Incremental Server Deployment for Scalable NFV-enabled Networks

Jianchun Liu[1,4] Hongli Xu*[2,4] Gongming Zhao[2,4] Chen Qian[3] Xingpeng Fan[2,4] Liusheng Huang[2,4]
[1]School of Data Science, University of Science and Technology of China
[2]School of Computer Science and Technology, University of Science and Technology of China
[3]Department of Computer Science and Engineering, University of California Santa Cruz, USA
[4]Suzhou Institute for Advanced Study, University of Science and Technology of China

Abstract*—Network Function Virtualization (NFV) is a new paradigm to enable service innovation through virtualizing traditional network functions. To construct a new NFV-enabled network, there are two critical requirements: *minimizing server deployment cost* and *satisfying switch resource constraints*. However, prior work mostly focuses on the server deployment cost, while ignoring the switch resource constraints (*e.g.*, switch's flow-table size). It thus results in a large number of rules on switches and leads to massive control overhead. To address this challenge, we propose an incremental server deployment (INSD) problem for construction of scalable NFV-enabled networks. We prove that the INSD problem is NP-Hard, and there is no polynomial-time algorithm with approximation ratio of $(1-\epsilon)\cdot\ln m$, where ϵ is an arbitrarily small value and m is the number of requests in the network. We then present an efficient algorithm with an approximation ratio of $2\cdot H(q\cdot p)$[1], where q is the number of VNF's categories and p is the maximum number of requests through a switch. We evaluate the performance of our algorithm with experiments on physical platform (Pica8), Open vSwitches, and large-scale simulations. Both experiment and simulation results show high scalability of the proposed algorithm. For example, our solution can reduce the control and rule overhead by about 88% with about 5% additional server deployment, compared with the existing solutions.

***Index Terms*—Incremental Server Deployment, Scalability, Rules, NFV.**

I. Introduction

Today's networks rely on a wide spectrum of specialized network functions (NFs) or middleboxes (MBs) [1] [2], such as firewalls, traffic monitors, web proxies, and instruction detection systems. They have been widely deployed in various networking scenarios, including campus networks, backbone networks, and data center networks. Network traffic usually needs to pass through several NFs in a particular order, which is known as a service function chain (SFC) [3]. For instance, in data centers, some requests need to traverse a firewall and a proxy in sequence, while other requests need only to traverse the firewall for security processing.

Due to the high price and inflexibility of physical NFs or MBs, Network Function Virtualization (NFV) [4] has been an emerging approach in which network functions are no longer executed by dedicated hardware but instead can be run on general-purpose servers located in cloud nodes [5], called Virtual Network Functions (VNFs) [6]. Compared with the physical NFs, the NFV technology contributes to reducing the price and improving the system flexibility. With these advantages of NFV, many users, including corporations, communities, and governments, are expecting to deploy an NFV-enabled network. Since scalability has been a core issue for large network development, there are two critical requirements of scalability, minimizing server deployment cost and satisfying switch resource constraints for rule configuration.

VNFs are running on the commodity general-purpose servers. The problem of VNF placement on servers has been widely studied in recent years for different targets, such as link/server load balancing, resource utility maximization, and reliability [7] [8] [9] [10] [11] [12]. Furthermore, due to traffic dynamics, different joint optimization problems have been investigated in literatures [13] [14]. Most of these studies by default assume that a set of servers have been deployed on given positions. In fact, for enterprise and edge networks, a large number of production servers are unavailable, and it is also challenging and time-consuming to find the right hosting servers. Moreover, with the increase of hosting servers, the complexity of VNF management, *e.g.*, fault diagnosis and localization, grows especially as the servers may be from different owners [15] [16]. Different from the existing work on VNF placement or the joint optimization problem, we mainly focus on the *incremental server placement* for VNFs so as to pursue the minimum deployment cost while satisfying hardware resource constraints.

Previous work [9] [17] has studied the incremental server deployment for network function virtualization. However, these methods have two main disadvantages of network scalability. First, almost all the previous solutions, *e.g.*, [9] [17], ignore the impact of the limited Ternary Content Addressable Memory (TCAM) size on the switches. TCAMs are 400$\times$ more expensive and consume 100$\times$ more power per Mbit than the RAM-based storage on the switches [18]. Besides, the lookup speed and insertion speed are highly related to the size of TCAM. As a result, most of today's commodity switches only support 4-20K entries [18] (*e.g.*, 6K entries on HP HPE6960 switches and 4K entries on PICA8 P-5401 switches [19]). The previous solutions implement the SFC routing with the granularity of ingress-egress pairs, which needs a large number of rules on switches for VNF processing. For example, a data center network with a thousand switches [20] may require up to millions of possible rules on a switch, which certainly does not fit the TCAM size. Moreover, installing

[1]$H(n)$ is the n-th harmonic number defined as $H(n)=1+\frac{1}{2}+...+\frac{1}{n}\approx\log n$.

Schemes	SFC Policy	No. of Rules	Overhead
GFT [9]	No	Many	High
T-SAT [17]	Yes	Many	High
Our work	Yes	Few	Low

TABLE I: Comparison of existing server deployment solutions and our scheme.

more TCAM rules also leads to massive control overhead. Second, some methods, *e.g.*, GFT [9], only focus on one type of network function, which can not be directly applied to the situation of SFCs. Though the work T-SAT [17] extends the server deployment to consider the SFC requirement, their algorithm can not guarantee the approximation performance. We summarize the advantages and disadvantages of the existing solutions and our scheme in Table I.

We believe it is necessary to design a new solution of incremental server deployment to construct a scalable NFV-enabled network with TCAM size constraint. Our solution is motivated by the following considerations. An SFC request is specified by an ingress switch, an egress switch and the SFC requirement. Since request-based SFC needs to install a massive number of rules on switches, we expect to use coarse-grained (*i.e.*, wildcard-based) rules to effectively reduce the TCAM cost and control overhead. To the best of our knowledge, *we are the first to propose a provably efficient algorithm for incremental server deployment within the network while taking the flow table size constraint into considerations.* The main contributions of this paper are:

- We propose an incremental server deployment (INSD) problem for the construction of scalable NFV-based networks and analyze its NP-Hardness. We also prove that there is no polynomial-time algorithm with an approximation ratio of $(1-\epsilon)\cdot\ln m$, where ϵ is an arbitrarily small value, and m is the number of requests in the network.
- We present an efficient and polynomial-time algorithm, called KPGD, for the INSD problem, and analyze the approximation ratio of $2\cdot H(q\cdot p)$, where q is the number of VNF's categories, and p is the maximum number of requests through a switch.
- We evaluate the performance of our proposed method with experiments on both physical platform (Pica8) and Open vSwitch (OVS), as well as large-scale simulations. Both experimental results and simulation results show that the proposed solution can achieve better scalability in terms of deployment and configuration cost. For example, our solution can reduce the control overhead by about 88% with deploying additional servers about 5%, compared with the existing solutions.

The rest of this paper is organized as follows. Section II formalizes the INSD problem and gives the inapproximation result. We propose an approximation algorithm for INSD and analyze the approximation performance in Section III. We report our simulation results and experimental results in Section IV. We conclude the paper in Section V.

II. Preliminaries and Problem Formulation

In this section, we define the network model in Section II-A. Besides, we illustrate the rule installment for one VNF instance (Section II-B) and SFC processing (Section II-C). Finally, we give the problem formulation in Section II-D.

