource of this breaking point would have been allocated to process it. Therefore, new breaking points would only appear at a lower level. Hence, this procedure scans all the breaking points (including the new ones coming up during this procedure) of the tree from the root to the leaves.

Next, we want to show that throughout the above procedure, removing external flows of a breaking point does not reduce the number of VNF instances placed within the subtree rooted at each breaking point. We prove this by contradiction. Suppose that the number of VNF instances decreases after an external flow is removed. Then, there must exist at least one VNF instance, whose computing resource is entirely used to process external flows. However, this could not happen because GFT would not activate a new VNF instance for an external flow in the first place. This implies for all the nodes that host VNF instances, only part of the computing resource of one VNF instance is used to process external flows of this node. This property also ensures that each of the new breaking points hosts at most one VNF instance that is not fully utilized.

We now consider the system with all the external flows removed for each breaking point. Let F_r denote the set of remaining flows. We can construct a new JPA-VNF problem $I' = (T, F_r, R)$. Assume that there are k breaking points in I', including all the new breaking points. We denote the set of all breaking points by $V' = \{v_{p_1,q_1}, v_{p_2,q_2}, \cdots, v_{p_k,q_k}\}$. The breaking points are sorted in a nondecreasing order of their level, i.e., $p_1 \geq p_2 \geq \cdots \geq p_k$. The nodes at the same level are sorted according to the second index q. Note that there remain N_g VNF instances in the system. As mentioned earlier, every breaking point hosts at most one VNF instance that is not fully utilized. Therefore, these N_g VNF instances form a conservative solution for I'. Assume that an optimal solution uses N'_o VNF instances to solve I'. It is easy to see that $N'_o \leq N_o$ since $F_r \subseteq F$. Therefore, in order to prove Eq. (9), it is sufficient to show the following:

$$N_g \leq N'_o. \tag{10}$$

In the sequel, we prove Eq. (10). We iteratively remove the subtrees rooted at the breaking points, by starting with the breaking point at the lowest level (i.e., v_{p_1,q_1}) and the

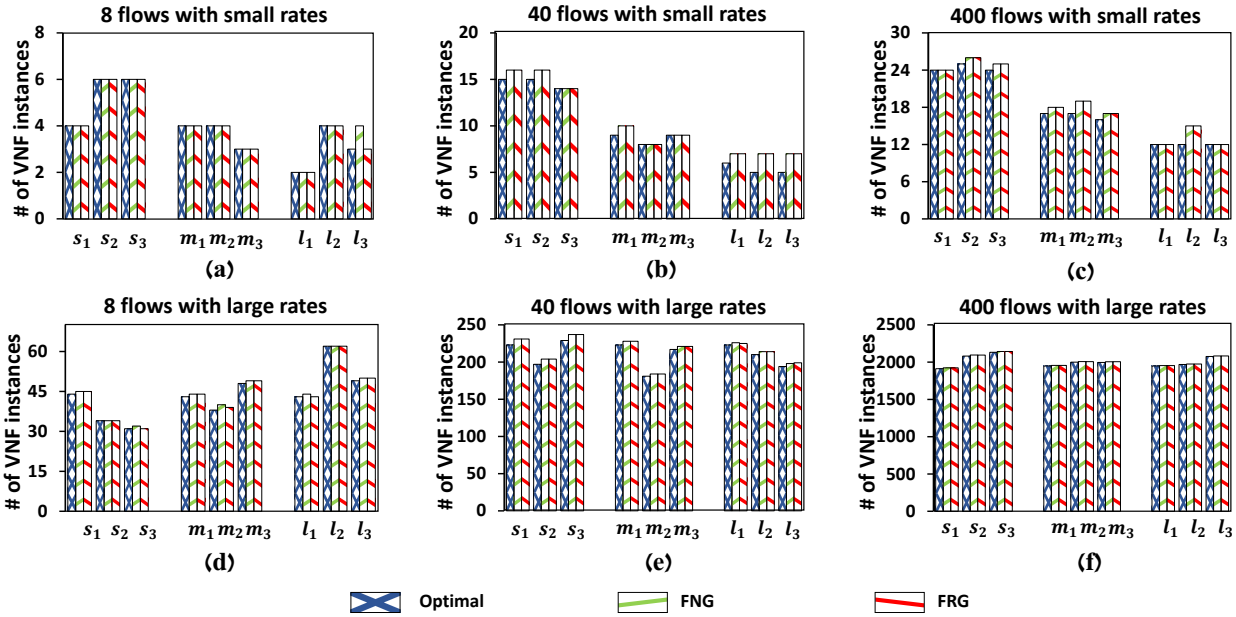

Fig. 3: Simulation results for the random topology.

corresponding subtree T_{p_1,q_1}. According to Lemma 2, no algorithm can put fewer VNF instances in T_{p_1,q_1}. Let N_{gs}^1 and N_{os}^1 be the number of VNF instances in subtree T_{p_1,q_1} for our solution and the optimal solution, respectively. Then, the following inequality follows from Lemma 2:

$$N_{gs}^1 \leq N_{os}^1. \tag{11}$$

We remove subtree T_{p_1,q_1} from T and also remove all the flows within T_{p_1,q_1}. Let T^1 be the remaining topology. Let the set of remaining flows be $F_r^1 \subseteq F_r$. Now, we have a new JPA-VNF problem $I^1 = (T^1, F_r^1, R)$.

