# Incremental Server Deployment for Software-Defined NFV-enabled Networks

Jianchun Liu, *Student Member, IEEE,* Hongli Xu, *Member, IEEE,* Gongming Zhao, *Student Member, IEEE,* Chen Qian, *Member, IEEE,* Xingpeng Fan, Xuwei Yang, He Huang, *Member, IEEE,*

**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 $\epsilon$ 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 $2\cdot H(q\cdot p)$, 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 experimental results 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**—*Software Defined Networks, Incremental Server Deployment, Scalability, Rules, NFV.*

## 1 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 intrusion 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]. Furthermore, due to traffic dynamics, different joint optimization problems have been investigated in literatures [10] [11]. 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 servers are unavailable, and it is also challenging and time-consuming to find the right servers for placement. Moreover, with the increasing of servers, the complexity of VNF management, *e.g.*, fault diagnosis and localization, grows especially as the servers may be from different owners [12] [13]. Differently 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] [14] has studied the incremental server

- *Some earlier results of this paper were published in the Proceedings of INFOCOM 2020.*
- *J. Liu is with the School of Data Science, H. Xu, G. Zhao, X. Fan and X. Yang are with the School of Computer Science and Technology, University of Science and Technology of China, Hefei, Anhui, China, 230027, and also with Suzhou Institute for Advanced Study, University of Science and Technology of China, Suzhou, Jiangsu, China, 215123.*
  *E-mail: jsen617@mail.ustc.edu.cn, xuhongli@ustc.edu.cn, zgm1993@mail.ustc.edu.cn, fx364117@mail.ustc.edu.cn, issacyxw@mail.ustc.edu.cn.*
- *C. Qian is with the Department of Computer Science and Engineering, Jack Baskin School of Engineering, University of California Santa Cru. E-mail: cqian12@ucsc.edu*
- *H. Huang is with the School of Computer Science and Technology, Soochow University. E-mail: huangh@suda.edu.cn*

| Schemes | SFC Policy | No. of Rules | Overhead |
|---|---|---|---|
| GFT [9] | No | Many | High |
| T-SAT [14] | Yes | Many | High |
| **Our work** | Yes | Few | Low |

TABLE 1: Comparison of existing server deployment solutions and our scheme.

deployment for network function virtualization. However, these methods have two main disadvantages of network scalability. First, almost all the previous solutions, *e.g.*, [9] [14], ignore the impact of the limited Ternary Content Addressable Memory (TCAM) size on the switches. TCAMs are 400× more expensive and consume 100× more power per Mbit than the RAM-based storage on the switches [15]. 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 [15] (*e.g.*, 6K entries on HP HPE6960 switches and 4K entries on PICA8 P-5401 switches [16]). 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 [17] may require up to millions of possible rules on a switch, which certainly does not fit the TCAM size. Moreover, installing 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 [14] 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 1.

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 $\epsilon$ 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 $2 \cdot H(q\cdot p)$[1], where $q$ is the number of VNF's categories, and $p$ is the maximum number of requests through a switch. To be more practical, we extend our algorithm to address the general case, in which there is no pre-computed path for each request.
- 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 by about 5%, compared with the existing solutions.

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

| Symbol | Semantics |
|---|---|
| $V_e$ | a set of egress switches |
| $\mathcal{R}_v$ | a set of requests whose egress switch is $v \in V_e$ |
| $q$ | the number of VNF's categories, *i.e.*, $q = \lvert\mathcal{F}\rvert$ |
| $p$ | the maximum number of requests on all switches |
| $z(v)$ | 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 $r$ |
| $\theta_f$ | the processing cost per-packet of VNF $f$ |
| $x_v$ | a server is deployed on switch $v$ or not |
| $\Omega_r$ | the feasible switch set of request $r$ for deployment |
| $y_{v,t}^f$ | a rule matching egress switch $t$ and VNF $f$ is installed on switch $v$ or not |
| $z_v^f$ | the VNF $f$ is placed on the server which is connected to the switch $v$ or not. |
| $\delta_i^f$ | the requests covered by VNF $f$ on switch $v_i$ |
| $\alpha_i^f$ | rule cost of the requests covered by VNF $f$ on switch $v_i$ |
| $\beta_i^f$ | processing cost of the requests covered by VNF $f$ on switch $v_i$ |

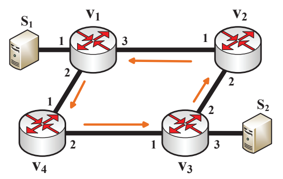
TABLE 2: Key Notations

## 2 Preliminaries and Problem Formulation

### 2.1 Network Model

An SDN is physically 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 [18] [19] 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 first $z$ switches are egress switches, denoted as $V_e = \{v_1, ..., v_z\}$, with $z < n$. 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 $\theta_f$ indicate the processing cost per packet (measured by the number of CPU cycles) for each VNF $f$. Given a set of requests



**Rule Installment under Different Schemes**

| Scheme | $V_1$ | $V_3$ |
|---|---|---|
| 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.

$\mathcal{R} = \{r_1, r_2, ..., r_m\}$ 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 $v \in V_e$.

## 2.2 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 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 $v_3$.

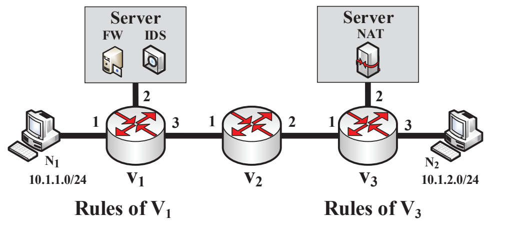
There are two different schemes of rule installment. First, the previous VNF placement solutions assume by default that the request-based rules will be installed on switches [1] [20]. Thus, there requires 12 rules for VNF processing in the network, *i.e.*, 6 rules on both switches $v_1$ and $v_3$. The rules for VNF processing on switch $v_1$ are listed in Table 3. For example, we install a rule "$src = v_2, dst = v_1, inport = 3, actions = output : 1$" for request $v_2 \rightarrow v_1$.

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 3. 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

| 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 3: Installed Rules for VNF Processing on Switch $v_1$.

with the request based scheme in this example. When there are more requests in the general scenario, our proposed scheme also can reduce the number of rules by 74% compared with the state-of-art solutions through extensive simulations in Section 4. 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.