Symbol	Semantics
V	a set of switches
V_e	a set of egress switches
$\mathcal{R}$	a set of requests
$\mathcal{R}_v$	a set of requests whose egress switch is $v \in V_e$
$\mathcal{F}$	a set of VNFs
q	the number of VNF's categories, *i.e.*, $q = \|\mathcal{F}\|$
p	the maximum number of requests on all switches
$c(v)$	the maximum number of rules for VNF processing on switch v
$c(s)$	the processing capacity of server s
$N(r)$	the number of packets of request r
θ_f	the processing cost per-packet of VNF f
x_v	a server is deployed on switch v or not
$y_{v,t}^f$	a rule matching egress switch t and VNF f is installed on switch v or not
δ_i^f	the requests covered by VNF f on switch v_i
α_i^f	rule cost of the requests covered by VNF f on switch v_i
β_i^f	processing cost of the requests covered by VNF f on switch v_i

TABLE II: Key Notations

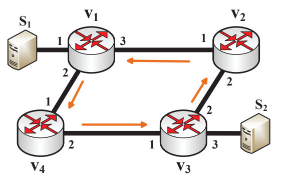
A. Network Model

An SDN is typically separated into the control plane and the data plane. The control plane consists of a logically-centralized controller, which may be a cluster of distributed controllers [21] [22] and is responsible for managing the whole network. The data plane consists of a set of n SDN switches, $V = \{v_1, ..., v_n\}$. Without loss of generality, the former z switches are egress switches, denoted as $V_e = \{v_1, ..., v_z\}$. The network topology can be modeled by a connected graph $G = (V, E)$, where E is the set of links connecting the switches. Since we focus on the data plane metrics (*e.g.*, the number of deployed servers and rules), the number of controllers will not significantly impact these metrics. For simplicity, we assume that there is only one controller in the control plane.

There is a set of VNFs, *e.g.*, firewalls, IDSes and proxies, denoted as $\mathcal{F} = \{f_1, f_2, ..., f_q\}$, with $q = |\mathcal{F}|$. Let θ_f indicate the processing cost per packet (measured by the number of CPU cycles) for each VNF f. Given a set of requests $\mathcal{R} = \{r_1, r_2, ..., r_m\}$ with $m = |\mathcal{R}|$, each request is specified by an ingress switch, an egress switch and the SFC requirement. For simplicity, if request r_i is processed by VNF f_j, we call that request r_i is *covered* by VNF f_j. Through long-term traffic observation, the controller has full knowledge of the requests, *e.g.*, the number of packets $N(r)$ of request $r \in \mathcal{R}$. We use $\mathcal{R}_v$ to represent the set of requests, whose egress switch is $v \in V_e$. For ease of reference, the key notations are listed in Table II.

B. Rule Installment for One VNF Instance

In this section, we will introduce the processing of rule installment on switches in the case of one VNF instance. To



Rule Installment under Different Schemes

Scheme	V1	V3
Request-based	6	6
Egress-based	2	2

Fig. 1: Rule Installment for One VNF Instance. The left plot illustrates an example with 4 switches and 2 servers. The right table illustrates the number of required rules on switches v_1 and v_3 under different rule installment schemes.

facilitate understanding, we illustrate rule installment through an example. On the left plot of Fig. 1, there are 4 switches and 12 ingress-egress switch pairs. Suppose that there are 12 requests in the network and each request between a switch pair is forwarded in a counter-clockwise direction. For example, the forwarding path from v_2 to v_4 is $v_2 \rightarrow v_1 \rightarrow v_4$. VNF instances are deployed on two servers, which are connected to switches v_1 and v_3, respectively. All the requests should be processed by a VNF instance. Assume that the paths of these requests are available to the controller with the help of SDN's centralized control. Then the controller configures 6 requests (*e.g.*, $v_2 \rightarrow v_1$, $v_3 \rightarrow v_1$, $v_4 \rightarrow v_1$, $v_1 \rightarrow v_4$, $v_2 \rightarrow v_4$, $v_3 \rightarrow v_4$) to be processed by server s_1, and other requests are processed by s_2. We focus on the rule installment on switches v_1 and v_3.

There are two different schemes of rule installment. First, the previous VNF placement solutions assume by default that the request-based rules will be installed on switches [2] [23]. Thus, there requires 12 rules for VNF processing in the network, *i.e.*, 6 rules on both switches v_1 and v_3. The rules for VNF processing on switch v_1 are listed in Table III. For example, we install a rule "$src = v_2, dst = v_1, inport = 3, actions = output : 1$" for request $v_2 \rightarrow v_1$.

Requests	Request-based	Egress-based
$v_2 \rightarrow v_1$	src=v_2, dst=v_1, inport=3, actions=output:1	dst=v_1, inport=3, actions=output:1
$v_3 \rightarrow v_1$	src=v_3, dst=v_1, inport=3, actions=output:1	
$v_4 \rightarrow v_1$	src=v_4, dst=v_1, inport=3, actions=output:1	
$v_1 \rightarrow v_4$	src=v_1, dst=v_4, inport=3, actions=output:1	dst=v_4, inport=3, actions=output:1
$v_2 \rightarrow v_4$	src=v_2, dst=v_4, inport=3, actions=output:1	
$v_3 \rightarrow v_4$	src=v_3, dst=v_4, inport=3, actions=output:1	

TABLE III: Installed Rules for VNF Processing on Switch v_1.

Second, to reduce the number of required rules, we then consider the egress switch based wildcard scheme for rule installment. Since this scheme just needs to install wildcard rule for each egress switch, only 4 rules are required for VNF processing in the network. Specifically, both switches v_1 and v_3 require to install two rules, as shown in Table III. Each wildcard rule only specifies the egress switch (*e.g.*, v_1 or v_4), and can match all the ingress switches in the network. For example, we need to install a rule "$dst = v_1, inport = 3, actions = output : 1$" for the three requests with the same egress switch v_1. The egress switch based scheme can reduce the number of required rules by 67% compared with the request based scheme in this example. Thus, we use the egress switch based scheme so as to meet the needs of less rule cost and less control overhead in our proposed solution.

C. Tag Operations for SFC Processing

It is worth noting that, even if the egress-based rules for VNF processing have been installed, the request still may not traverse the SFC properly. To this end, we adopt efficient tag operations to support SFC [2] [23]. Specifically, to record the SFC information, we use two fields (*e.g.*, VLAN, MPLS labels or other unoccupied fields) in the packet header as tags. The controller adopts unique identities (*e.g.*, $1, 2, ..., n$) to distinguish these n NFs in the network. For example, in a moderate-size network, the network may contain less than 255 NFs [24]. Then, it only requires 8 bits to differentiate 255 NFs. Moreover, the SFC's length is usually not more than 5 [24]. So, it will cost 5 bytes (or 40 bits) for the SFC information in the packet header. For some programmable switches (*e.g.*, Open vSwitches [25], barefoot switches [26]), adding two new fields into the packet header is easily implemented.

Assume that a request from subnet 10.1.1.0/24 to subnet 10.1.2.0/24 should traverse a service function chain: Firewall-IDS-NAT for security benefits. We use 0x01-0x02-0x03 to denote the SFC requirement. We adopt two fields, Network Functions Label Matching (*NFLM*) and *MPLS*, to match the next NF to be processed and to store the rest NFs in the SFC.

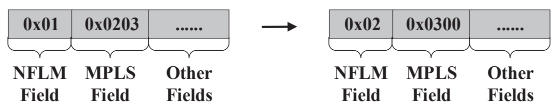


Fig. 2: Illustration of Tag Operations.

We illustrate the tag operations through an example. When a request arrives at the ingress switch, the controller configures the SFC policy (*e.g.*, *NFLM=0x01, MPLS=0x0203*), as shown in the left plot of Fig. 2. After the request has been processed by the VNF instance 0x01, the switch will update two fields in the packet header. The switch sets the *NFLM* field as the first NF (*i.e.*, 0x02) in the *MPLS* field, and removes this NF from the *MPLS* field. For example, after the request has been processed by the Firewall function, we set the *NFLM* field as 0x02 (*i.e.*, IDS), and update the *MPLS* field as 0x0300, as shown in the right plot of Fig. 2. More detailed information about tag operation and SFC routing can refer to our previous work [27]. Thus, the SFCs of all requests in the network can be processed properly according to the rule matching and tag operation.

D. Problem Definition

In this section, we give the definition of the Incremental Server Deployment (INSD) problem. The network administrators will specify the SFC processing requirement for each request [2]. Since we do not consider the VNF processing order in the server deployment, the set of VNFs in the SFC requirement of request r is denoted as $\mathcal{F}_r$. For example, if the SFC requirement of request r is Firewall-IDS-NAT, $\mathcal{F}_r = \{\text{IDS, NAT, Firewall}\}$. Note that the SFC requirements can be satisfied through efficient routing algorithms [2] [28]. For simplicity, we suppose that each VNF can work independently with others [7] [29].

