Multi-Resource VNF Deployment in a Heterogeneous Cloud

Jiaqi Zheng, *Member, IEEE*, Zixuan Zhang, Qiufang Ma, Xiaofeng Gao, *Member, IEEE*, Chen Tian, *Member, IEEE*, and Guihai Chen, *Senior Member, IEEE*

Abstract—The emerging paradigm of Network Function Virtualization (NFV) promises to shorten the renewal cycles of network functions and reduce the capital expenses by flexibly deploying virtualized network functions (VNFs) implementation on commodity servers. However, the required resource of each type (CPU, memory, etc.) for the running VNF should be provisioned to guarantee the performance when processing packets. This comes with different deployment cost, especially in a heterogeneous cloud consisting of a large number of network function platforms from various vendors. To optimally operate VNFs, it is necessary for the network operator to dynamically deploy VNFs in the expensive cloud infrastructures. In this article, we initiate the study of minimizing the deployment cost under multi-resource constraints in a heterogeneous cloud. We formulate multi-resource VNF deployment problem (MVDP) as an optimization program and prove its hardness. We propose an offline $(1, d+1)$-bicriteria approximation algorithm and an $(\mathcal{O}(1), \mathcal{O}(n \cdot \log n))$-competitive online algorithm to deploy VNFs in a scalable manner, where d is the number of resource types and n is the number of required VNFs. Large-scale simulations and DPDK-based OpenNetVM implementation show that our algorithms can reduce the overall cost by 34% and improve the performance in terms of multi-resource allocation.

Index Terms—Network function virtualization, heterogeneous cloud, approximation algorithm

1 Introduction

Massive expensive and dedicated hardware middleboxes such as firewalls, intrusion detection systems (IDSs), deep packet inspection (DPI) and WAN optimization are deployed in a cloud to provide various network functions, which can perform a set of specific security policies [1] and improve performance. The emerging paradigm of Network Function Virtualization (NFV) [2], [3] makes it possible to flexibly deploy virtualized network functions (VNFs) implementation on general-purpose commodity servers. NFV can accelerate network innovation for the network operators by shortening the renewal cycles, reducing capital expenses and saving energy [4]. Meanwhile, hardware acceleration techniques such as Intel DPDK [5] and SR-IOV [6] enable high-performance VNFs to process packets at line rate [7].

The current cloud computing infrastructure from Amazon, Google and Rackspace typically includes a heterogeneous collection of platforms [8] and provides a wide range of network services. To optimally operate a NFV-based heterogeneous cloud platform, an operator requires dynamically deploying various VNF instances on virtual machines or docker-based containers [9] on one physical network function platforms with limited resource capacities (CPU, memory, etc). However, the different VNF instances consume diverse types of resources when processing packets. For example, intrusion detection systems and deep packet inspection both bottleneck on CPU, while software implementation of virtual routers bottleneck on memory [10]. This also comes with amount of deployment cost, leading to unfavorable capital expenses.

Existing offline [11], [12], [13] and online [14], [15] deployment approaches only consider single resource consumption and do not respect the diversity of multiple resource consumptions for each type of VNFs, leading to unstable packet processing performance [16]. NFVnice [17] and ResQ [18] report that a VNF may become bottleneck even though only one of its available resources is limited, where the packets will be dropped and the performance could be significantly degraded. Usually these dropped packets have already been processed by the upstream VNF of a service chain and this undoubtedly results in wasted work. Furthermore, the processing capacity of multiple resources in the expensive cloud infrastructure cannot be fully utilized. In addition, the previous work assumes that the provisioning cost is a constant even if an identical VNF is deployed onto different network function platforms, which cannot be established especially in a heterogeneous cloud consisting of a large number of platforms from various vendors. Say an example in Amazon EC2 cloud platform [19], the deployment cost ranges from \$10 to \$40, depending on the network platform with different physical resources.

In this paper, we study a general problem of minimizing deployment cost with multi-resource constraints in a heterogeneous cloud. Given the VNFs for each arrived flow and their required resource vector, we aim to determine an

- *Jiaqi Zheng, Chen Tian, and Guihai Chen are with the State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210023, China. E-mail: {jzheng, tianchen, gchen}@nju.edu.cn.*
- *Zixuan Zhang and Xiaofeng Gao are with the Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, 200240, China. E-mail: zhangzx.sjtu@gmail.com, gao-xf@cs.sjtu.edu.cn.*
- *Qiufang Ma is with the State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210023, China. E-mail: mg1633053@smail.nju.edu.cn.*

Manuscript received 19 July 2020; revised 29 Oct. 2020; accepted 21 Nov. 2020. Date of publication 3 Dec. 2020; date of current version 13 Dec. 2021.
(Corresponding author: Guihai Chen.)
Recommended for acceptance by R. Wang.
Digital Object Identifier no. 10.1109/TC.2020.3042247

optimal deployment with the objective of minimizing the overall cost such that each type of consumed resource cannot beyond its capacity, where the overall cost refers to the provisioning cost for the required resource and operational cost. Our problem is fundamentally different from previous work as the heterogeneous cloud model generalizes the VNF deployment problem and makes the optimization essentially harder. Furthermore, we allow the operator to navigate a broader design space, where one may flexibly make decision by steering resource consumption of each type.

We make three novel contributions in this paper. First, we propose a general optimization framework for the multi-resource VNF deployment problem (MVDP) in a heterogeneous cloud. Generally speaking, the optimization program aims to determine VNF deployment during each epoch, where the required VNFs for each arrived flow are given, such that the total deployment cost is minimized and the resource capacity of each type cannot be overbooked.

Our second contribution is an offline and online algorithm to solve MVDP. We prove that MVDP is NP-hard, and thus focus on designing approximation algorithms. We first propose a $(1, d+1)$-bicriteria approximation algorithm and prove that it yields a near optimal solution and the resource capacity constraints can be violated by a factor of d, where d is the number of resource types. Despite the near optimal performance offered, this offline algorithm cannot be applied to the case that the order of flow requests is unknown. We further propose an $(\mathcal{O}(1), \mathcal{O}(n \cdot \log n))$-competitive online algorithm and prove that it produces a solution bounded by a constant and the resource capacity constraints are violated by a factor of $\mathcal{O}(n \cdot \log n)$, where n is the number of required VNFs for the arrived flows.