**Rules of $V_1$**

| Match Fields | Instruction |
|---|---|
| dst=$v_3$,nflm=0x01 | output=2 |
| dst=$v_3$,nflm=0x02 | output=2 |

**Rules of $V_3$**

| Match Fields | Instruction |
|---|---|
| dst=$v_3$,nflm=0x03 | output=2 |

Fig. 2: Illustration of Rule Installment for SFC Processing on the Switches. The controller specifies that requests from subnet 10.1.1.0/24 to subnet 10.1.2.0/24 should be traversed a service function chain: Firewall-IDS-NAT for security benefits.

## 2.3 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 [1] [20]. 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 [21]. Then, it only requires 8 bits to differentiate 255 NFs. Moreover, the SFC's length is usually not more than 5 [21]. 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[2], barefoot switches[3]), adding two new fields into the packet header is easily implemented.

As shown in Fig. 2, we assume that a request from subnet 10.1.1.0/24 to subnet 10.1.2.0/24 should traverse a service func-

[2]OVS. Available: http://openvswitch.org/
[3]Barefoot Switches. Available: https://www.barefootnetworks.com

tion 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. There are 3 rules in total required to be installed on the switches (2 rules on $v_1$ and 1 rule on $v_2$) for SFC processing.

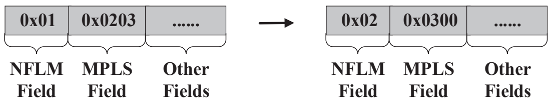


Fig. 3: Illustration of Tag Operations.

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. 3. 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. 3. More detailed information about tag operation and SFC routing can refer to our previous work [22]. Thus, the SFCs of all requests in the network can be processed properly according to the rule matching and tag operation.

## 2.4 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 [1]. 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, $\mathcal{F}_r = \{$IDS, NAT, Firewall$\}$. In fact, the order of SFC requirements will not change the processing resource cost on servers. As a result, we do not consider the VNF processing order, which will not impact the deployment result in this paper. Note that the SFC requirements can be satisfied through efficient routing algorithms [1] [23] if the VNFs have been placed on the deployed servers. For simplicity, we suppose that each VNF can work independently with others [7] [24].

Assume that the controllers have pre-computed the path for each request $r$ [25] [26], denoted by $p_r$. We will also study the problem without pre-computed paths in Section 3.3. The two resource constraints should be considered here. On one hand, we consider the rule cost for VNF processing. Let $y^f_{v,t} \in \{0,1\}$ 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 $y^{f_1}_{v_3,v_2} = 1$. 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 $z(v)$.

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

TABLE 4: Per-Packet Processing Cost of VNFs. [24]

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 [10] 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 [24], we list the number of required CPU cycles for some typical VNFs in Table 4. 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:

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

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

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 (1a) 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 (1b) 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 (1c) expresses the flow-table size (FTS) constraint for VNF processing on a switch. The fourth set of constraints (1d) tells that the total cost for VNF's processing should not exceed the server's computing capacity, where $s_v$ denote the server connected to the switch $v$. 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 [27]: let $E = \{l_1, l_2, ..., l_x\}$ be a set of $x$ 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 all elements $l \in E$ can be covered by the union of subsets in $C'$.

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 [28], Feige [29], 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 $\epsilon > 0$. □

## 3 Algorithm Design for INSD

In this section, we design an approximation algorithm for the INSD problem (Section 3.1) and give the performance analysis (Section 3.2). Besides, we extend the algorithm to solve a more general case (Section 3.3). Moreover, we give some discussions to enhance our proposed solution (Section 3.4).

### 3.1 A Knapsack-based Algorithm for INSD

In this section, we present a knapsack-based approximation algorithm, called KPGD, to solve 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 $\Gamma^f = \{\gamma^f_{v_1}, \gamma^f_{v_2}, ..., \gamma^f_{v_z}\}$, where $v_j$ is an egress switch. The rest number of requests that VNF $f$ needs to cover is denoted as $g_f$. Let $U^f_i$ be the set of requests that are uncovered by VNF $f$ on server $s_i$. The KPGD algorithm is formally described in Alg. 1. 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 which will be introduced in the next section to derive the set of chosen VNFs, denoted as $\mathcal{P}_i$, for each switch $v_i$ (Line 7-9). We choose one switch with the maximum profit for server deployment (Line 10-11). In the second step, the KPGD algorithm updates the uncovered request set (Line 12-16). For each VNF $f$ which has been chosen to place on the server connected to switch $v_i$, the requests that need to be covered by $f$ will be updated (Line 13-14). Besides, some servers have not been chosen to be deployed in the network. The rest VNFs of the SFCs need to be placed on these servers. Thus, the requests uncovered by these VNFs also should

**Algorithm 1** 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:         $U^f_i \leftarrow \Gamma^f_i$
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 algorithm, $\mathcal{P}_i \leftarrow KP(v_i)$
10:    Select a switch $v_i$ with the maximum profit $\sum_{f\in\mathcal{P}_i}\delta^f_i$ 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:        $g_f \leftarrow g_f - |U^f_i|$
15:        for each switch $v_j \in V - V'$ do
16:            $U^f_j \leftarrow U^f_j - U^f_i$
17: return $V'$
```

be updated (Line 15-16). The algorithm will terminate until each request $r$ has been covered by any VNF in $\mathcal{F}_r$.

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 [30]. 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.

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 $\Gamma = \{\gamma^{f_1}_{v_1}, ..., \gamma^{f_q}_{v_1}, ..., \gamma^{f_1}_{v_z}, ..., \gamma^{f_q}_{v_z}\}$, where $f_i \in \mathcal{F}$ and $v_j \in V_e$. $\gamma^{f_i}_{t_j}$ 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 $\Gamma^f_i$ to denote the set of requests that need to be processed by VNF $f$ in $\Gamma_i$. We denote $\mathcal{P}$ (or $\widetilde{\mathcal{P}}$) as the set of chosen (or unchosen) VNFs on this server. The profit value $\delta^f_i$ means the number of requests that can be covered by VNF $f$ on server $s_i$. Besides, we use $\alpha^f_i$ and $\beta^f_i$ 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 $\vartheta^f_i = (\alpha^f_i, \beta^f_i)$. We use $\|\vartheta^f_i\|$ to represent the norm of the vector, *i.e.*, $\|\vartheta^f_i\| = \sqrt{(\alpha^f_i)^2 + (\beta^f_i)^2}$. The variable $V^i_e \subseteq V_e$ indicates a set of egress switches of the requests which pass through switch $v_i$.

**Algorithm 2** 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 $\mathcal{P}$
```

The KP algorithm is described in Alg. 2. 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 3-6). The algorithm ranks all the unchosen VNFs in the decreasing order of the unit profit (Line 7). We then greedily choose the VNFs with the maximum unit profit under the server processing capacity and FTS constraints for VNF processing (Line 8-12). At the end of each iteration, the profit $\delta_i^f$ of each unchosen VNF $f \in \mathcal{F}$ will be computed. Then, the set $\mathcal{P}$ will be substituted by VNF $f$ if the profit $\delta_i^f$ is greater than the sum profit of all VNFs $f_j \in \mathcal{P}$, which have been deployed on this server (Line 13-15).