We use N_g^1 and N_o^1 to denote the number of VNF instances left on T^1 after removing T_{p_1,q_1} under our algorithm and the optimal algorithm, respectively. Note that $N_g^1 = N_g - N_{gs}^1$ and $N_o^1 = N_o' - N_{os}^1$. Due to Eq. (11), in order to show Eq. (10), it remains to show $N_g^1 \leq N_o^1$.

We repeat the above procedure and argument until all the k breaking points are removed. Then, there are two cases for the remaining topology: (i) it is empty; and (ii) it is a tree without any breaking point. Case (i) is trivial. In Case (ii), let N_g^k and N_o^k denote the number of VNF instances left in the remaining topology. Since there is no breaking point, there is no resource waste for the N_g^k VNF instances. Hence, we have $N_g^k \leq N_o^k$. This completes the proof. ■

VI. Numerical Results

In this section, we evaluate the performance of our proposed greedy algorithms in various scenarios. We conduct simulations both for a randomly generated network topology (see Fig. 4). A more realistic backbone network topology from InternetMCI [14] is considered in our technical report [15].

The randomly generated network topology consists of 40 nodes and 234 links. We assume that each VNF instance has a processing capacity of 10, i.e., $R = 10$. We evaluate the performance of our proposed greedy algorithms by comparing them with the optimal solution computed by the GNU Linear Programming Kit (GLPK) [20]. For the problem instances we consider, GLPK computes an optimal solution within a reasonable amount of time.

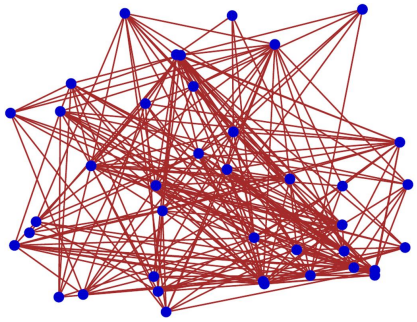

Fig. 4: A randomly generated topology with 40 nodes.

In order to obtain a comprehensive understanding of the empirical performance of our algorithms, we conduct simulations in various scenarios. Specifically, we consider the following settings: (i) different path lengths, (ii) different flow rates, and (iii) different number of flows. The simulation results are shown in Fig. 3. Each set of simulation results consists of 6 subfigures; each subfigure consists of the results of three different settings; each setting has three different simulation instances. We use the title and the x axis to distinguish different simulation settings. Along the x axis, we use "s", "m", and "l" to denote the setting where flows have short paths, medium paths, and long paths, respectively. Label "s_i" denotes the i-th simulation instance of the short-path setting. Other labels have similar meanings. The y axis is the total number

of VNF instances used in the network. We will discuss the impact of path length, flow rate, and topology complexity. The simulation results show that our algorithms tend to use much fewer nodes, which is a nice feature from the application point of view. The discussion about the number of nodes used for hosting the VNF instances is provided in our online technical report [15].

Impact of path length: We consider the following ranges for each type of flow path. A short path has a length uniformly distributed in the range of $[1, n/10]$ hops, where n is the number of nodes in the network. Similarly, the range for the length of a medium path and of a long path is $[1, n/4]$ hops and $[1, n/2]$ hops, respectively. A larger average path length implies that the flows have a bigger chance to intersect each others. This provides a larger room for optimization by combining the processing of multiple flows at fewer nodes. Therefore, the total number of VNF instances would decrease as the average path length becomes larger, especially when the flow rates are small (see Fig 3-(a), (b), and (c)) since an isolated small rate flow can generate large resource waste.

Impact of flow rate: Fig. 3-(a), (b), and (c) show the results where all the flows have small rates. The flow rates are uniformly distributed in the range of $[0, R/m]$, where m is the number of flows. Fig. 3-(d), (e), and (f) correspond to the results for large flow rates, which are uniformly distributed in the range of $[0, 10R]$. In both cases, the solutions generated by the greedy algorithms are very close to the optimal solution. An intuitive explanation is the following. When the flow rates are large, the density of the solution given by our algorithms is typically large (e.g., larger than 10). This leads to an approximation ratio close to 1 due to Lemma 1.

Impact of topology complexity: The above simulation results show that our proposed greedy algorithms empirically perform very well in a randomly generated dense network topology. However, in reality backbone network topologies are typically sparse. To that end, we also repeat our evaluations for a realistic backbone network topology of InternetMCI [14]. The simulation results show that when the topology of a network is more sparse, the performance of our proposed greedy algorithms becomes closer to that of the optimal solution. The reason is the following. When the network is smaller and the topology is more sparse, the room for optimization becomes smaller, and thus, the performance gap between our proposed greedy algorithms and the optimal solution also reduces. The simulation results are similar to Fig. 3 and are thus provided in our online technical report [15].