Our third contribution is a comprehensive performance evaluation of our algorithms. Large-scale simulations using synthetic network topologies show that our algorithms can reduce the total deployment cost by 34%. Meanwhile, our algorithms run faster compared to state of the art and can achieve near optimal. We also develop a prototype on the DPDK-based OpenNetVM platform [20]. Experimental results show that our solution can improve the performance in terms of multi-resource allocation.

2 Related Work

We briefly review prior art on the VNFs deployment problem. VNF-P [21] proposes a hybrid VNF deployment model to allocate physical resources, i.e., network services can be provided by a mixture of traditional dedicated hardware and VNFs. As for the fully virtualized environment, Addis *et al.* [22] formulate the VNF deployment problem as an integer programming and solve the solution using standard solver. Ghaznavi *et al.* [14] present a model to minimize the operational cost and provide elastic services. Furthermore, Cohen *et al.* [11] develop an approximation algorithm to minimize the distance cost between the clients and VNFs such that the capacity constraint of single resource should be satisfied. Feng *et al.* [13] jointly optimize the VNF deployment and routing selection to reduce the resource consumption. To respect the order among different VNFs, Ma *et al.* [23] design a heuristic algorithm to deploy interdependent VNFs, where the relations among these VNFs can be captured by a partially- or totally-ordered set. Instead of that one VNF processes all of the flows, Sang *et al.* [12] admit that one VNF can process a fraction of one flow while the others can process the rest, whose objective is minimizing the number of running VNF instances. Zhang *et al.* [15] develop an online learning-assisted algorithm to deploy VNFs in the cloud for cost minimization. NFVnice [17] and NFP [24] deploy the VNFs of a service chain onto one network function platform with multiple CPU cores. RABA [25] and REINFORCE [26] study the VNF failover mechanisms. You *et al.* [27] propose fair queueing algorithm to provide QoS guarantees for VNFs.

The work above only focuses on single resource constraint, leading to unstable packet processing performance [16]. Multi-resource generalized assignment problem is first introduced in [28]. However, its naive heuristic algorithm cannot provide provable guarantee. Guo *et al.* [29] develop a set of algorithms for vector packing constraints, while their model cannot be used to solve our problem. Our work is complementary to previous work. The novelty lies in a general optimization framework and provably algorithms of considering multi-resource constraints and various deployment cost that can well capture the heterogeneous cloud model. In addition, we provide in-depth theoretical analysis both in offline and online manner, which to our knowledge has not been done before.

3 An Optimization Framework

In this section, we introduce our optimization framework for multi-resource VNF deployment problem.

3.1 A Heterogeneous Cloud Model

Before formulating the problem, we first present our heterogeneous cloud model. Our model captures the rack-based VNF deployment in data centers [30]. We deploy the VNFs onto the heterogeneous network platforms belonging to one data center. Our model can be captured by a tuple (M, F), where M is the set of network function platforms (locations) and F represents the set of network flows. It should be noted that a flow in our model is actually an aggregate of all flows between the same source destination pair. Each flow should pass through a set of VNFs to perform a specific function. The required VNFs set for each flow are known and which VNF should be deployed onto which network function platform needs to be determined. Without loss of generality, we assume each type of resources (CPU, memory, bandwidth, etc.) at network function platform i has capacity constraint as the following equation.

$$\boldsymbol{R_i} = \left(r^1, r^2, \ldots, r^k, \ldots, r^d\right), \tag{1}$$

Where $\boldsymbol{R_i}$ is the resource capacity vector consisting of all types of resources and d is the number of resource types. The resource consumption during epoch t at each network function platform should be provisioned to guarantee the performance of the running VNFs. We use $r^k_{f,i,j}(t)$ to represent the resource consumption during epoch t if the required VNF j for flow f is deployed onto the network function platform i. Accordingly, we denote the provisioning cost $c_{f,i,j}(t)$ as the deployment cost during epoch t. To highlight our novelty and simplify the model, we only focus

TABLE 1
Key Notations in This Paper

Input	F	The set of flows f
	M	The set of network function platforms i
	VNF_f	The set of required VNFs for flow f
	$r^k_{f,i,j}(t)$	The resource consumption during epoch t if the required VNF j for flow f is deployed onto the network function platform i
	$\boldsymbol{R_i}$	The resource capacity vector at network platform i
	r^k	The k_{th} component of resource capacity vector $\boldsymbol{R}$
	d	The number of resource types
	$c_{f,i,j}(t)$	The provisioning cost during epoch t if the required VNF j for flow f is deployed onto the network function platform i
Output	$x_{f,i,j}(t)$	The zero-one integer variables that indicate that whether the required VNF j for flow f is deployed onto network function platform i during epoch t

on multi-resource constraints. This does not lose generality of the model and is complementary to previous work. For convenience, we summarize important notations in Table 1.

3.2 Problem Formulation

Based on the network model and problem definition above, we formulate MVDP, i.e., multi-resource VNF deployment problem, as an integer linear program (2). We seek to determine an optimal VNF deployment that minimizes the overall cost under multi-resource constraints. At the beginning, we discuss the meaning of the constraints one by one in detail.*The capacity of each resource must be respected.*

$$\left(\ldots, \sum_{f\in F}\sum_{j\in VNF_f} r^k_{f,i,j}(t)\cdot x_{f,i,j}(t), \ldots\right) \le \boldsymbol{R_i}, \quad \forall i\in M, \forall t\in T, k\in\{1,2,\ldots,d\}, \tag{2a}$$

The LHS of constraint (2a) characterizes that the total consumed resource of each type at network function platform i should be less than or equal to the resource capacity vector $\boldsymbol{R_i}$, where the definition of vector $\boldsymbol{R_i}$ is shown in equation (1) and the index k indicates the resource type. The VNF especially for the stateful VNF like IDS and DPI not only requires the CPU resource, but also consumes memory and other resources [31], [32]. Multi-resource constraints are a natural and necessary extension.