## 3.2 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 [30] have proved that the KP algorithm can achieve an approximation ratio of 2 for the 0-1 knapsack problem.

**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 $\varphi_i(k) \le h_i(k)$, where $\varphi_i(k)$ is the profit achieved by KPGD on switch $v_i$ in the $k$-th iteration. 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 $\delta_i(k) \le \delta_{v(k)}(k)$, where $v(k)$ denotes the chosen switch in the $k$-th iteration. 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

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

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

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

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 $\sum_{f \in \mathcal{P}_i^*} |A_i^f| \leq \sum_{f \in \mathcal{F}} |\Gamma_i^f| \leq q \cdot p$. So we have

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

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

**Theorem 6.** *The total time complexity of the KPGD algorithm is $O(n^2qm)$, where $n$ is the number of switches in the network, $q$ is the number of VNF's categories and $m$ is the size of the requests set.*

*Proof.* Because there are $n$ switches in the network, KPGD consists of $O(n)$ iterations at most. Our proposed KPGD algorithm has two main steps in each iteration. In the first step, KPGD runs the KP algorithm which has a time complexity of $O(mlogm)$ to compute the profit, where $m$ is the size of the requests set. Then, it chooses the maximum profit value with $O(n)$ time complexity. In the second step, KPGD takes $O(nqm)$ time complexity to update the requests set, where $q$ is the number of VNF's categories. Thus, the total time complexity for $n$ iterations of KPGD is $O(n^2qm)$. □

## 3.3 Extending to the General Case

In this section, we consider the more general version of INSD, denoted as INSD-G, in which each request has no pre-computed path in the network. That is, we do not know the exact path for each request before server deployment. Different from INSD, each request $r \in \mathcal{R}$ has a feasible switch set for server deployment and VNF processing in INSD-G. We introduce the concept of Stretch [31] which provides the alternative constraints for server deployment. For each request $r \in \mathcal{R}$, it can be represented by $r = (s, d, \mathcal{F}_r)$, where $s, d$ and $\mathcal{F}_r$ is the ingress switch, the egress switch and the SFC requirement, respectively. Besides, we use $l(v_a, v_b)$ to denote the length of the shortest path between $v_a$ and $v_b$, where $v_a$ and $v_b$ are any pair of switches in the network. We call $\omega$ is a feasible switch of $r$, if and only if $l(s, \omega) + l(\omega, d) \leq \mu \cdot l(s, d)$, where $\mu \geq 1$ is the stretch. Thus, the feasible switch set of request $r$ can be denoted as $\Omega_r = \{\omega \,| l(s, \omega) + l(\omega, d) \leq \mu \cdot l(s, d)\}$ in INSD-G.

We use the original network topology $G = (V, E)$ to describe the INSD-G problem, where $V$ is a set of SDN switches and $E$ is the set of links connecting these switches. Similar to INSD, we use $x_v$ to indicate whether a server will be deployed on switch $v$ or not. Let $z_v^f \in \{0, 1\}$ be an indicator variable to denote whether the VNF $f$ will be placed on the server which is connected to switch $v$ or not.If $z_v^f = 1$, the VNF $f$ will be placed on the server to accordingly process the requests that pass through the switch $v$ and need to be processed by $f$. Otherwise, no VNF will be placed on the server. We also use $y_{v,t}^f$ to denote whether a rule matching the egress switch $t$ and the VNF $f$ will be installed on switch $v$ or not. The same two constraints (*i.e.*, FTS and server computing capacity) are considered in the INSD-G problem. According to the above description, we formalize the INSD-G problem as follows:

$$\min \sum_{v \in \mathcal{V}} x_v$$
$$s.t. \begin{cases} x_v \geq z_v^f, & \forall v \in V, f \in \mathcal{F} \quad (6a) \\ z_v^f \geq y_{v,t}^f, & \forall v \in \Omega_r, t \in V_e, f \in \mathcal{F} \quad (6b) \\ \sum_{v \in \Omega_r} y_{v,t_r}^f \geq 1, & \forall r \in \mathcal{R}, f \in \mathcal{F}_r \quad (6c) \\ \sum_{t \in V_e} \sum_{f \in \mathcal{F}} y_{v,t}^f \leq c(v), & \forall v \in V \quad (6d) \\ \sum_{t \in V_e} \sum_{r \in \mathcal{R}_v} \sum_{f \in \mathcal{F}_r} y_{v,t_r}^f N(r) \theta_f \leq c(s_v), & \forall v \in V \quad (6e) \\ x_v, z_v^f, y_{v,t}^f \in \{0, 1\}, & \forall v \in V, t \in V_e, f \in \mathcal{F} \quad (6f) \end{cases}$$

The first set of inequalities (6a) denotes whether a server will be deployed on switch $v$ or not. The second set of inequalities (6b) means that the VNF $f$ should be placed on the server connected to the switch $v$ if the rule matching egress switch $t$ and VNF $f$ has been installed on switch $v$.The third set of inequalities (6c) means that each VNF $f \in \mathcal{F}_r$ should be deployed at least once on the switch $v \in M_r$, where $t_r$ denotes the egress switch of request $r$. The fourth set of inequalities (6d) expresses the FTS constraint for VNF processing on the switch $v$. The fifth set of inequalities (6e) tells that the processing cost on each server should not exceed its capacity. The objective is to minimize the number of deployed servers in the NFV-enabled networks.