Assume that the controllers have pre-computed the path for each request r [30] [31], denoted by p_r. We will also study the problem without pre-computed paths as our future work. The two resource constraints should be considered here. On one hand, we consider the rule cost for VNF processing. Let $y_{v,t}^f \in \{0,1\}$ denote whether a rule matching the egress switch t and VNF f will be installed on switch v or not. For example, as shown in Fig. 1, there is one request $v_4 \to v_2$ that needs to be processed by VNF f_1 placed on server s_2. Assume that we have installed a wildcard rule matching egress switch v_2 on switch v_3, which means $y_{v_3,v_2}^{f_1} = 1$. Then, all requests with the same egress switch v_2 will be forwarded to server s_2 for VNF processing. Due to the TCAM size constraint, we expect that the rule cost on switch v for VNF processing should not exceed a given threshold $c(v)$.

VNFs	CPU Cycles per Packet
Firewall	1348
NAT	1631
IDS	1348
Monitor	1676

TABLE IV: Per-Packet Processing Cost of VNFs. [29]

On the other hand, we consider the resource consumption for VNF processing on servers. The resources can be expressed in terms of CPU, memory and network bandwidth. The existing work [13] shows that CPU is usually the bottleneck resource for most VNF instances. Moreover, different VNFs require different numbers of CPU cycles for processing a packet. By testing in [29], we list the number of required CPU cycles for some typical VNFs in Table IV. According to installed rules on a switch, we know which flows will be processed on the connected server. As a result, we can derive the total processing cost on a server. We require that the total VNF processing cost on a server s should not exceed its computing capacity $c(s)$.

The objective of the INSD problem is to minimize the number of deployed servers in the network. Accordingly, we formulate the INSD problem as follows:

$$\min \sum_{v \in \mathcal{V}} x_v$$

$$s.t. \begin{cases} x_v \geq y_{v,t}^f, & \forall v \in V, t \in V_e, f \in \mathcal{F} \\ \sum\limits_{v \in p_r} y_{v,t_r}^f \geq 1, & \forall r \in \mathcal{R}, f \in \mathcal{F}_r \\ \sum\limits_{t \in V_e} \sum\limits_{f \in \mathcal{F}} y_{v,t}^f \leq c(v), & \forall v \in V \\ \sum\limits_{t \in V_e} \sum\limits_{r \in \mathcal{R}_v} \sum\limits_{f \in \mathcal{F}_r} y_{v,t_r}^f N(r)\theta_f \leq c(s_v), & \forall v \in V \\ x_v \in \{0,1\}, & \forall v \in V \\ y_{v,t}^f \in \{0,1\}, & \forall v \in V, t \in V_e, f \in \mathcal{F} \end{cases} \tag{1}$$

We use a binary variable x_v to indicate whether a server will be deployed on switch v or not. The first set of inequalities means that each request will be processed by a server only if the server has been deployed on switch v. The second set of inequalities means that each VNF $f \in \mathcal{F}_r$ should be deployed at least once along the path of request r, where t_r denotes the egress switch of request r. The third set of inequalities expresses the flow-table size (FTS) constraint for VNF processing on a switch. The fourth set of constraints tells that the total cost for VNF's processing should not exceed the server's computing capacity. The objective is to deploy a minimum number of servers for NFV-enabled networks.

Theorem 1. *The INSD problem is NP-Hard.*

Proof. Consider an instance of the Minimum Set Cover (MSC) problem [32]: let $E = \{l_1, l_2, ..., l_x\}$ be a set of a elements, $C = \{E_i \subseteq E, i = 1, 2, ..., y\}$ is a set of subsets of E, where $y = |C|$. MSC will choose a minimum set $C' \subseteq C$ such that each element $l \in E$ is contained in at least one member of C'. Then, we consider a special case of the INSD problem, in which there is only one VNF in the network and each server is equipped with the infinite computing capacity. Each request is abstracted as an element in E and the request set through switch v_i is abstracted as E_i. We expect to deploy the minimum number of servers to cover all requests in the network. Thus, the special instance of the INSD problem becomes the traditional MSC problem, which is NP-Hard. Accordingly, the INSD problem is NP-Hard too. □

Theorem 2. *The INSD problem cannot be solved by a polynomial time algorithm with an approximation ratio of $(1-\epsilon) \cdot \ln m$, for any $\epsilon > 0$, where m is the number of requests in the network, unless $P = NP$.*

Proof. Some previous works, *e.g.*, Raz and Safra [33], Feige [34], have proved that the MSC problem cannot be approximable within $(1-\epsilon) \cdot \ln n$, for any $\epsilon > 0$, where n is the number of elements in the MSC problem, unless $P = NP$. Since the MSC problem is a special case of our INSD problem, if there exists an algorithm with a better approximation ratio than $(1-\epsilon) \cdot \ln m$, where m is the number of requests in the network, for INSD, this algorithm can also be applied to solve the MSC problem, which contradicts with the previous inapproximation results. Thus, we can conclude that the inapproximation ratio of the INSD problem is $(1-\epsilon) \cdot \ln m$, for any $\epsilon > 0$. □

III. Algorithm Design for INSD

In this section, we design an approximation algorithm for the INSD problem (Section III-A) and give the performance analysis (Section III-B). Moreover, we give some discussions to enhance our proposed solution (Section III-C).

A. A Knapsack-based Algorithm for INSD

In this section, we present a knapsack-based approximation algorithm, called KPGD, to solve the INSD problem. We first consider a simple case, in which server s_i has been deployed on switch v_i. We will place feasible VNFs on this server so as to maximize the number of requests covered by these VNFs. When VNF f has been placed on server s_i, some requests will be covered (or processed) by f under the server's and switch's capacity constraints. Therefore, we regard this case as the 0-1 knapsack problem [35] [36]. Specifically, the knapsack capacity is the joint consideration of the server's processing capacity and switch's flow-table size constraints. The item size is the VNF's processing cost and the item's profit is the number of requests covered by this VNF. Similar to the knapsack problem, the goal of this version is to maximize the number of requests that can be covered by these VNFs deployed on the server.

Algorithm 1 KP Algorithm on Switch v_i

```
1: $\mathcal{P} \leftarrow \phi$, $\widetilde{\mathcal{P}} \leftarrow \mathcal{F}$
2: $c_v^i \leftarrow c(v_i), c_s^i \leftarrow c(s_i)$
3: for each VNF $f \in \widetilde{\mathcal{P}}$ do
4:     $\delta_i^f \leftarrow \sum_{t \in V_e^i} |\gamma_t^f|$, $\alpha_i^f \leftarrow |\Gamma_i^f|$
5:     $\beta_i^f \leftarrow N(\delta_i^f) \cdot \theta_f$, $\vartheta_i^f \leftarrow (\alpha_i^f, \beta_i^f)$
6:     $\widetilde{\mathcal{P}} \leftarrow \widetilde{\mathcal{P}} - \{f\}$
7: rearrange the VNFs $f \in \mathcal{F}$ in the decreasing order with
   the unit profit value $\frac{\delta_i^f}{\|\vartheta_i^f\|}$
8: while $\alpha_i^f \le c_v^i$ and $\beta_i^f \le c_s^i$ do
9:     $\mathcal{P} \leftarrow \mathcal{P} \bigcup \{f\}$
10:    $c_v^i \leftarrow c_v^i - \alpha_i^f$
11:    $c_s^i \leftarrow c_s^i - \beta_i^f$
12:    $\mathcal{F} \leftarrow \mathcal{F} - \{f\}$
13: for each VNF $f \in \mathcal{F}$ do
14:    if $\sum_{f_j \in \mathcal{P}} \delta_i^{f_j} \le \delta_i^f$ then
15:        $\mathcal{P} \leftarrow \{f\}$
16: return $\mathcal{P}$
```