A VNF Must be Assigned to One Network Function Platform, Not Split.

$$\sum_{i\in M} x_{f,i,j}(t) = 1, \quad \forall f\in F, j\in VNF_f, \forall t\in T, \tag{2b}$$

Constraint (2b) captures a fact that one VNF can only be deployed onto one network function platform. That is to say, we cannot split one VNF into different parts.

The solution uses 1 to indicate a match in the mapping (0 otherwise).

$$x_{f,i,j}(t)\in\{0,1\}, \quad \forall f\in F, i\in M, j\in VNF_f, \forall t\in T. \tag{2c}$$

The zero-one integer variable $x_{f,i,j}(t)$ equals one when the required VNF j for flow f is deployed onto network function platform i during epoch t, and equals zero otherwise. This optimization variable determines that which VNF should be deployed onto which network function platform.

Combining the constraints (2a), (2b) and (2c), the formulation of multi-resource VNF deployment problem is shown in (2). The objective aims to minimize the total provisioning cost and the optimization variables $x_{f,i,j}$ determine the deployment policy.

$$\begin{aligned} \text{minimize} \quad & \sum_{f\in F}\sum_{i\in M}\sum_{j\in VNF_f} c_{f,i,j}(t)\cdot x_{f,i,j}(t) \qquad (2)\\ \text{subject to} \quad & (2a), (2b), (2c). \end{aligned}$$

3.3 Hardness Analysis

We establish the hardness of MVDP below.

Theorem 1. *MVDP is NP-hard.*

Here we only give an intuition. Given a special case of MVDP where the resource type d is equal to one, we can construct a polynomial reduction from the classic generalized assignment problem (GAP) [33] to it. The work in [33] presents a (1,2)-bicriteria approximation algorithm. Essentially, MVDP is a harder variant of GAP. We extend the model of GAP and provide in-depth theoretical analysis both in offline and online manner.

4 An Offline Approximation Algorithm

Given the VNFs deployment problem formulated as (2), we seek to solve this problem through a LP rounding technique. Our offline algorithm is not just a simple extension from [11]. The LP-rounding techniques such as classic deterministic rounding or randomized rounding cannot directly be applied into our problem with multi-resource constraints. Specifically, we novelly introduce an *average function* to comprehensively capture the multiple resource consumption for each VNF and sort the VNFs according to the results of this average function to affiliate the rounding procedure, which to our knowledge has not been done before.

We first transform it to a relaxed LP and obtain a fractional solution. Based on a constructed bipartite graph, we round it to an integer solution by minimum weight matching algortihm. Note that our rounding procedure may violate the constraint (2a) bounded by $(d+1)r^k$ while still ensuring the optimal property, which will be discussed in Theorem 3. The complete algorithm is shown in Algorithm 1, which works as the following four steps.

Step 1. Transforming to a Simplified Relaxed LP. In the offline setting, the required VNFs for all the flows are known as a prior and we can remove the index t in program (2) to simplify the formulation. Furthermore, the constraint (2b) and (2c) can be replaced by the constraint (3b) and (3c). Note that the LHS in constraint (3b) cannot be larger than one as this has been implied by the objective function.

Offline LP (MVDP-Primal):

$$\text{minimize} \quad \sum_{f\in F}\sum_{i\in M}\sum_{j\in VNF_f} c_{f,i,j}\cdot x_{f,i,j} \tag{3}$$

$$\text{subject to} \left(\ldots, \sum_{f \in F} \sum_{j \in VNF_f} r^k_{f,i,j} x_{f,i,j}, \ldots \right) \leq \boldsymbol{R_i}, \quad \forall i, k, \tag{3a}$$

$$\sum_{i \in M} x_{f,i,j} \geq 1, \quad \forall f, j, \tag{3b}$$

$$x_{f,i,j} \geq 0, \quad \forall f, i, j. \tag{3c}$$

In this way, the integer program (2) can be relaxed to the linear program (3). We are able to obtain a fractional solution $\{\tilde{x}_{f,i,j}\}$ in polynomial time by standard solver, before we round it to a feasible integer solution.

Step 2. Constructing Slots and the Bipartite Graph. The fractional solution $\{\tilde{x}_{f,i,j}\}$ indicates that we assign in total $\sum_{j=1}^{n} \tilde{x}_{f,i,j}$ VNFs on machine i. Accordingly, we allocate t_i "slots" for machine i to be assigned VNFs, where there is at most one VNF assigned on each slot, then

$$t_i = \left\lceil \sum_{f \in F} \sum_{j \in VNF_f} \tilde{x}_{f,i,j} \right\rceil \tag{4}$$

Now we can model the restriction by constructing a bipartite graph $B = (J, S, E)$, where J denotes the set of VNF nodes ($j = \{1, 2, \ldots, n\}$), and S denotes the set of machine slots.

$$S = \{(i, s) : i = 1, \ldots, m, s = 1, \ldots, t_i\}$$

As for the edge set E, we are going to explain how to add edges to the bipartite graph in *Step 3*.

Step 3. Fractional Bin Packing. Consider the slot nodes (i, s) as bins of capacity one, where $s = 1, 2, \ldots, t_i$, and the fractional solution $\tilde{x}_{f,i,j}$ as sizes of pieces of items to be packed in these bins, where $j = 1, 2, \ldots, n$. We first sort the fractional solution $\{\tilde{x}_{f,i,j}\}$, according to a pre-defined "average function" $\alpha_{i,j}$ in descending order, where

$$\alpha_{i,j} = \sum_{k=1}^{d} \frac{r^k_{f,i,j}}{r^k}$$

The average function $\alpha_{i,j}$ we introduced can be viewed as a comprehensive description of resources usage. After we sort $\tilde{x}_{f,i,j}$, we assume without loss of generality that (for each machine i) the inequation (5) hold.

$$\alpha_{i,1} \geqslant \alpha_{i,2} \geqslant \cdots \geqslant \alpha_{i,n.} \tag{5}$$

Next the bin-packing procedure begins. We can place the pieces in the bin from slot $(i, 1)$ to slot (i, t_i). The pieces with larger $\alpha_{i,j}$ values is always packed in the bin first. (According to inequation (5), the order will be $\tilde{x}_{f,i,1}, \tilde{x}_{f,i,2}, \ldots, \tilde{x}_{f,i,n}$. Note that if there is only capacity z remaining in the bin (where $\tilde{x}_{f,i,j} > z$), then we pack z of this piece j (of size $\tilde{x}_{f,i,j}$) into slot (i, s), and pack the remaining $\tilde{x}_{f,i,j} - z$ in the next slot $(i, s+1)$. In this way, the slots of machine i can never run out because we can easily infer that $t_i \geqslant \sum_{j=1}^{n} \tilde{x}_{f,i,j}$ from equation (4). Once we pack a positive fraction of job j in slot (i, s), we add an edge between j and (i, s), and set $\tilde{y}_{j,(i,s)}$ equal to that fraction at the same time; all other components of $\tilde{y}$ are set to 0.