VII. Conclusion

In this paper, we studied the problem of joint placement and allocation of VNF instances in a new NFV-enabled networking paradigm. We proved that the formulated problem is NP-hard. Then, we proposed two simple greedy algorithms that are asymptotically optimal in general topologies and an optimal greedy algorithm for tree topologies. The simulation results elucidated our theoretical analyses. We believe that our analytical results provide important insights that will be useful in practice. However, several important issues remain unaddressed. First, we have assumed that the flow routes are fixed. It would be interesting to investigate the problem of joint VNF placement and flow routing. Second, we considered a simplified model that has only one single network function. It would be important to account for the practical constraint of service function chaining and design new algorithms with provable performance guarantees in such settings.

References

[1] "Etsi. network functions virtualisation introductory white paper." 2012. [Online]. Available: https://portal.etsi.org/NFV/NFV_White_Paper.pdf

[2] Y. Li, L. Phan, and B. T. Loo, "Network functions virtualization with soft real-time guarantees," in *Proceedings of IEEE INFOCOM*, 2016.

[3] D. Kreutz, F. M. Ramos, P. Esteves Verissimo, C. Esteve Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2015.

[4] A. Olteanu, Y. Xiao, K. Wu, and X. Du, "An optimal sensor network for intrusion detection," in *Proceedings of IEEE International Conference on Communications*, 2009, pp. 1–5.

[5] J. Lv, W. Yang, L. Gong, D. Man, and X. Du, "Robust wlan-based indoor fine-grained intrusion detection," *Proceedings of IEEE GLOBECOM*, 2016.

[6] R. Cohen, L. Lewin-Eytan, J. S. Naor, and D. Raz, "Near optimal placement of virtual network functions," in *Proceedings of IEEE INFOCOM*, 2015.

[7] M. Casado, T. Koponen, R. Ramanathan, and S. Shenker, "Virtualizing the network forwarding plane," in *Proceedings of the Workshop on Programmable Routers for Extensible Services of Tomorrow*, 2010, p. 8.

[8] M. F. Bari, S. R. Chowdhury, R. Ahmed, and R. Boutaba, "On orchestrating virtual network functions," in *Proceedings of 11th International Conference on Network and Service Management (CNSM)*, 2015, pp. 50–56.

[9] A. Gember, A. Krishnamurthy, S. S. John, R. Grandl, X. Gao, A. Anand, T. Benson, V. Sekar, and A. Akella, "Stratos: A network-aware orchestration layer for virtual middleboxes in clouds," *arXiv preprint arXiv:1305.0209*, 2013.

[10] A. Gember-Jacobson, R. Viswanathan, C. Prakash, R. Grandl, J. Khalid, S. Das, and A. Akella, "Opennf: Enabling innovation in network function control," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4, pp. 163–174, 2015.

[11] A. Mohammadkhan, S. Ghapani, G. Liu, W. Zhang, K. Ramakrishnan, and T. Wood, "Virtual function placement and traffic steering in flexible and dynamic software defined networks," in *The 21st IEEE International Workshop on Local and Metropolitan Area Networks*, 2015, pp. 1–6.

[12] B. Addis, D. Belabed, M. Bouet, and S. Secci, "Virtual network functions placement and routing optimization," in *Proceedings of IEEE 4th International Conference on Cloud Networking (CloudNet)*, 2015.

[13] T.-W. Kuo, B.-H. Liou, K. C.-J. Lin, and M.-J. Tsai, "Deploying chains of virtual network functions: On the relation between link and server usage," in *Proceedings of IEEE INFOCOM*, 2016.

[14] [Online]. Available: http://topology-zoo.org/maps/Internetmci.jpg

[15] "Provably efficient algorithms for joint placement and allocation of virtual network functions," January 2017, technical report. [Online]. Available: https://cis.temple.edu/~boji/publications.html

[16] U. Feige, "A threshold of ln n for approximating set cover," *Journal of the ACM (JACM)*, vol. 45, no. 4, pp. 634–652, 1998.

[17] J. Soares, M. Dias, J. Carapinha, B. Parreira, and S. Sargento, "Cloud4nfv: A platform for virtual network functions," in *Proceedings of IEEE 3rd International Conference on Cloud Networking (CloudNet)*, 2014, pp. 288–293.

[18] J. Soares, C. Goncalves, B. Parreira, P. Tavares, J. Carapinha, J. P. Barraca, R. L. Aguiar, and S. Sargento, "Toward a telco cloud environment for service functions," *Communications Magazine, IEEE*, vol. 53, no. 2, pp. 98–106, 2015.

[19] S. Seyyedi and B. Akbari, "Hybrid cdn-p2p architectures for live video streaming: Comparative study of connected and unconnected meshes," in *Proceedings of International Symposium on Computer Networks and Distributed Systems (CNDS)*, 2011, pp. 175–180.

[20] [Online]. Available: https://www.gnu.org/software/glpk/