We adopt a greedy algorithm, called KP, to place some VNFs on a server so that more requests can be covered by these VNFs. We first construct a set of request subsets $\Gamma = \{\gamma_{v_1}^{f_1}, ..., \gamma_{v_1}^{f_q}, ..., \gamma_{v_z}^{f_1}, ..., \gamma_{v_z}^{f_q}\}$, where $f_i \in \mathcal{F}$ and $v_j \in V_e$. $\gamma_{t_j}^{f_i}$ denotes the set of requests with egress switch t_j that needs to be processed by VNF f_i. Let $\Gamma_i \subseteq \Gamma$ be the set of requests through switch v_i. Moreover, we use Γ_i^f to denote the set of requests that need to be processed by VNF f in Γ_i. We denote $\mathcal{P}$ (or $\widetilde{\mathcal{P}}$) as the set of chosen (or unchosen) VNFs on this server. The profit value δ_i^f means the number of requests that can be covered by VNF f on server s_i. Besides, we use α_i^f and β_i^f to denote the rule cost and VNF's processing cost of the requests covered by VNF f on server s_i, respectively. For convenience of computing, the two cost variables are vectorized, which is denoted as $\vartheta_i^f = (\alpha_i^f, \beta_i^f)$. We use $\|\vartheta_i^f\|$ to represent the norm of the vector, *i.e.*, $\|\vartheta_i^f\| = \sqrt{(\alpha_i^f)^2 + (\beta_i^f)^2}$. The variable $V_e^i \subseteq V_e$ indicates a set of egress switches of the requests which pass through switch v_i.

The KP algorithm is described in Alg. 1. At the beginning, the KP algorithm initializes some variables, *e.g.*, c_v^i and c_s^i, to store the available rule and computing resources for VNF placement (Line 1-2). We compute the profit of each unchosen VNF $f \in \widetilde{\mathcal{P}}$. Moreover, we compute the rule cost and CPU processing cost for each f, respectively (Line 3-8). The algorithm ranks all the unchosen VNFs in the decreasing order of the unit profit (Line 9). We then greedily choose the VNFs with the maximum unit profit under the server processing capacity and FTS constraints for VNF processing (Line 10-14). At the end of each iteration, the set of placed VNFs and the available computing/rule resources will be updated (Line 15-17).

We then propose the greedy KPGD algorithm for the INSD problem. According to the problem definition, each request r should traverse the specific SFC in the network. In other words, each type of VNF $f \in \mathcal{F}$ needs to cover the request set $\Gamma^f = \{\gamma_{v_1}^f, \gamma_{v_2}^f, ..., \gamma_{v_z}^f\}$, where v_j is an egress switch. The rest number of requests that VNF f needs to cover is denoted as g_f. Let U_i^f be the set of requests that are uncovered by VNF f on server s_i. The KPGD algorithm is formally described in Alg. 2. Our proposed algorithm consists of a group of iterations, each of which includes two main steps. In the first step, the algorithm adopts the KP algorithm to derive the set of chosen VNFs, denoted as $\mathcal{P}_i$, for each switch v_i (Line 8-10). We choose one switch with the maximum profit for server deployment (Line 11-12). In the second step, the KPGD algorithm updates the uncovered request set (Line 13-18). The algorithm will terminate until each request r has been covered by any VNF in $\mathcal{F}_r$.

Algorithm 2 KPGD: Greedy Algorithm for INSD

```
1: $V' \leftarrow \phi$
2: for each VNF $f \in \mathcal{F}$ do
3:     $g_f \leftarrow |\Gamma^f|$
4:     for each switch $v_i \in V$ do
5:         $U_i^f \leftarrow \Gamma_i^f$
6: while $g_f > 0, \forall f \in \mathcal{F}$ do
7:     Step 1: Choose one switch to deploy a server
8:     for each switch $v_i \in V - V'$ do
9:         Choose the set of VNFs according to the KP algo-
           rithm, $\mathcal{P}_i \leftarrow KP(v_i)$
10:    Select a switch $v_i$ with the maximum profit $\sum_{f \in \mathcal{P}_i} \delta_i^f$
       and deploy a server
11:    $V' \leftarrow V' \bigcup \{v_i\}$
12:    Step 2: Update the request set
13:    for each VNF $f \in \mathcal{P}_i$ do
14:        $g_f \leftarrow g_f - |U_i^f|$
15:        for each switch $v_j \in V - V'$ do
16:            $U_j^f \leftarrow U_j^f - U_i^f$
17: return $V'$
```

B. Performance Analysis for KPGD

In this section, we analyze the approximation performance of the KPGD algorithm. KPGD consists of several iterations, each of which will execute the KP algorithm. Some prior works [35] [36] have proved that the KP algorithm can achieve an approximation ratio of 2 for the 0-1 knapsack problem.

Assume that our proposed KPGD algorithm and the optimal solution will deploy ρ and η servers for the INSD problem, respectively. We first give some preliminaries for the following analysis. We use $v(k)$ to denote the index of the chosen switch for server deployment in the k-th iteration of the KPGD algorithm. The set of requests that have not been covered by VNF f on server s_i after the k-th iteration, is denoted as $\Gamma_i^f(k)$, *i.e.*, $\Gamma_i^f(k) = \Gamma_i^f - \bigcup_{l=1}^k \Gamma_{v(l)}^f$. Specially, $\Gamma_i^f(0) = \Gamma_i^f$.

Without loss of generality, assume that the optimal solution for INSD will choose one switch for server deployment during each iteration. In the optimal solution, the set of requests covered by VNF f is denoted as $\widetilde{\Gamma}^f$ and will be updated with algorithm execution. At the beginning, $\widetilde{\Gamma}^f = \phi$ for each VNF $f \in \mathcal{F}$. When switch v_i is chosen to deploy a server s_i, a set of requests, that will be covered by VNF f placed on server s_i, is denoted as A_i^f, *i.e.*, $A_i^f = \Gamma_i^f - \widetilde{\Gamma}^f$. Similarly, we define $A_i^f(k)$ as the set of requests in A_i^f that have not been covered by VNF f after the k-th iteration. That is $A_i^f(k) = A_i^f - \bigcup_{l=1}^{k} \Gamma_{v(l)}^f$.

The set of chosen VNFs on server s_i in the k-th iteration is denoted as $\mathcal{P}_i(k)$ and the profit of VNF f on server s_i in the k-th iteration is $\delta_i^f(k) = |\Gamma_i^f(k-1) - \Gamma_i^f(k)|$. Then we can derive that the total profit $\delta_i(k) = \sum_{f\in\mathcal{P}_i(k)} \delta_i^f(k)$ on this server. The total cost on server s_i in the k-th iteration is $\vartheta_i(k) = \sum_{f\in\mathcal{P}_i(k)} \|\vartheta_i^f\|$. Besides, in the k-th iteration of KPGD, the profit of covered requests in A_i^f is denoted as $\varphi_i^f(k) = |A_i^f(k-1) - A_i^f(k)|$. Thus, the total profit in the k-th iteration is $\varphi_i(k) = \sum_{f\in\mathcal{F}} \varphi_i^f(k), \forall f \in \mathcal{F}$. It follows $\varphi_i(k) \le \sum_{f\in\mathcal{F}} |A_i^f|$.