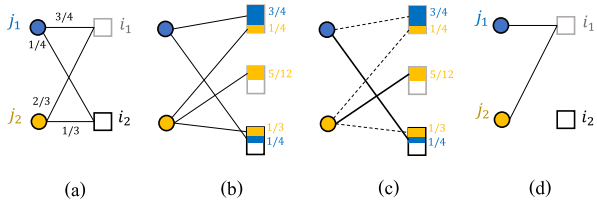


Fig. 1. An illustration of rounding the fractional solution to an integer solution.

Step 4. Rounding. When the bin-packing procedure is done, we obtain a fractional complete matching $\{\tilde{y}_{j,(i,s)}\}$ between VNFs $j = 1, \ldots, n$ and slots $S = \{(i, s) : i = 1, \ldots, m, \ s = 1, \ldots, t_i\}$ which satisfies the following constraints.

$$\sum_{\forall (i,s) \in S} \tilde{y}_{j,(i,s)} = 1, \qquad \forall j \in J$$

$$\sum_{j=1}^{n} \tilde{y}_{j,(i,s)} \leqslant 1, \qquad \forall (i, s) \in S$$

$$0 \leqslant \tilde{y}_{j,(i,s)} \leqslant 1. \ \ \forall j \in J, \ \forall (i, s) \in S.$$

Obviously, for each edge $e \in E$, which connects j and (i, s), it has a weight of $c_{i,j}$. Now the problem is transformed to determining a minimum weight matching $\{y_{j,(i,s)}\}$ between J and S, where

$$y_{j,(i,s)} \in \{0, 1\}.$$

We use Kuhn-Munkres algorithm [34] to complete the matching procedure. We can allocate j on machine i if and only if $y_{j,(i,s)}$ is equal to one. Eventually we obtain the optimal integer solution $x_{i,j}$. The complete algorithm is shown in Algorithm 1.

We provide a toy example here as an illustration of our bin-packing and LP rounding procedure defined in Step 2 to Step 4. We consider allocating two jobs j_1 and j_2 onto two machines i_1 and i_2. At Step 2, as shown in Fig 1a, suppose that we solve a relaxed LP and obtain the fractional solution $(\tilde{x}_{1,1}, \tilde{x}_{1,2}) = (\frac{3}{4}, \frac{3}{4})$ and $(\tilde{x}_{2,1}, \tilde{x}_{2,2}) = (\frac{2}{3}, \frac{1}{3})$, we first construct the number of slots according to equation (4), thus we have $t_1 = \lceil \frac{3}{4} + \frac{2}{3} \rceil = 2$ and $t_2 = \lceil \frac{1}{4} + \frac{1}{3} \rceil = 1$, and we construct 2 slots for i_1 and 1 slot for i_2 respectively. At Step 3, as shown in Fig. 1b, we conduct a bin-packing procedure where each fractional solution is packed into a slot (with capacity 1) in an order according to our newly defined average function $\alpha_{i,j}$. Here we suppose $\alpha_{1,1} > \alpha_{1,2}$ and $\alpha_{2,1} > \alpha_{2,2}$ for illustration, so j_1 is first packed into i_1 while j_2 is first packed into i_2. Note that if the remaining capacity is not enough for accommodating the next element, we cut the next element into two pieces to first fill up the previous slot and then pack the remaining one to the next slot. (For example, in Fig 1b, the fractional solution $\tilde{x}_{2,1} = 2/3$ is divided into $\frac{1}{4}$ and $\frac{5}{12}$ and packed into two different slots respectively.) Each time we packing part of the fractional solution into a slot, we add an edge between the jobs and the corresponding slot with an edge weight of the allocation cost $c_{i,j}$. Finally, we can get a bipartite graph between each VNF instance and slot. At Step 4, after we obtain the weighted bipartite graph as shown in Fig. 1c, we use Kunh-Munkres algorithm to obtain a minimum weight complete matching in this bipartite graph, that is, each VNF instance is mapped to a certain slot. Finally we

map each VNF instance to the machine i where the slot is located to and get our integer solution for the problem.

Now we analyze the time complexity of Algorithm 1.

Algorithm 1. A Bicriteria Approximation Algorithm

Input: The set of network function platforms M; the set of flows F and the set of required VNFs VNF_f for each flow f; the number of resource types d; the capacity r^k of k_{th} resource.
Output: The integer solution $\{x_{f,i,j}\}$
Obtain an optimal fractional solution $\tilde{x}_{f,i,j}$ to the relaxed LP of (3)
for $\forall\ i \in M$ **do**
  Calculate the number of slots t_i for machine i
Build a bipartite graph $G = (J, S, E)$, where J denotes the set of VNFs and S denotes the set of slots
Initialize $E = \varnothing$
for $\forall\ f \in F$ **do**
  for $\forall\ j \in VNF_f$ **do**
    Sort the VNFs according to $\alpha_{i,j}$ in descending order
    Let j' denote the index of sorted VNFs
    Let $z = 1$ denote the capacity of slots
    Let $s = 1$ denote the index of slots on machine i
    Initialize $\tilde{y}_{j,(i,s)} = 0$ to denote the allocation results
    for $\forall\ j' \in VNF_f$ **do**
      if $\tilde{x}_{f,i,j} \leqslant z$ **then**
        Add an edge from j' to (i, s) weighted $c_{i,j}$
        $\tilde{y}_{j,(i,s)} = \tilde{x}_{f,i,j}$
        $z = z - \tilde{x}_{f,i,j}$
      if $\tilde{x}_{f,i,j} > z$ **then**
        Add an edge from j' to (i, s) weighted $c_{i,j}$
        $\tilde{y}_{j,(i,s)} = z$
        $s = s + 1$
        Add an edge from j' to (i, s) weighted $c_{i,j}$
        $\tilde{y}_{j,(i,s)} = \tilde{x}_{f,i,j} - z$
        $z = 1 - \tilde{y}_{j,(i,s)}$
Find a minimum weight matching $y_{j,(i,s)}$ in bipartite graph $G = (J, S, E)$
if $y_{j,(i,s)} = 1$ **then**
  $x_{f,i,j} = 1$