In order to solve the INSD-G problem, we design the KPGD-G algorithm which is similar to KPGD. In INSD, each request $r$ in INSD has a specified path for server deployment. While in the INSD-G problem, each request $r$ will be processed on its feasible switches. In other words, we regard that there is a logical path, which consists of all feasible switches in $\Omega_r$ for request $r$ for server deployment. Thus, the INSD-G problem can be transformed into the INSD problem. Our proposed KPGD-G algorithm also consists of a group of iterations, each of which includes two main steps. In the first step, the KP algorithm is adopted to compute the total profit of each switch which has no connected server.Then we choose the switch with the maximum profit value to deploy a server. In the second step, the rest requests that are uncovered by the placed VNFs in the network should be updated. INSD-G will terminate until each request $r$ has been covered by any VNF $f \in \mathcal{F}_r$. Due to the similarity of the algorithms, we omit the pseudo code of INSD-G.

**Lemma 7.** *Let $\rho'$ and $\eta'$ be the number of servers for the INSD-G problem by KPGD-G and the optimal solution, respectively. Similar to Theorem 5, we can derive that*

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

**Theorem 8.** *By Lemma 7, we can conclude that our proposed KPGD-G algorithm can achieve $2 \cdot H(q \cdot p)$-approximation for the INSD-G problem.*

**Theorem 9.** *The total time complexity of our proposed KPGD-G* *algorithm is $O(n^2mq\sigma)$, where $\sigma = md$ is the possible networks for server deployment and $d$ is average feasible paths for each request.*

*Proof.* Different from KPGD which has a specified path for each request, KPGD-G has a feasible switches set. Assume that there are average $d$ feasible paths for each request. So, there are $\sigma$ possible networks for server deployment, where $\sigma = md$ and $m$ is the size of the requests set. Then, similar to KPGD, KPGD-G also consists of $O(n)$ iterations at most and execute two main steps to place servers. Thus, we can conclude that the total time complexity of KPGD-G is $O(n^2mq\sigma)$ □

### 3.4 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, both Network Service Header (NSH) and Segment Routing (SR) [32] can support SFC routing. Though NSH is a data plane transmission protocol, it realizes the strategy of SFC control plane and helps users create and deploy SFC dynamically. Thus, it is also a practical solution for SFC routing. Since SR only requires to store forwarding rules on some selected switches (not all switches) along the route path, it helps to reduce the usage of forwarding rules on switches. There are two types of SR architecture, IPv6 Segment Routing (SRv6) and Multiprotocol Label Switching Segment Routing (SR-MPLS) [32]. In the SRv6 architecture, an IPv6 extension headers allows including a list of segments, where the length of each segment is 128 bits, in the IPv6 packet header which may cause overload with the long segment list. In SRMLPS architecture, the list of segments is compatible with the label stack of the MPLS data plane. Since this paper focuses on the server deployment problem, we ignore the detailed implementation of SFC routing here.

3) Our proposed solution incrementally places servers using coarse-grained (*i.e.*, wildcard-based) rules, which helps to effectively reduce the TCAM cost and control overhead. Moreover, the wildcard rules can be further compressed by applying rule optimizations that delete overlapping rules and install just one to cover them. Since this paper focuses on server deployment using long-term traffic observations, and the runtime rule installment depends on the traffic dynamics, we do not consider the rule optimization in our problem.

## 4 Performance Evaluation

This section first introduces the metrics and benchmarks for performance comparison (Section 4.1). Then, we evaluate our proposed algorithm by comparing with the previous methods through large-scale simulations (Section 4.2). Finally, we implement our algorithm on the SDN platform with physical Pica8 switches[4] and Open vSwitches (OVSes), and give the testing results (Section 4.3). All our code is publicly available at github[5]

### 4.1 Performance Metrics and Benchmarks

For server deployment and VNF placement, we design the efficient algorithms and extend it to the general case. 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 utilization of a server is the CPU computing load on this server divided by its computing capacity. Then, we calculate the average utilization of all servers, as the server utilization. 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.) and average (or Avg.) 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 and average 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 testbed 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 [33]. 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[6] to test the performance of OpenFlow controllers.
4) Network failure is a common scenario in today's networks. Thus, we measure the duration from failure occurrence to failure recovery as *failure response time*. In this case, we deal with the various failures, *e.g.*, single or multiple server/link/switch failures. The tag operation in our proposed solution may exert negative effect on the packet forwarding rate. Therefore, we adopt vnStat[7] tool to measure the network

[4]Pica8. Available: https://www.pica8.com
[5]The code is available at https://github.com/lyl617/VNF_INSD.
[6]Cbench. Available: https://github.com/mininet/oflops/tree/master/cbench
[7]vnStat. Available: https://humdi.net/vnstat/

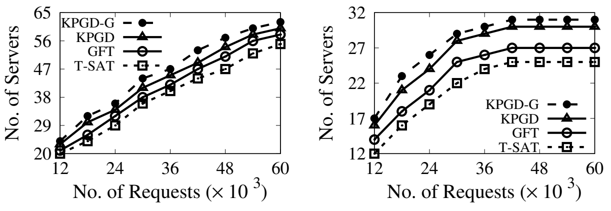


Fig. 4: No. of Servers vs. No. of Requests without FTS Constraint. *Left plot*: Topology (a); *right plot*: Topology (b).

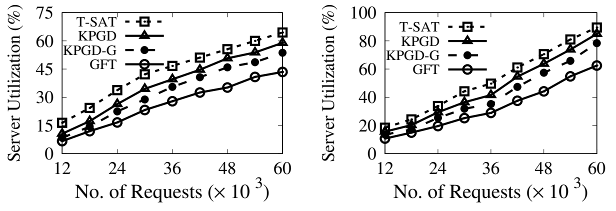


Fig. 5: Server Utilization vs. No. of Requests without FTS Constraint. *Left plot*: Topology (a); *right plot*: Topology (b).

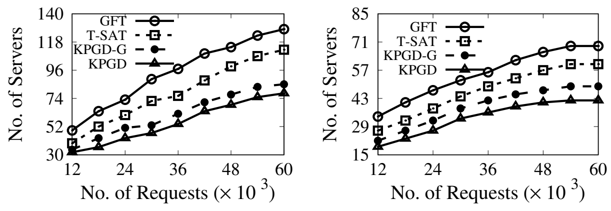


Fig. 6: No. of Servers vs. No. of Requests with the FTS Constraint. *Left plot*: Topology (a); *right plot*: Topology (b).