Lemma 3. *For each $k \in \{1,2,...,\rho\}$, it follows*

$$\varphi_i(k) \le 2 \cdot \delta_{v(k)}(k) \tag{2}$$

Proof. In each iteration of KPGD, the knapsack problem can be solved for each switch which has not been chosen to deploy the server. $h_i(k)$ denotes the optimal profit of KPGD on switch v_i in the k-th iteration. So we have $\varphi_i(k) \le h_i(k)$. Since the approximation ratio of KP is 2, and $\delta_i(k)$ is the approximate result of KP, we can derive that $h_i(k) \le 2 \cdot \delta_i(k)$. Because KPGD always chooses a switch with the maximum profit for server deployment in each iteration, we have $\delta_i(k) \le \delta_{v(k)}(k)$. Thus, we can conclude that $\varphi_i(k) \le 2 \cdot \delta_{v(k)}(k), \forall k \in \{1,2,...,\rho\}$. □

In the KPGD algorithm, the set of requests that need to be covered by VNF f after the k-th iteration is denoted as $\lambda_f(k)$. $\lambda(k)$ is the total number of the requests that need to covered by all VNFs, *i.e.*, $\lambda(k) = \sum_{f\in\mathcal{F}} \lambda_f(k)$. Note that $\lambda(0) = \sum_{i=1}^{\eta} \sum_{f\in\mathcal{P}_i^*} |A_i^f|$. Then we prove that the total profit of the chosen switch in the k-th iteration is more than a given value as follows. The request set will be covered x times, if it is covered by x types of VNFs. We define the possible times the requests are covered as an integer variable $\tau_i \in \{1,2,...,\sum_{f\in\mathcal{P}_i^*} |A_i^f|\}$ for $\forall i \in \{1,2,...,\eta\}$. Thus, there are possibly $\lambda(0) = \sum_{i=1}^{\eta} \sum_{f\in\mathcal{P}_i^*} |A_i^f|$ integers, perhaps including some duplicated values. Then we rearrange these integer values into a non-decreasing sequence. For simplicity, we use e_x to denote these values and set them as $e_1 \le e_2 \le ... \le e_{\lambda(0)}$. For example, let $\eta = 3$, $\tau_1 = \{1\}, \tau_2 = \{1,2,3\}, \tau_3 = \{1,2\}$, so there are 6 integers, *i.e.*, $\lambda(0) = 6$. We rearrange these 6 integers as $1 \le 1 \le 1 \le 2 \le 2 \le 3$.

Lemma 4. *For each $k \in \{1,2,...,\rho\}$, we have*

$$e_{\lambda(k)} \le 2 \cdot \delta_{v(k)}(k) \tag{3}$$

Proof. After the k-th iteration, the KPGD algorithm will incrementally cover the requests $\lambda(k)$ times. If KPGD covers all rest requests in the optimal sets, the cover ratio for each VNF $f \in \mathcal{F}$ can be guaranteed. That is, $\sum_{i=1}^{\eta} \varphi_i(k) \ge \lambda(k)$. According to the definition of e_x and $\varphi_i(k)$, we can derive that e_x is not more than $\varphi_i(k)$, *i.e.*, $e_x \in \{1,2,...,\varphi_i(k)\}$, $\forall i \in \{1,2,...,\eta\}$. According to Lemma 3, we have

$$e_x \le 2 \cdot \delta_{v(k)}(k) \tag{4}$$

Since

$$\sum_{i=1}^{\eta} \varphi_i(k) \ge \lambda(k) \text{ and}$$
$$\varphi_i(k) \le \max_{s\le\eta}\{\sum_{f\in\mathcal{F}} |A_s^f|\}, \forall i \in \{1,2,...,\eta\}$$

There are at least $\lambda(k)$ indices x satisfying Eq. (4). Combining $e_1 \le e_2 \le ... \le e_{\lambda(0)}$, we can derive that $e_{\lambda(k)} \le 2 \cdot \delta_{v(k)}(k), \forall k \in \{1,2,...,\rho\}$. □

For ease expression, we use q and p to denote the number of VNF's categories and the maximum number of requests through a switch in the network, respectively.

Theorem 5. *Our proposed KPGD algorithm can achieve $2 \cdot H(q \cdot p)$-approximation for the INSD problem.*

Proof. According to Lemma. 4, for each $k \in \{1,2,...,\rho\}$, we have

$$2 \cdot \delta_{v(k)}(k) \ge e_{\lambda(k)} \ge e_{\lambda(k)-1} \ge ... \ge e_{\lambda(k+1)+1}$$
$$\Rightarrow \frac{1}{2 \cdot \delta_{v(k)}(k)} \le \frac{1}{e_{\lambda(k)}} \le \frac{1}{e_{\lambda(k)-1}} \le ... \le \frac{1}{e_{\lambda(k+1)+1}}$$

Since $\lambda(k) - \lambda(k+1) = \delta_{v(k)}(k)$, it follows

$$1 \le 2 \cdot (\frac{1}{e_{\lambda(k)}} + \frac{1}{e_{\lambda(k)-1}} + ... + \frac{1}{e_{\lambda(k+1)+1}})$$

Combining the above inequalities, we can derive that

$$\begin{aligned}\rho \le& 2 \cdot (\frac{1}{e_{\lambda(1)}} + ... + \frac{1}{e_1}) \le 2 \cdot (\frac{1}{e_{\lambda(0)}} + ... + \frac{1}{e_1}) \\ =& 2 \cdot \sum_{i=1}^{\eta} H(\sum_{f\in\mathcal{P}_i^*} |A_i^f|) \le 2 \cdot \eta \cdot H(q \cdot p)\end{aligned} \tag{5}$$

We should note that the third equation in Eq. (5) is based on the definition of e_x and the last inequality in Eq. (5) is based on $\sum_{f\in\mathcal{P}_i^*} |A_i^f| \le \sum_{f\in\mathcal{F}} |\Gamma_i^f| \le q \cdot p$. So we have

$$\frac{\rho}{\eta} \le 2 \cdot H(q \cdot p)$$

Thus, we can conclude that KPGD can achieve $2 \cdot H(q \cdot p)$-approximation for the INSD problem. □

C. Discussion

In this section, we discuss some practical issues to enhance the proposed solution.

1) In practice, the number of requests in the network will vary from time to time. For example, the number of requests may peak during the day and underestimate at night. However, deploying servers over time is unrealistic and requires a lot of resources (*e.g.*, time or deployment cost), even leading to network failure. Thus, we focus on server deployment during the request peak, so that time-varying requests can also be processed by the specific network functions.

2) After the server deployment problem has been solved, the SFC requirement will be posed for request routing. Different methods can be applied for SFC routing. In addition to our proposed scheme of tag operations, Network Service Header

(NSH), which is a data plane transmission protocol, is also a practical solution for SFC routing. It realizes the strategy of SFC control plane and helps users create and deploy SFC dynamically. Since this paper focuses on the server deployment problem, we ignore the detailed implementation of SFC routing here.

IV. Performance Evaluation

This section first introduces the metrics and benchmarks for performance comparison (Section IV-A). Then, we evaluate our proposed algorithm by comparing with the previous methods through large-scale simulations (Section IV-B). Finally, we implement our algorithm on the SDN platform with physical Pica8 switches [37] and Open vSwitches (OVSes) [25], and give the testing results (Section IV-C).

A. Performance Metrics and Benchmarks

In this paper, we design the incremental server deployment algorithm for the construction of scalable NFV-enabled networks. We adopt the following metrics to evaluate scalability and efficiency of our proposed algorithm.

1) The number of deployed servers. To satisfy the SFC requirements of all the requests, more servers will be deployed in the network with the increasing number of requests. We count the number of deployed servers in the network. Besides, the server utilization is the CPU computing load on a server divided by its computing capacity. Low server utilization indicates a huge waste of computing resources on servers. We focus on the CPU resource consumption for VNF processing of all servers in the network.
2) The maximum (or Max.) number of rules on any switch at any time during the simulation. When the servers have been deployed in the network, rules should be installed for VNF processing on the switches. Thus, we measure the number of installed rules on each switch and determine the maximum number of rules among all switches.
3) Considering traffic dynamics (*e.g.*, traffic rate fluctuation), if all requests follow the wildcard rules for VNF processing, it may lead to processing congestion on some server(s). Thus, we need to install some extra request-based rules in the test-bed evaluation so that some requests will be processed on different servers. How to install extra rules for congestion avoidance is similar to the solution for link congestion avoidance in [38]. Due to space limitations, we omit the detailed description here. During the update process, *update delay* and *control overhead* is important for scalability. Specifically, for update delay, we measure the during time of update procedure. Moreover, for control overhead, the total amount of traffic was measured between the conrtoller and the switches during the update procedure. We use a tool Cbench [39] to test the performance of OpenFlow controllers.