Theorem 2. *The time complexity of Algorithm 1 is $\mathcal{O}(m^{3.5}n^{3.5}k^2)$, where m, n, and k denote the number of machines, the number of VNFs and the number of resource types respectively.*

Proof. Algorithm 1 consists of the following four steps. In *Step 1*, we solve a simplified linear program where we have $m \cdot n$ variables and k constraints. The time complexity is $\mathcal{O}((m \cdot n)^{3.5} \cdot k^2)$ [35].

In *Step 2*, we first calculate the number of slots for each machine and then construct a bipartite graph between n VNFs and m machines. For each machine i, we calculate the summation of $\widetilde{x}_{f,i,j}$. Therefore, the time complexity in this step is

$$T_2 = \mathcal{O}(m \cdot n).$$

In *Step 3*, we execute a fractional bin-packing procedure. On each machine i, we calculate the average functions $\alpha_{i,j}$ before sorting and packing the $\widetilde{x}_{f,i,j}$s into the slots. We utilize quick sort with $\mathcal{O}(n \cdot \log n)$ in this step. The total time complexity in this step is

$$T_3 = m \cdot (\mathcal{O}(n \cdot k) + \mathcal{O}(n \cdot \log n) + \mathcal{O}(n)) \leqslant \mathcal{O}(mnk + mn \log n).$$

In *Step 4*, we implement Kuhn-Munkres algorithm [34] to find a minimum weight matching between VNFs and slots. The number of VNFs is m and the total number of slots is no more than $m \cdot n$. Therefore, utilizing the stack techniques when updating $\varphi(j)$ and $\varphi(i, s)$, the total time complexity is $T_4 \leqslant \mathcal{O}(m^2n^3)$. Therefore, the total time complexity of Algorithm 1 is bounded by

$$T = T_1 + T_2 + T_3 + T_4 \leqslant \mathcal{O}(m^{3.5}n^{3.5}k^2),$$

where m, n, and k denote the number of machines, the number of VNFs and the number of resource types respectively. □

Now we begin to analyze the performance of our algorithm.

Theorem 3. *Algorithm 1 is a $(1, d+1)$-bicriteria approximation algorithm, which outputs an integer solution $\{x_{f,i,j}\}$ that satisfies the following two conditions:*

$$(R1): \sum_{i\in M}\sum_{f\in F}\sum_{j\in VNF_f} c_{f,i,j} \cdot x_{f,i,j} \leqslant \sum_{i\in M}\sum_{f\in F}\sum_{j\in VNF_f} c_{f,i,j} \cdot \tilde{x}_{f,i,j}$$

$$(R2): \sum_{f\in F}\sum_{j\in VNF_f} r^k_{f,i,j} x_{f,i,j} \leqslant (d+1) r^k.\ \forall i \in M,\ \forall k \in [1, d]$$

where $\{\tilde{x}_{f,i,j}\}$ is the fractional solution of the relaxed LP of program (3).

Proof. We first prove the formula (R1). From the definition of $y_{j,(i,s)}$ and $\tilde{y}_{j,(i,s)}$ in our algorithm, we obtain

$$\sum_{i\in M}\sum_{f\in F}\sum_{j\in VNF_f} c_{f,i,j} \cdot x_{f,i,j} = \sum_{f\in F}\sum_{j\in VNF_f}\sum_{(i,s)\in S} c_{f,i,j} \cdot y_{j,(i,s)}$$

$$\sum_{i\in M}\sum_{f\in F}\sum_{j\in VNF_f} c_{f,i,j} \cdot \tilde{x}_{f,i,j} = \sum_{f\in F}\sum_{j\in VNF_f}\sum_{(i,s)\in S} c_{f,i,j} \cdot \tilde{y}_{j,(i,s)}$$

Based on the two equations above, we only need to prove

$$\sum_{f\in F}\sum_{j\in VNF_f}\sum_{(i,s)\in S} c_{f,i,j} y_{j,(i,s)} \leqslant \sum_{f\in F}\sum_{j\in VNF_f}\sum_{(i,s)\in S} c_{f,i,j} \tilde{y}_{j,(i,s)} \tag{6}$$

We are ready to show that the equation (6) can be established from the Kuhn-Munkres algorithm we used to determine the minimum weight matching. While implementing Kuhn-Munkres algorithm, we define *potential*[1] for each node in the bipartite graph and we use $\varphi(j)$ and $\varphi(i, s)$ to represent such *potential* for each pair of nodes in vertex set J and S, where

$$\varphi(j) + \varphi(i, s) \leqslant c_{f,i,j.}$$

Besides, we introduce *tight edges* to denote the edges that only contains the edges $\langle j, (i, s)\rangle$ and satisfies the inequation $\varphi(j) + \varphi(i, s) = c_{i,j}$. Next we change the value of $\varphi(j)$ and $\varphi(i, s)$ repeatedly until we determine a complete matching among the set of tight edges. We can observe that for each complete matching $y_{j,(i,s)}$,

1. A detailed explanation of Kuhn-Munkres algorithm can be found in [34].

$$\sum_{f\in F}\sum_{j\in VNF_f}\sum_{(i,s)\in S} c_{f,i,j}\cdot y_{j,(i,s)} = \sum_{j\in J,(i,s)\in S}\varphi(j)+\varphi(i,s)$$

And for arbitrary fractional matching $\tilde{y}_{j,(i,s)}$,

$$\sum_{f\in F}\sum_{j\in VNF_f}\sum_{(i,s)\in S} c_{f,i,j}\cdot \tilde{y}_{j,(i,s)} \geqslant \sum_{j\in J,(i,s)\in S}\varphi(j)+\varphi(i,s)$$

Hence, the equation (6) can be established, which concludes the proof that the condition (R1) holds.