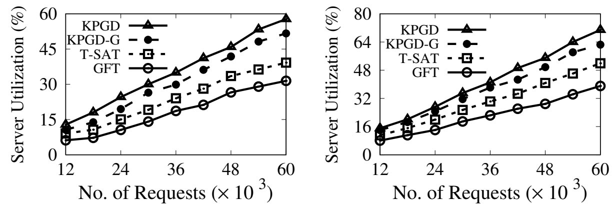


Fig. 7: Server Utilization vs. No. of Requests with the FTS Constraint. *Left plot*: Topology (a); *right plot*: Topology (b).

throughput. Besides, the server and link load of each server and link on the network are measured.

## 4.2 Simulation Evaluation

### *4.2.1 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 [14], 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.

### *4.2.2 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 [34], 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 [35]. 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 [36]. 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 with a subset of 5 types of VNFs. For example, the SFC requirement for request 1 is IDS-Monitor-Firewall, and that for request 2 may be Proxy-Monitor-Firewall-IPSec. We execute each simulation 100 times, and take the average of the numerical results.

### *4.2.3 Simulation Results*

We run five groups of simulations on two different topologies to check the effectiveness of the proposed algorithms. The first set of two simulations shows the deployment cost (*e.g.*, the number of servers) and server utilization without the FTS constraint on switches. Fig. 4 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 four solutions is very close. However, KPGD and KPGD-G 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. 11.

Fig. 5 shows the server utilization of four 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, KPGD-G and T-SAT. That is, KPGD can improve server utilization by about 10% compared to GFT.

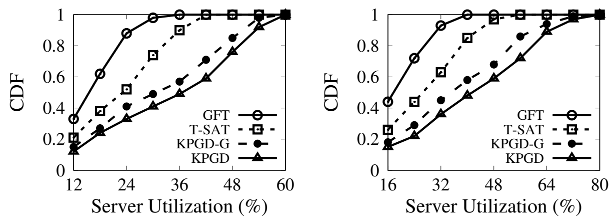


Fig. 8: CDF vs. Server Utilization. *Left plot*: Topology (a); *right plot*: Topology (b).

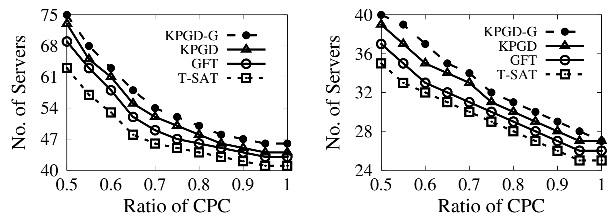


Fig. 9: No. of Servers vs. Ratio of CPC without FTS Constraint. *Left plot*: Topology (a); *right plot*: Topology (b).

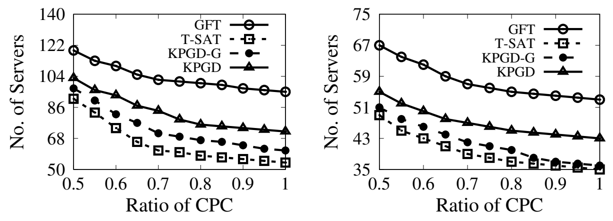


Fig. 10: No. of Servers vs. Ratio of CPC with FTS Constraint. *Left plot*: Topology (a); *right plot*: Topology (b).

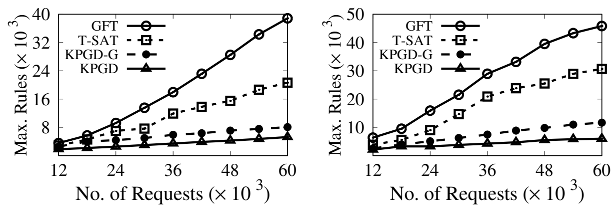


Fig. 11: Max. Rules vs. No. of Requests. *Left plot*: Topology (a); *right plot*: Topology (b).

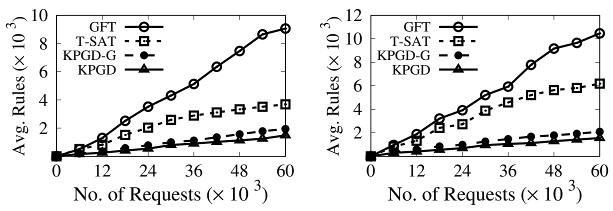


Fig. 12: Avg. Rules vs. No. of Requests. *Left plot*: Topology (a); *right plot*: Topology (b).

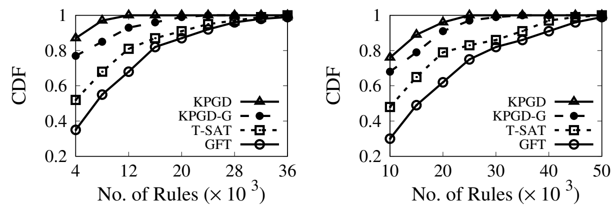


Fig. 13: CDF vs. No. of Rules. *Left plot*: Topology (a); *right plot*: Topology (b).

However, KPGD and KPGD-G 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 shows 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. 6, 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 rules on the switches for VNF processing with the request granularity, which may require a massive number of rules and violate the FTS constraint. Both KPGD and KPGD-G can effectively reduce the number of installed rule by using the wildcard. Besides, Fig. 7 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.

Fig. 8 shows the CDF of server utilization for four solutions when there are 60K requests in the network. We observe that KPGD can achieve the best performance in terms of server utilization among four solutions in both two topologies. For example, in topology (b), nearly 50% of servers will achieve utilization more than 0.45 by KPGD, while only about 5% and 15% of servers achieve the same utilization by GFT and T-SAT, respectively.

The third set of simulations shows the number of deployed servers with (or without) the FTS constraint by changing the ratio of CPU processing capacity (CPC) from 0.5 to 1 on servers. By Figs. 9-10, given a fixed number of requests (*e.g.*, 36K), the number of required servers in the network decreases for all four solutions with the increasing ratio of CPC. Our proposed KPGD and KPGD-G algorithms may deploy a little more number of servers compared with the other two solutions GFT and KPGD without the FTS constraint. For example, as shown in Fig. 9, when the ratio of CPC is 0.7 in the topology (a), KPGD needs to deploy 52 servers and KPGD-G needs to deploy 54 servers, while GFT and T-SAT need to deploy 49 and 46 servers, respectively. However, Fig. 10 shows that KPGD and KPGD-G can reduce the number of required servers compared with GFT and T-SAT with the FTS constraint. For example, given the same ratio (0.7) of CPC in the topology (a), the number of deployed servers for KPGD is 61, while KPGD-G, GFT and T-SAT need to deploy 69, 102 and 84 servers, respectively.