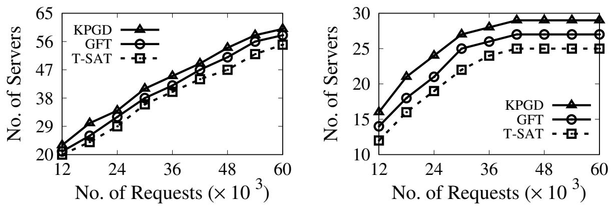


Fig. 3: No. of Servers vs. No. of Requests without FTS Constraint. *Left plot*: Topology (a); *right plot*: Topology (b).

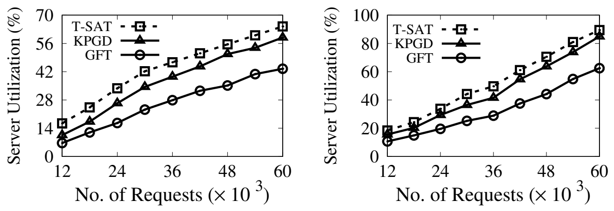


Fig. 4: Server Utilization vs. No. of Requests without FTS Constraint. *Left plot*: Topology (a); *right plot*: Topology (b).

B. Simulation Evaluation

1) Benchmarks: To evaluate how well our proposed algorithm performs, we compare with the other two benchmarks. Sang *et al.* [9] study the VNF placement problem, which minimizes the total number of VNF instances, subject to the constraint that all the requests need to be fully processed. The proposed algorithm, called GFT, considers the joint placement and allocation of VNF instances in a new NFV-enabled networking paradigm. Not only does the algorithm need to decide how many VNF instances to place on each server, but also need to determine how to allocate the computing resource for each VNF instance to process the requests through each switch. The second benchmark is called T-SAT [17], which addresses the VNF placement problem. The proposed solution first accomplishes the mapping of the SFCs or VNFs, then determines the placement of the related VNFs and allocates resources for VNFs based on the mapping results and the workloads of VNFs. Both two benchmarks process packets according to the request granularity.

2) Simulation Settings: In the simulations, as running examples, we select two typical and practical topologies, one for data center networks and the other for campus networks. The first topology, denoted as (a), is the fat-tree topology [40], which has been widely used in many data center networks. The fat-tree topology contains in total 320 switches (including 128 edge switches, 128 aggregation switches, and 64 core switches) and 1024 terminals. The second one, denoted as (b), is a campus network topology [41]. The topology (b) contains 100 switches and 200 terminals. We generate requests following DCTCP (data center TCP) and CPTCP (campus TCP) patterns for two topologies [42]. Since the topologies do not provide VNF information, we assume that there have deployed 5 types of VNFs (*e.g.*, IDS, Proxy, Monitor, Firewall and IPSec) on servers. We randomly generate an SFC requirement for each request. We execute each simulation 100 times, and take the average of the numerical results.

3) Simulation Results: We run four groups of simulations on two different topologies to check the effectiveness of the

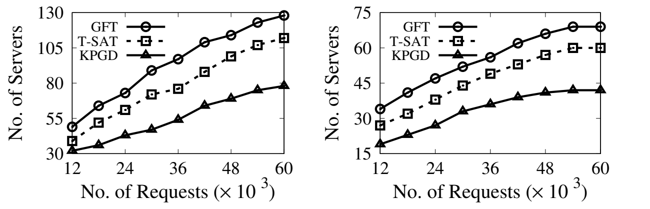


Fig. 5: No. of Servers vs. No. of Requests with FTS Constraint. *Left plot*: Topology (a); *right plot*: Topology (b).

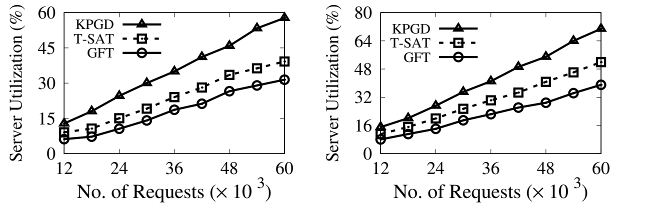


Fig. 6: Server Utilization vs. No. of Requests with FTS Constraint. *Left plot*: Topology (a); *right plot*: Topology (b).

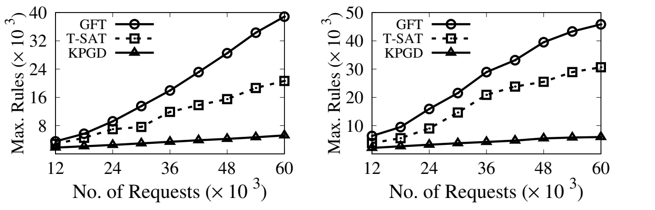


Fig. 7: Max. Rules vs. No. of Requests. *Left plot*: Topology (a); *right plot*: Topology (b).

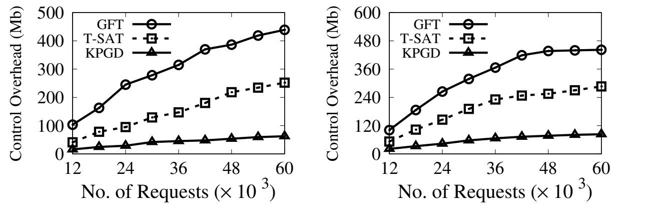


Fig. 8: Control Overhead vs. No. of Requests. *Left plot*: Topology (a); *right plot*: Topology (b).

proposed algorithms.

The first set of two simulations observes the deployment cost (*e.g.*, the number of servers) and server utilization without the FTS constraint on switches. Fig. 3 shows the number of deployed servers by changing the number of requests in both two topologies. With more requests from 12K to 60K, the number of deployed servers in the network is almost linearly increasing in topology (a). Given a fixed number of requests, the number of deployed servers by three solutions is very close. However, KPGD may deploy a little more servers than GFT and T-SAT on both two topologies. For example, when there are 60K requests in the network, KPGD needs to deploy 63 servers, while GFT and T-SAT need to deploy 60 and 58 servers in topology (a), respectively. In conclusion, our KPGD algorithm will increase the server deployment cost by about 5-9% compared with GFT and T-SAT. That's because both GFT and T-SAT control the requests in a fine-grained manner. However, two solutions require a massive number of rules on switches compared with our solution, which will be illustrated in Fig. 7.

Fig. 4 shows the server utilization of three algorithms. The figure demonstrates that server utilization is increasing linearly with more requests in both two topologies. In topology (a), when there are 36K requests, the server utilization of GFT is less than that of both KPGD and T-SAT. That is, KPGD can improve server utilization by about 10% compared to GFT. However, KPGD may achieve slightly ($<$ 5% on average) worse performance in terms of server deployment cost without the FTS constraint compared with T-SAT and GFT.

The second set of simulations observes the number of deployed servers and server utilization with the FTS constraint (*e.g.*, 4K) for VNF processing by changing the number of requests from 12K to 60K in the network. By the left plot of Fig. 5, our proposed solution can reduce the number of deployed servers compared with the other two solutions. For example, when there are 60K requests in the network, the number of deployed servers for KPGD is 76, while T-SAT and GFT need to deploy 112 and 128 servers. So KPGD can reduce the number of deployed servers by about 32% and 41% compared with T-SAT and GFT, respectively. That's because GFT and T-SAT install the rules on the switches for VNF processing with the request granularity, which may require a massive number of rules and violate the FTS constraint. KPGD can effectively reduce the number of installed rule by using the wildcard. Besides, Fig. 6 shows that KPGD can improve server utilization by about 47% and 45%, respectively, compared with GFT and T-SAT in topology (b). Thus, our proposed solution can deploy fewer servers and achieve better server utilization than the other two solutions with the FTS constraint.