In order to prove the condition (R2), we define $r^k_{max}(i,s)$ as the maximum resource requirement on slot (i,s), and $j_0(i,s)$ as the corresponding j when $r^k_{i,j}$ reaches the maximum. Accordingly, we get

$$\begin{aligned}\sum_{f\in F}\sum_{j=1}^{|VNF_f|}\tilde{x}_{f,i,j}r^k_{f,i,j} &\leqslant \sum_{s=1}^{t_i} r^k_{max}(i,s)\\ &= r^k_{max}(i,1)+\sum_{s=2}^{t_i} r^k_{max}(i,s),\end{aligned} \tag{7}$$

We divide the sum of $r^k_{max}(i,s)$ in equation (7) into two parts. Obviously, the inequation $r^k_{max}(i,1)\leqslant r_k$ in the first part holds and we only need to prove the inequation in the second part.

$$\sum_{s=2}^{t_i} r^k_{max}(i,s)\leqslant d\cdot r^k. \tag{8}$$

Consider $r^k_{f,i,j} > 0,\ \forall i,j,k$, we have

$$\begin{aligned}r^k_{max}(i,s) &= r^k\cdot\frac{r^k_{max}(i,s)}{r^k} = r^k\cdot\frac{r^k_{i,j_0}}{r^k}\\ &\leqslant r^k\cdot\sum_{t=1}^{k}\frac{r^k_{i,j_0}}{r^k} = r^k\alpha_{i,j_0(i,s)}.\end{aligned}$$

We can conclude that the equation $\sum_j \tilde{y}_{j,(i,s)}=1$ holds, where $s=1,2,\ldots,t_i-1$. Since we only start the next bin when we have completely filled the previous one, we can view the equation $\sum_j r^k_{i,j}\tilde{y}_{j,(i,s)}=1$ as a weighted average of the relevant $r^k_{i,j}$ values. According to our sorting results from equation (5) and the bin-packing rules in *Step 3*, we get $\alpha_{i,j_0(i,s)}\leqslant\sum_{j=1}^{n}\alpha_{i,j}\tilde{y}_{j,(i,s)}$

Accordingly, we have

$$\begin{aligned}\sum_{s=2}^{t_i} r^k_{max}(i,s) &\leqslant r^k\sum_{s=2}^{t_i}\alpha_{i,j_0(i,s)} \leqslant r^k\sum_{s=2}^{t_i}\sum_{f\in F}\sum_{j=1}^{|VNF_f|}\alpha_{i,j}\tilde{y}_{j,(i,s-1)}\\ &= r^k\sum_{s=1}^{t_i-1}\sum_{f\in F}\sum_{j=1}^{|VNF_f|}\alpha_{i,j}\tilde{y}_{j,(i,s)} \leqslant r^k\sum_{s=1}^{t_i}\sum_{f\in F}\sum_{j=1}^{|VNF_f|}\alpha_{i,j}\tilde{y}_{j,(i,s)}.\end{aligned}$$

Since the equation $\tilde{x}_{f,i,j}=\sum_s\tilde{y}_{j,(i,s)}$ can be established, by interchanging the order of summations, we can obtain $\sum_{s=2}^{t_i} r^k_{max}(i,s)\leqslant r^k\sum_{f\in F}\sum_{j=1}^{|VNF_f|}\alpha_{i,j}\tilde{x}_{f,i,j}$. According to the constraints (2a), we can sum up the terms from $k=1$ to d, $\sum_{j=1}^{|VNF_f|}\alpha_{i,j}\tilde{x}_{f,i,j}\leqslant d$. Accordingly, we obtain $\sum_{s=2}^{t_i} r^k_{max}(i,s)\leqslant d\cdot r^k$, which implies that inequation (8) holds and concludes the proof. □

5 An Online Algorithm

In this section, we design an online algorithm to deploy the required VNFs for each arrived flow. In our online MVDP problem, the flows enter the network unpredictably. Accordingly, the deployment decision (i.e., which VNF needs to be deployed onto which network platform) has to be made in an online manner. Based on the primal-dual paradigm [36] and weight update approach [37], we design our online VNF deployment algorithm. Furthermore, we analyze the lower bound and the competitive ratio.

We first formulate the dual of primal program (3). For each required VNF of the flow f, we introduce dual variables $z_{f,j}$ and for each type of resource k on the network platform i, we introduce dual variables y^k_i.

Offline LP (MVDP-Dual):

$$\text{maximize}\quad \sum_{f\in F}\sum_{j\in VNF_f} z_{f,j} - \sum_{i\in M}\sum_{k=1}^{d} r^k\cdot y^k_i \tag{9}$$

$$\text{subject to}\quad z_{f,j}\leq c_{f,i,j}+\sum_{k=1}^{d} y^k_i\cdot r^k_{f,i,j} \quad \forall f\in F, i\in M, j\in VNF_f, \tag{9a}$$

$$y^k_i\geq 0,\quad \forall i,\forall k\in\{1,2,\ldots,d\}, \tag{9b}$$

$$z_{f,j}\geq 0,\quad \forall f\in F, j\in VNF_f. \tag{9c}$$

In the dual formulation, the variables y^k_i capture the shadow prices for different resources. Specifically, the term $y^k_i\cdot r^k_{f,i,j}$ in constraint (9a) indicates the weighted price for the kth resources. And the RHS of constraint (9a) represents the revenues for different VNF deployment. The objective aims to maximize the profit, i.e., the difference between the revenues of the utilized resources and their cost.