The fourth set of two simulations shows the TCAM consumption (*e.g.*, the maximum and average number of rules) of four solutions. As shown in Fig. 11, the maximum number of required rules increases for all solutions with the increasing number of

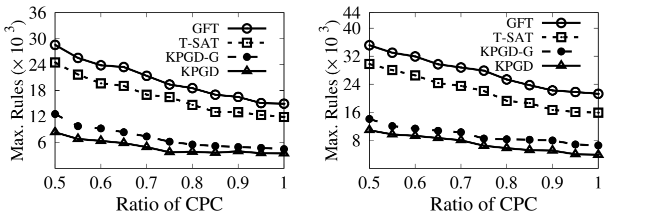


Fig. 14: Max. Rules vs. Ratio of CPC. *Left plot*: Topology (a); *right plot*: Topology (b).

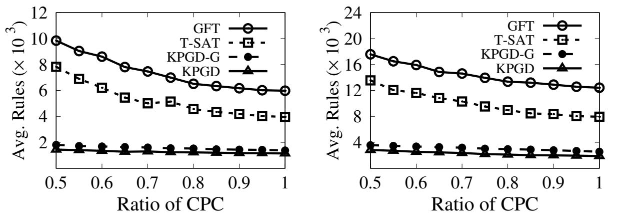


Fig. 15: Avg. Rules vs. Ratio of CPC. *Left plot*: Topology (a); *right plot*: Topology (b).

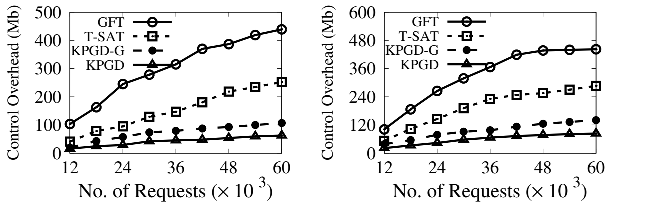


Fig. 16: Control Overhead vs. No. of Requests. *Left plot*: Topology (a); *right plot*: Topology (b).

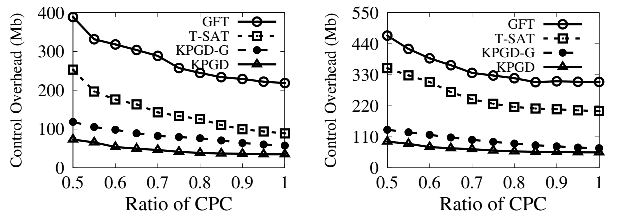


Fig. 17: Control Overhead vs. Ratio of CPC. *Left plot*: Topology (a); *right plot*: Topology (b).

requests. However, the increasing ratio of KPGD and KPGD-G is much slower than that of the other two benchmarks. In comparison, KPGD and KPGD-G require 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 GFT 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. Besides, we also observe the average number of required rules for the requests in the network. As shown in the topology (b) of Fig. 12, when there are 36K requests in the network, the average number of required rules of KPGD is about 1K, while T-SAT and GFT need about 4.6K and 5.9K rules, respectively. That is to say, our proposed KPGD solution can reduce the average number of required rules by about 77% and 82% compared with T-SAT and GFT, respectively. Fig. 13 shows the CDF of rules under a fixed number (*e.g.*, 60K) of requests. By this figure, about 90% of switches need less than 4K rules, while over 50% of switches need more than 10K rules by T-SAT and GFT. Therefore, our proposed solution can significantly reduce the TCAM consumption of all switches compared with the existing solutions.

When a server can process more SFC requests, *i.e.*, the ratio of CPC becomes larger, the number of required rules will decrease in the network. By Figs. 14-15, the maximum and average numbers of required rules decrease by changing the ratio of CPC from 0.5 to 1. KPGD and KPGD-G need to deploy both fewer rules compared with the other two solutions. For example, as shown in Fig. 14, KPGD needs to deploy about 4.8K rules if the ratio of CPC is 0.7, while T-SAT and GFT need to deploy about 20.2K and 17.6K rules, respectively, in the topology (a). Thus, KPGD can reduce the maximum number of rules by about 76% and 72% compared with T-SAT and GFT, respectively.

The last set of simulations shows the performance in terms of control traffic overhead of four solutions, including GFT, T-SAT, KPGD and KPGD-G. As shown in Fig. 16, with the number of requests increasing, GFT and T-SAT deploy more rules than KPGD and KPGD-G, leading to higher control traffic overhead. For example, when there are 36K requests in topology (a), the control overhead of KPGD and KPGD-G is about 40Mb and 75Mb, 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. Besides, we test the control overhead by changing the ratio of CPC. Fig. 17 shows that KPGD and KPGD-G also can reduce the control overhead compared with the other two solutions because of fewer number of required rules in the network which has been shown in the Figs. 14-15.

From these simulation results in Figs. 4-17, we can make the following three conclusions. First, by Figs. 4-9, our KPGD and KPGD-G solutions 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 GFT and T-SAT with the FTS constraint. Second, by Figs. 11-15, our proposed solutions can reduce the number of required rules by 90% compared with two benchmarks. Third, Figs. 16-17 show that KPGD can reduce the control overhead by about 88% and 73% compared with GFT and T-SAT, respectively. Moreover, KPGD-G achieves the similar performance of these metrics compared with KPGD. These results show that our KPGD and KPGD-G algorithms can significantly improve the scalability of NFV-based networks compared with two benchmarks.