The third set of two simulations observes the TCAM consumption (*e.g.*, the maximum number of rules) of three solutions. As shown in Fig. 7, the maximum number of required rules increases for all solutions with the increasing number of requests. However, the increasing ratio of KPGD is much slower than that of the other two benchmarks. In comparison, KPGD requires fewer rules than GFT and T-SAT. For example, given 36K requests in topology (a), KPGD uses a maximum number of 1.4K rules, while T-SAT and T-SAT need about 12.1K and 17.9K rules, respectively. In other words, KPGD can reduce the maximum number of required rules by about 88% and 92% compared with T-SAT and GFT, respectively. Therefore, our proposed solution can significantly reduce the TCAM consumption of all switches compared with the existing solutions.

The last set of simulations observes the performance in terms of control traffic overhead of three solutions, including GFT, T-SAT and KPGD. As shown in Fig. 8, with the number of requests increasing, GFT and T-SAT deploy more rules than KPGD, leading to higher control traffic overhead. For example, when there are 36K requests in topology (a), the control overhead of KPGD is 40Mb, while that of T-SAT and GFT increases to 150Mb and 340Mb, respectively. In other words, KPGD can reduce control overhead by about 73% and 88% compared with T-SAT and GFT, respectively.

From these simulation results in Figs. 3-8, we can make the following three conclusions. First, by Figs. 3-6, our KPGD solution may achieve a slightly worse but comparable performance ($<$ 5% on average) in terms of servers deployment cost and server utilization without the FTS constraint. However, KPGD can reduce the server deployment cost by about 36% and improve server utilization by about 47% compared with

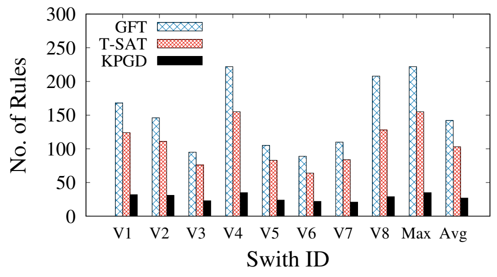


Fig. 9: No. of Rules on Each Switch in Telstra Topology

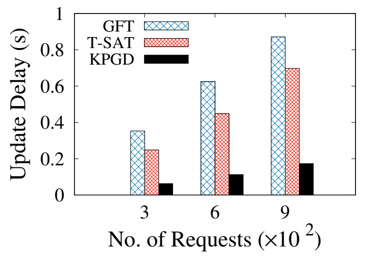


Fig. 10: Update Delay vs. No. of Requests in Telstra Topology

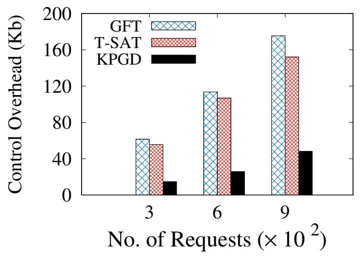


Fig. 11: Control Overhead vs. No. of Request in Telstra Topology

GFT and T-SAT with the FTS constraint. Second, by Figs. 7, our proposed solution can reduce the number of required rules by 90% compared with two benchmarks. Third, Fig. 8 shows that KPGD can reduce the control overhead by about 88% and 73% compared with GFT and T-SAT, respectively. These results show that our KPGD algorithm can significantly improve the scalability of NFV-based networks compared with two benchmarks.

C. Test-bed Evaluation

1) Implementation on the Platform: We implement the GFT, T-SAT and KPGD algorithms on a real test-bed. The topology of our platform is a small-scale topology Telstra from the Rocketfuel dataset [41]. The topology is composed of four main parts: a server installed with the controller's software, a set of OpenFlow enabled switches, a set of servers and some terminals. Specifically, we choose RYU [43] as the controller software running on a server with a core i7-9700k and 32GB of RAM. We build the data plane with 2 Pica8 3297 switches and 6 Open vSwitches (OVSes with version 2.8.5). Each OVS is run on a single server with a core i7-8700k processor and 32GB of RAM. Besides, there are three kinds of VNFs, including IDS, Proxy and Monitor on servers, each of which is equipped with a core i5-3470 processor and 8GB of RAM.

We adopt the Packet Generator (PktGen) [42] to generate network traffic, which is a powerful tool also used by [44] [45]. Using PktGen, requests can be generated with various sizes and patterns. Since there are 8 switches in the network, each 5-tuple flow is regarded as a request so as to generate more requests in the test-bed. In the experiments, we generate DCTCP pattern requests [42]. According to the request size distribution, the rate of 40% requests is set as 500Kbps and that of the rest requests is set as 800Kbps. We divide the differentiated services code point (DSCP) into four parts, *i.e.*, DSCP 0-3, each of which accounts for 25%. Since we implement our algorithm on a small-scale testbed, the server deployment problem can be optimally solved by the integer programming solver. Thus, we ignore the server deployment performance comparison among three algorithms here.

2) Testing Results: In the first set of experiments, we generate 30s TCP requests in the network. As shown in Fig. 9, we count the number of installed rules on each switch and determine the maximum (average) number of these rules. Our proposed KPGD solution needs to install fewer rules than other solutions. For example, it needs to install 220 and 150 rules on switch v_4 by GFT and T-SAT, respectively, while KPGD installs only about 35 rules on this switch. In other words, KPGD can reduce the maximum number of rules by about 84% and 77% compared with GFT and T-SAT, respectively.

We also conduct the traffic dynamics, which require to dynamically update rules in the second set of experiments, on the test-bed implementation. We change the requests in the network over time. If these rules are updated at a low speed, the network performance will be greatly decreased. KPGD can achieve a lower update delay compared with the other two benchmarks by Fig. 10. Because our proposed algorithm can significantly reduce the number of required rules compared with other solutions. For example, when there are 600 requests in the network, KPGD reduces the number of rules by about 75% and 80% compared with T-SAT and GFT, respectively. Accordingly, less control overhead will be required between the controllers and the switches during update procedure. For example, Fig. 11 shows that KPGD can reduce control overhead by about 76% and 77% compared with T-SAT and GFT, respectively. Lower update delay and control traffic overhead show the better scalability of KPGD compared with two benchmarks.

V. CONCLUSIONS

In this paper, we propose the incremental server deployment problem for construction of scalable NFV-based networks. We give the inapproximation performance of INSD with a ratio of $(1-\epsilon)\ln m$, for any $\epsilon > 0$ and m is the number of requests. We then design an efficient algorithm with an approximation performance of $2 \cdot H(q \cdot p)$ for INSD, where q is the number of VNF's categories and p is the maximum number of requests through a switch. The experimental results and extensive simulation results show the high efficiency of our proposed algorithm. In the future, we will study the server deployment without the pre-computed path for each request and build a moderate-size physical platform to support NFV and optimize the SFC realization.

VI. ACKNOWLEDGEMENT

This research of Liu, Xu, Zhao, Fan and Huang is partially supported by the National Science Foundation of China (NSFC) under Grants 61822210, U1709217, and 61936015; by Anhui Initiative in Quantum Information Technologies under No.AHY150300. This research of Qian is partially supported by National Science Foundation Grant 1750704.

References

[1] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, and V. Sekar, "Making middleboxes someone else's problem: network processing as a cloud service," *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 4, pp. 13–24, 2012.

[2] Z. A. Qazi, C.-C. Tu, L. Chiang, R. Miao, V. Sekar, and M. Yu, "Simple-fying middlebox policy enforcement using sdn," in *ACM SIGCOMM computer communication review*, vol. 43, no. 4. ACM, 2013, pp. 27–38.

[3] 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," *IEEE/ACM Transactions on Networking (TON)*, vol. 26, no. 4, pp. 1562–1576, 2018.

[4] H. Hawilo, A. Shami, M. Mirahmadi, and R. Asal, "Nfv: State of the art, challenges and implementation in next generation mobile networks (vepc)," *arXiv preprint arXiv:1409.4149*, 2014.

[5] T. Koponen, K. Amidon, P. Balland, M. Casado, A. Chanda, B. Fulton, I. Ganichev, J. Gross, P. Ingram, E. J. Jackson *et al.*, "Network virtualization in multi-tenant datacenters," in *NSDI*, vol. 14, 2014, pp. 203–216.