The complete online algorithm is shown in Algorithm 2. We first initialize all variables (line 1). The parameter a is selected so that the initial value of y^k_i is bounded (lines 2-5). For each VNF j of the arrived flow f, we determine the network platform i^* with the maximum profit in program (9) (lines 8-9). Once the network platform i^* is determined, we update the dual variables $z_{f,j}$ and y^k_i respectively (lines 10-11). This process essentially tries to maximize the profit over the time horizon.

Before analyzing the performance of our online algorithm, we briefly discuss the competitive ratio for multi-resource VNF deployment problem defined in Section 3. Let OPT and C denote the deployment cost obtained from offline optimal solution and an online solution respectively. Our objective aims to minimize the total cost and thus the competitive ratio $\alpha=\frac{C}{OPT}\geq 1$. We advocate to design an online algorithm with the competitive ratio α close to one. Furthermore, the consumption for each type of resource k can be less than or equal to $\beta\cdot r^k$, where r^k is the resource capacity.

Theorem 4. *If there exists a (α,β)-competitive online algorithm for MVDP (α is a positive constant), we can conclude that $\beta=\Omega(n\log n)$.*

This indicates that if an online algorithm for MVDP has a constant competitive ratio, the logarithmic violation is inevitable. This bound is asymptotically achieved by Algorithm 2, which will be discussed soon in Theorem 5. We first show the dual feasibility to facilitate our competitive analysis.

Algorithm 2. Online Algorithm for MVDP

Input: The set of network function platforms M; the set of required VNFs VNF_f for each arrived flow f; the number of resource types d; the capacity r^k of k_{th} resource.

Output: The integer solution $\{x_{f,i,j}\}$

1 Initialize $x_{f,i,j}=0$, $z_{f,j}=0$
2 $c^+_{f,i,j}=\max\{c_{f,i,j}|\forall f,i\in M,j\in VNF_f\}$
3 $c^-_{f,i,j}=\min\{c_{f,i,j}|\forall f,i\in M,j\in VNF_f\}$
4 $r^+=\max\left\{r^k_{f,i,j}|\forall f,i,j,\forall k\in\{1,2,\ldots,d\}\right\}$
5 $y^k_i=\frac{a\cdot c^+_{f,i,j}}{n}$, where $a\le\frac{c^-_{f,i,j}}{c^+_{f,i,j}}\cdot\frac{d}{r^+}$
6 **for** *each arrival of flow* f **do**
7 **for** *each* $j\in VNF_f$ **do**
8 $i^*=\arg\min_i\left\{c_{f,i,j}+\sum_k y^k_i\cdot r^k_{f,i,j}\right\}$
9 $x_{f,i^*,j}=1$
10 $z_{f,j}=c_{f,i^*,j}+\sum_k y^k_{i^*}\cdot r^k_{f,i^*,j}$
11 $y^k_{i^*}=y^k_{i^*}\cdot e^{\frac{\sqrt{1+4\frac{r^k_{f,i^*,j}}{r^k}}-1}{2}}$, where e is Euler's Number.

Lemma 1. *The dual variables update in Algorithm 2 (lines 10-11) can preserve the feasibility of constraints in program (9).*

Proof. The variables $z_{f,j}$ and y^k_i are initially nonnegative and the dual feasibility can be satisfied at the beginning. We only need to prove that the variables update procedure in Algorithm 2 can maintain the dual feasibility. For variables y^k_i, the update (line 11) can only increase their values. And for variables $z_{f,j}$, the network platform i^* is selected with minimum value (line 8) such that the constraint (9a) can be established (line 10). Further updates of y^k_i only makes the RHS in constraint (9a) larger, always preserving the feasibility in this constraint. Hence, the constraints (9a), (9b) and (9c) can be always satisfied in program (9). □

Based on the analysis above, we have the following theorem.

Theorem 5. *Algorithm 2 is* $(\mathcal{O}(1),\mathcal{O}(n\cdot\log n))$*-competitive, where* n *is the number of required VNFs for the arrived flows.*

Proof. We first prove that Algorithm 2 can produce a solution with cost bounded by a constant $\frac{1}{1-a}\cdot OPT$. During each iteration in the algorithm, the dual feasibility is ensured by Lemma 1 and the variation of objective value can be derived by the inequation below.

$$\begin{aligned}\Delta_{dual}&=\sum_f\sum_j\Delta z_{f,j}-\sum_{i^*}\sum_k r^k\cdot\Delta y^k_{i^*}\\&=c_{f,i^*,j}+\sum_{i^*}\sum_k y^k_{i^*}r^k_{f,i^*,j}\\&\quad-\sum_{i^*}\sum_k r^k\left(y^k_{i^*}e^{\frac{\sqrt{1+4\frac{r^k_{f,i^*,j}}{r^k}}-1}{2}}-y^k_{i^*}\right)\\&\ge c_{f,i^*,j}+\sum_{i^*}\sum_k y^k_{i^*}r^k_{f,i^*,j}-\\&\quad\sum_{i^*}\sum_k r^k\left[y^k_{i^*}\left(\frac{r^k_{f,i^*,j}}{r^k}+1\right)-y^k_{i^*}\right]\\&=c_{f,i^*,j},\end{aligned}$$

Hence, the deployment cost obtained from Algorithm 2 is $\sum_{f\in F}\sum_{j\in VNF_f}c_{f,i^*,j}$. Further, due to the weak duality theorem [38], the objective value of program (9) can provide a lower bound for OPT, i.e.,

$$\sum_{f\in F}\sum_{j\in VNF_f}c_{f,i^*,j}-\sum_{f\in F}\sum_{j\in VNF_f}a\cdot c^+_{f,i^*,j}\le OPT$$

Accordingly, the competitive ratio can be derived as the following.

$$\begin{aligned}&\frac{\sum_{f\in F}\sum_{j\in VNF_f}c_{f,i^*,j}}{OPT}\\&\le\frac{\sum_{f\in F}\sum_{j\in VNF_f}c_{f,i^*,j}}{\sum_{f\in F}\sum_{j\in VNF_f}c_{f,i^*,j}-\sum_{f\in F}\sum_{j\in VNF_f}a\cdot c^+_{f,i^*,j}}\\&\le\frac{\sum_{f\in F}\sum_{j\in VNF_f}c_{f,i^*,j}}{\sum_{f\in F}\sum_{j\in VNF_f}c_{f,i^*,j}-a\sum_{f\in F}\sum_{j\in VNF_f}c_{f,i^*,j}}\\&=\frac{1}{1-a}.\end{aligned}$$