## 4.3 Test-bed Evaluation

### *4.3.1 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

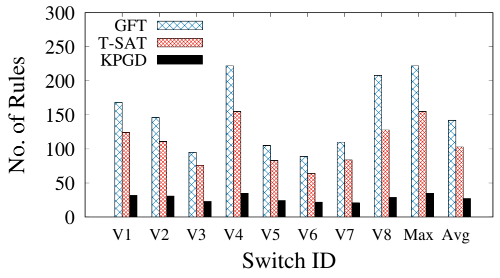


Fig. 18: No. of Rules on Each Switch in Telstra Topology

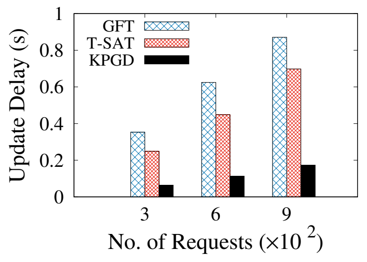


Fig. 19: Update Delay vs. No. of Requests in Telstra Topology

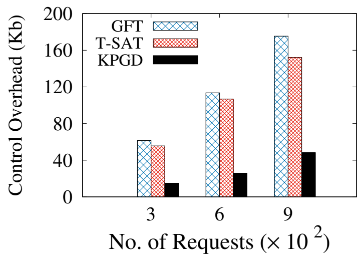


Fig. 20: Control Overhead vs. No. of Request in Telstra Topology

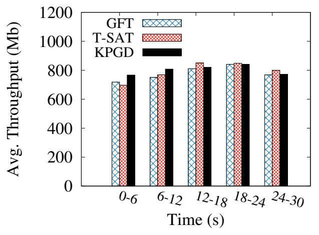


Fig. 21: Avg. Throughput vs. Time in Telstra Topology

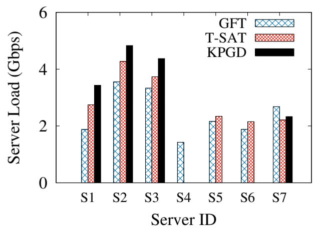


Fig. 22: Server Load of Each Server in Telstra Topology

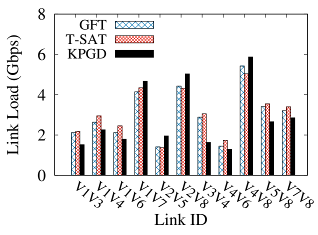


Fig. 23: Link Load of Each Link in Telstra Topology

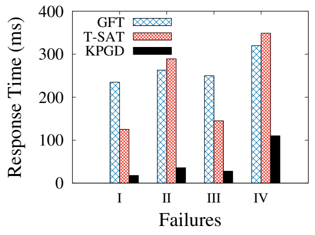


Fig. 24: Response Time vs. Failures in Telstra Topology[13]

Telstra from the Rocketfuel dataset [35]. 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[8] 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. The IDS is an open source, called *Snort*[9], which is a powerful network instrusion detection/defense system with real-time traffic analysis and network IP packet recording. *Squid*[10] is a high performance proxy cache server that supports FTP, HTTP and GOPHER protocols. *Prads*[11] is the shorter form of passive real-time asset detection system, which is a kind of Monitor.

We adopt the Packet Generator (PktGen)[12] to generate network traffic, which is a powerful tool also used by [37]. 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 [36]. 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.

### 4.3.2 Testing Results

In the first set of experiments, we generate 30s TCP requests in the network. As shown in Fig. 18, 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. 19. 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. 20 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

[8]RYU. Available: https://osrg.github.io/ryu/
[9]Snort. Available: http://www.snort.org
[10]Squid.. Available: http://www.squid-cache.org/
[11]Prads. Available: https://pradsinc.com/
[12]https://pktgen-dpdk.readthedocs.io/en/latest/
[13]I:single server failure, II:single link/switch failure, III:multi-server failures, IV:multi-link/switch failures.

overhead show the better scalability of KPGD compared with two benchmarks.

In the second set of experiments, we observe the performance of network (*e.g.*, network throughput, the load of server or link and failure response time) with the three solutions. Fig. 21 shows that KPGD achieves the similar performance of network throughput compared with GFT and T-SAT. Besides, the load of some servers and links by KPGD is a little higher than that of GFT and T-SAT in the Figs. 22-23. That means the wildcard scheme and tag operation of KPGD bring a little negative impact in the performance of network throughput and load of server/link. Moreover, Fig. 24 shows the response time of four failure scenarios:(I)single server failure,(II)single link/switch failure,(III)multi-server failures, and (IV)multi-link/switch failures. Some new rules need to be deployed on the switches in the network to re-route the requests because of failures. The time cost is mainly for the procedure of deploying rules. KPGD needs to deploy fewer number of rules compared with GFT and T-SAT which has been shown in the Fig. 18. Thus, the failure response time of KPGD is much shorter than the other two solutions. For example, when there are multiple server failures, the response time of KPGD is about 28ms, while GFT and T-SAT achieve about 250ms and 145ms, respectively.

## 5 Related Works

Recently, with the continuous development and maturing of NFV technology, it has gradually become a new way to design, deploy, and manage network services. The related VNF placement problem also have attracted plenty of attention of researchers.

Some mechanisms based on the NFV and SDN architecture have been proposed to improve the network performance and guarantee the network quality of service (QoS) in [38] [39] [40]. Guerzoni [38] introduced a Mixed Integer Programming (MIP) problem for a coordinated node and link mapping onto the underlying network infrastructure, in which VNFs or virtual machines (VMs) were mapped to a network of VM containers. The objective of the problem is to maximize the number of virtual network requests that can be optimally embedded into a given substrate.

The authors in [39] focused on network-aware VNFs placement, which alleviates mobile traffic load and reduce mobile operator cost, while neglecting the consolidation policy. However, they have no explicit solutions to solve the formulated NP-hard problems for VNFs placement. Xia [40] analyzed in detail the network traffic and its impact on VNFs placement. They formulate the problem of optimal VNFs placement in binary integer programming (BIP), and propose an alternative efficient heuristic algorithm to solve this problem.

These methods are interesting in VNFs placement problem formulation and solving them by using according algorithms. However, none of them considers the problem of service function chaining (SFC) in the NFV-enabled network.

In [6], the authors defined a model for formalizing the chaining of network functions using a context-free language. We process deployment requests and construct virtual network function graphs that can be mapped to the network. A Mixed Integer Quadratically Constrained Program (MIQCP) is described for finding the placement of the network functions and chaining them together, considering the limited network resources and requirements of the functions, such as latency, number of allocated nodes, and link utilizations. Formalization of the network function placement and chaining problem was presented by Luizelli [41]. They proposed an integer linear programmingbased optimization model to solve the placement. In this paper, they determined the locations of the VNFs, assign instances of VNFs to flow, and create paths that connect the VNFs.

VNF-OP [42] proposed a solution to determine the required number and palcement of VNFs that optimizes network operational cost and utilization, without violating SLAs. This approach includes energy cost and penalty for SLA violation as a part of network OPEX and dynamically rearranges VNFs to minimize the OPEX. A heuristic algorithm was also proposed in [43] and [44]. Li [43] used Simulated Annealing (SA) to find solutions in shorter times but simplifies the overall problem using only one type of VNF and considering rather small chains. In [44] a joint NFV resource allocation and service function chaining was proposed. The authors used a cost model to make a trade-off between service performance and network costs. Besides, they formulated a mixed-integer linear programming (MILP) to model the problem and propose a heuristic algorithm to solve it.