[6] S. Mehraghdam, M. Keller, and H. Karl, "Specifying and placing chains of virtual network functions," in *2014 IEEE 3rd International Conference on Cloud Networking (CloudNet)*. IEEE, 2014, pp. 7–13.

[7] R. Cohen, L. Lewin-Eytan, J. S. Naor, and D. Raz, "Near optimal placement of virtual network functions," in *2015 IEEE Conference on Computer Communications (INFOCOM)*. IEEE, 2015, pp. 1346–1354.

[8] M. Ghaznavi, A. Khan, N. Shahriar, K. Alsubhi, R. Ahmed, and R. Boutaba, "Elastic virtual network function placement," in *2015 IEEE 4th International Conference on Cloud Networking (CloudNet)*. IEEE, 2015, pp. 255–260.

[9] 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 *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*. IEEE, 2017, pp. 1–9.

[10] D. Bhamare, M. Samaka, A. Erbad, R. Jain, L. Gupta, and H. A. Chan, "Optimal virtual network function placement in multi-cloud service function chaining architecture," *Computer Communications*, vol. 102, pp. 1–16, 2017.

[11] H. Feng, J. Llorca, A. M. Tulino, D. Raz, and A. F. Molisch, "Approximation algorithms for the nfv service distribution problem," in *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*. IEEE, 2017, pp. 1–9.

[12] X. Li and C. Qian, "An nfv orchestration framework for interference-free policy enforcement," in *Proc. of IEEE ICDCS*, 2016.

[13] B. Addis, D. Belabed, M. Bouet, and S. Secci, "Virtual network functions placement and routing optimization." in *CloudNet*, 2015, pp. 171–177.

[14] Q. Zhang, Y. Xiao, F. Liu, J. C. Lui, J. Guo, and T. Wang, "Joint optimization of chain placement and request scheduling for network function virtualization," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2017, pp. 731–741.

[15] N. Rasmussen, "Strategies for deploying blade servers in existing data centers," *White Paper*, vol. 125, pp. 1–14, 2006.

[16] B. Leng, L. Huang, C. Qiao, and H. Xu, "A light-weight approach to obtaining nf state information in sdn+ nfv networks," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 2018, pp. 971–979.

[17] D. Li, P. Hong, K. Xue *et al.*, "Virtual network function placement considering resource optimization and sfc requests in cloud datacenter," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 7, pp. 1664–1677, 2018.

[18] N. Katta, O. Alipourfard, J. Rexford, and D. Walker, "Infinite cacheflow in software-defined networks," in *Proceedings of the third workshop on Hot topics in software defined networking*. ACM, 2014, pp. 175–180.

[19] FAST. (2016) Fpga based sdn swithing. [Online]. Available: https://fast-switch.github.io/

[20] X. Yu, H. Xu, D. Yao, H. Wang, and L. Huang, "Countmax: A lightweight and cooperative sketch measurement for software-defined networks," *IEEE/ACM Transactions on Networking (TON)*, vol. 26, no. 6, pp. 2774–2786, 2018.

[21] P. Wang, H. Xu, L. Huang, C. Qian, S. Wang, and Y. Sun, "Minimizing controller response time through flow redirecting in sdns," *IEEE/ACM Transactions on Networking (TON)*, vol. 26, no. 1, pp. 562–575, 2018.

[22] H. Xu, S. Chen, Q. Ma, and L. Huang, "Lightweight flow distribution for collaborative traffic measurement in software defined networks," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2019, pp. 1108–1116.

[23] L. Guo, J. Pang, and A. Walid, "Dynamic service function chaining in sdn-enabled networks with middleboxes," in *Network Protocols (ICNP), 2016 IEEE 24th International Conference on*. IEEE, 2016, pp. 1–10.

[24] V. Sekar, S. Ratnasamy, M. K. Reiter, N. Egi, and G. Shi, "The middlebox manifesto: enabling innovation in middlebox deployment," in *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*. AcM, 2011, p. 21.

[25] OVS. (2018) Open vswitch: open virtual switch. [Online]. Available: http://openvswitch.org/

[26] B. Switches. (2014). [Online]. Available: https://www.barefootnetworks.com

[27] G. Zhao, H. Xu, J. Liu, C. Qian, J. Ge, and L. Huang, "Safe-me: Scalable and flexible middlebox policy enforcement with software defined networking," in *2019 IEEE 27th International Conference on Network Protocols (ICNP)*. IEEE, 2019, pp. 1–11.

[28] A. Gushchin, A. Walid, and A. Tang, "Scalable routing in sdn-enabled networks with consolidated middleboxes," in *Proceedings of the 2015 ACM SIGCOMM Workshop on Hot Topics in Middleboxes and Network Function Virtualization*. ACM, 2015, pp. 55–60.

[29] F. Bari, S. R. Chowdhury, R. Ahmed, R. Boutaba, and O. C. M. B. Duarte, "Orchestrating virtualized network functions," *IEEE Transactions on Network and Service Management*, vol. 13, no. 4, pp. 725–739, 2016.

[30] H. Xu, J. Liu, C. Qian, H. Huang, and C. Qiao, "Reducing controller response time with hybrid routing in software defined networks," *Computer Networks*, vol. 164, p. 106891, 2019.

[31] J. Liu, L. Huang, C. Qiao, and S. Wang, "Tor-me: Reducing controller response time based on rings in software defined networks," in *2019 IEEE 11th International Conference on Communication Software and Networks (ICCSN)*. IEEE, 2019, pp. 27–33.

[32] C. Gao, X. Yao, T. Weise, and J. Li, "An efficient local search heuristic with row weighting for the unicost set covering problem," *European Journal of Operational Research*, vol. 246, no. 3, pp. 750–761, 2015.

[33] R. Raz and S. Safra, "A sub-constant error-probability low-degree test, and a sub-constant error-probability pcp characterization of np," in *STOC*, vol. 97. Citeseer, 1997, pp. 475–484.

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

[35] A. Gupta, M. Pál, R. Ravi, and A. Sinha, "What about wednesday? approximation algorithms for multistage stochastic optimization," in *Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques*. Springer, 2005, pp. 86–98.

[36] W. Shih, "A branch and bound method for the multiconstraint zero-one knapsack problem," *Journal of the Operational Research Society*, vol. 30, no. 4, pp. 369–378, 1979.

[37] Pica8. (2014) Pica8 p3297 switches. [Online]. Available: https://www.pica8.com/wp-content/uploads/pica8-datasheet-48x1gbe-p3297.pdf

[38] H. Xu, H. Huang, S. Chen, and G. Zhao, "Scalable software-defined networking through hybrid switching," in *IEEE INFOCOM*, 2017.

[39] Cbench. (2013) Controller benchmarker. [Online]. Available: https://github.com/mininet/oflops/tree/master/cbench

[40] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 4. ACM, 2008, pp. 63–74.

[41] N. Spring, R. Mahajan, and D. Wetherall, "Measuring isp topologies with rocketfuel," in *ACM SIGCOMM Computer Communication Review*, vol. 32, no. 4. ACM, 2002, pp. 133–145.

[42] W. Bai, L. Chen, K. Chen, and H. Wu, "Enabling {ECN} in multi-service multi-queue data centers," in *13th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 16)*, 2016, pp. 537–549.

[43] Ryu. (2017). [Online]. Available: https://osrg.github.io/ryu/

[44] G. Chen, Y. Lu, Y. Meng, B. Li, K. Tan, D. Pei, P. Cheng, L. L. Luo, Y. Xiong, X. Wang *et al.*, "Fast and cautious: Leveraging multi-path diversity for transport loss recovery in data centers," in *2016 {USENIX} Annual Technical Conference ({USENIX}{ATC} 16)*, 2016, pp. 29–42.

[45] B. Li, K. Tan, L. L. Luo, Y. Peng, R. Luo, N. Xu, Y. Xiong, P. Cheng, and E. Chen, "Clicknp: Highly flexible and high performance network processing with reconfigurable hardware," in *Proceedings of the 2016 ACM SIGCOMM Conference*. ACM, 2016, pp. 1–14.