In the above mentioned solutions, VNFs placement and chaining probelms are solved in two main steps: VNFs are placed first and second the traffic is steered through the chains via the shortest possible path. What's more, all of these solutions process SFC routing according to the request granularity, ignoring the impact of the limited TCAM size on the switches. Our approach can combine the two steps well without violeting limitation of TCAM size.

## 6 Conclusion

In this paper, we study the incremental server deployment problem for construction of NFV-based networks. Different from the previous solutions based on request-level flows, we adopt wildcard based scheme so as to reduce the number of required rules and improve the system scalability. 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 KPGD 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. To be more practical, we extend our algorithm to address the general case, in which there is no pre-computed path for each request. The experimental results and extensive simulation results show that KPGD can reduce the required number of rules by about $87\%$ and the control overhead by about $82\%$ compared with the existing solutions, respectively. These simulation results reveal that wildcard-based rule installment can help to achieve high efficiency and significantly improve the scalability of NFV-based networks.

## Acknowledgment

This research of Liu, Xu, Zhao, Fan and Yang 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 Huang is supported by NSFC under Grant 61873177.

## REFERENCES

[1] 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.

[2] 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.

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

[11] 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.

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

[13] 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.

[14] 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.

[15] 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.

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

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

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

[30] 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.

[31] T. Lukovszki, M. Rost, and S. Schmid, "It's a match!: Near-optimal and incremental middlebox deployment," *ACM SIGCOMM Computer Communication Review*, vol. 46, no. 1, pp. 30–36, 2016.

[32] C. Filsfils, N. K. Nainar, C. Pignataro, J. C. Cardona, and P. Francois, "The segment routing architecture," in *2015 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2015, pp. 1–6.

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

[34] 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.

[35] 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.

[36] 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.

[37] 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.

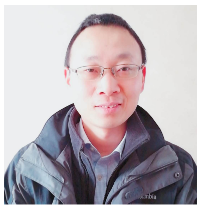
[38] R. Guerzoni, R. Trivisonno, I. Vaishnavi, Z. Despotovic, A. Hecker, S. Beker, and D. Soldani, "A novel approach to virtual networks embedding for sdn management and orchestration," in *2014 IEEE Network Operations and Management Symposium (NOMS)*. IEEE, 2014, pp. 1–7.

[39] M. Bagaa, T. Taleb, and A. Ksentini, "Service-aware network function placement for efficient traffic handling in carrier cloud," in *2014 IEEE Wireless Communications and Networking Conference (WCNC)*. IEEE, 2014, pp. 2402–2407.

[40] M. Xia, M. Shirazipour, Y. Zhang, H. Green, and A. Takacs, "Network function placement for nfv chaining in packet/optical datacenters," *Journal of Lightwave Technology*, vol. 33, no. 8, pp. 1565–1570, 2015.

[41] M. C. Luizelli, L. R. Bays, L. S. Buriol, M. P. Barcellos, and L. P. Gaspary, "Piecing together the nfv provisioning puzzle: Efficient placement and chaining of virtual network functions," in *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*. IEEE, 2015, pp. 98–106.
[42] M. F. Bari, S. R. Chowdhury, R. Ahmed, and R. Boutaba, "On orchestrating virtual network functions," in *2015 11th International Conference on Network and Service Management (CNSM)*. IEEE, 2015, pp. 50–56.
[43] X. Li and C. Qian, "The virtual network function placement problem," in *2015 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2015, pp. 69–70.
[44] L. Wang, Z. Lu, X. Wen, R. Knopp, and R. Gupta, "Joint optimization of service function chaining and resource allocation in network function virtualization," *IEEE Access*, vol. 4, pp. 8084–8094, 2016.

**Jianchun Liu** received B.S. degree in 2017 from the North China Electric Power University. He is currently a Ph.D. candidate in the School of Data Science, University of Science and Technology of China (USTC). His main research interests are software defined networks, network function virtualization, edge computing and federated learning.



**Hongli Xu** (Member, IEEE) received the B.S. degree in computer science from the University of Science and Technology of China, China, in 2002, and the Ph. D degree in computer software and theory from the University of Science and Technology of China, China, in 2007. He is a professor with the School of Computer Science and Technology, University of Science and Technology of China (USTC), China. He was awarded the Outstanding Youth Science Foundation of NSFC, in 2018. He has won the best paper award or the best paper candidate in several famous conferences. He has published more than 100 papers in famous journals and conferences, including the IEEE/ACM Transactions on Networking, IEEE Transactions on Mobile Computing, IEEE Transactions on Parallel and Distributed Systems, Infocom and ICNP, etc. He has also held more than 30 patents. His main research interest is software defined networks, edge computing and Internet of Thing.

**Gongming Zhao** received the Ph.D. degree in computer software and theory from the University of Science and Technology of China in 2020. He is now an associate professor in University of Science and Technology of China. His current research interests include software-defined networks and cloud computing.

**Chen Qian** received the B.S. degree from Nanjing University in 2006, the M.Phil. degree from The Hong Kong University of Science and Technology in 2008, and the Ph.D. degree from The University of Texas at Austin in 2013, all in computer science. He is currently an Assistant Professor with the Department of Computer Engineering, University of California at Santa Cruz. His research interests include computer networking, network security, and Internet of Things. He has authored over 60 research papers in highly competitive conferences and journals. He is a member of the ACM.

**Xingpeng Fan** received B.S. degree in 2017 from the University of Science and Technology of China . He is currently a Ph.D candidate student in Computer Science at the University of Science and Technology of China. He will receive the doctor's degree in 2022. His main research interest is software defined networks, data center network, cloud computing and edge computing.

**Xuwei Yang** received B.S. degree in network engineering from the Chang'an University in 2016. He is currently a doctor-candidate student in Computer Science at the University of Science and Technology of China. He will receive the doctor degree in 2021. His main research interest is software defined networks, network function virtualization and data center network.

**He Huang** is currently a professor in the School of Computer Science and Technology at Soochow University, P. R. China. He received his Ph.D. degree in School of Computer Science and Technology from University of Science and Technology of China (USTC), in 2011. His current research interests include traffic measurement, computer networks, and algorithmic game theory. He is a Member of both IEEE and ACM.