Now we begin to prove that the constraint (2a) is violated by a factor of $\mathcal{O}(n\cdot\log\frac{n}{a})$. We can obtain the lower bound of variables y^k_i by applying the inequation $\frac{\sqrt{1+4x}-1}{2}\ge\frac{x}{2}$ and the definition of variable a in Algorithm 2. The inequation $\frac{\sqrt{1+4x}-1}{2}\ge\frac{x}{2}$ can be established if and only if $x\in[0,4]$. Note that the inequation $r^k_{f,i,j}\le r^k$ is reasonable as the resource consumption for one specific flow is always less than the resource capacity in practice, which indicates the ratio $\frac{r^k_{f,i,j}}{r^k}\in[0,1]\subset[0,4]$.

$$\begin{aligned}y^k_i&=\frac{a\cdot c^+_{f,i,j}}{n}\cdot e^{\sum_{f\in F}\sum_{j\in VNF_f}\frac{\sqrt{1+4\frac{r^k_{f,i,j}}{r^k}}-1}{2}}\\&\ge\frac{a\cdot c^+_{f,i,j}}{n}\cdot e^{\sum_{f\in F}\sum_{j\in VNF_f}\frac{1}{2}\frac{r^k_{f,i,j}}{r^k}}.\end{aligned}\tag{10}$$

Before analyzing the upper bound of variables y^k, we first introduce the definition of maximum utilization l^*_k for resource type k.

$$l^*_k=\max\left\{\frac{r^k_{f,i,j}}{r^k}\middle|\forall f\in F,i\in M,j\in VNF_f\right\}$$

We obtain the upper bound of variables y^k_i by applying the inequation $e^{\frac{\sqrt{1+4x}-1}{2}}\le x+1$,

$$\begin{aligned}y^k_i&=\frac{a\cdot c^+_{f,i,j}}{n}\cdot e^{\sum_{f\in F}\sum_{j\in VNF_f}\frac{\sqrt{1+4\frac{r^k_{f,i,j}}{r^k}}-1}{2}}\\&\le\frac{c^-_{f,i,j}}{c^+_{f,i,j}}\cdot\frac{d}{r^+}\cdot\frac{c^+_{f,i,j}}{n}\cdot e^{\sum_{f\in F}\sum_{j\in VNF_f}(1+l^*_k)}\\&\le\frac{c^-_{f,i,j}}{r^+}\cdot\frac{d}{n}\cdot(1+l^*_k)^n.\end{aligned}$$

Since the number of resource types d is always less than that of required VNFs n for each flow, i.e., $\frac{d}{n}\le1$, we have

$$y^k_i\le\frac{c^-_{f,i,j}}{r^+}\cdot(1+l^*_k)^n.\tag{11}$$

TABLE 2
Maximum CPU and Memory Consumption Comparison

Packet arrival rate (%)		60	80	100
OPT	CPU(%)	9.5097	9.5871	9.6351
	Mem.(KB)	77172	77024	77872
MVDP-Offline	CPU(%)	9.5516	9.6109	9.6381
	Mem.(KB)	77404	77420	77965
MVDP-Online	CPU(%)	9.5633	10.0328	10.5763
	Mem.(KB)	78091	79503	80939

Combining inequation (10) and (11), we can obtain that the resource utilization can be captured by the following inequation.

$$\sum_{f\in F}\sum_{j\in VNF_f}\frac{r^k_{f,i,j}}{r^k} \le 2n\cdot\ln\frac{n}{a}\cdot\frac{c^-_{f,i,j}(1+l^*_k)}{c^+_{f,i,j}\cdot r^+} \tag{12}$$

We conclude our proof by omitting the constant coefficient in inequation (12). □

6 Experimental Evaluation

We evaluate our scheduling algorithm using both prototype implementation and large-scale simulation.

Benchmark Schemes. We compare the following schemes with our algorithm.

- *MVDP-Offline*. Our offline approximation algorithm for MVDP in Algorithm 1.
- *MVDP-Online*. Our online algorithm for MVDP in Algorithm 2.
- *Greedy*. The heuristic algorithm proposed in [28].
- *OPT*. The optimal solution of the offline MVDP integer program (2) obtained using branch and bound. To perform an "apples to apples" comparison with *MVDP-Offline*, we enlarge the resource capacity to $d\cdot \boldsymbol{R_i}$ in constraint (3a).

The trace used in our evaluation is generated from [39]. We use heterogeneous VNFs with different processing complexity [17] and ensure that each flow can traverse a predefined VNFs. The length of VNF chains varies from 2 to 10. Each data point is an average of at least 30 runs.

6.1 Implementation and Testbed Emulations

Implementation. We develop a prototype of our algorithms on the DPDK-based OpenNetVM platform [20], where the polling mechanism is used in RX and TX threads for receiving and sending packets from NIC. Now we describe how to perform VNF deployment in the service chain in our experiments. We first obtain solutions to MVDP-Offline and MVDP-Online using Algorithm 1 and 2 respectively. According to these solutions, we bind each VNF to a dedicated CPU core located onto one server. The `CORELIST` and `SERVICE_ID` parameter in OpenNetVM specify the index of CPU core and VNF, respectively.

Testbed Setup. One Mellanox SN2700 switch connects five servers, two of which have dual Xeon(R) E5-2630 CPUs (2x8 physical cores) with 128GB memory, and the rest have Intel Xeon E5-2650V4 CPU (2x12 physical cores) with 64GB memory. Each one is equipped with an Intel 82599ES 10G dual port NIC and runs Ubuntu 14.04.3 with kernel version 3.19.0. We use `pktgen` to generate 14 UDP flows with 64 bytes packet size in each run, where each flow is required to pass through 2 or 4 pre-defined VNFs deployed onto different servers. There are 52 VNFs performing basic forwarding and monitor function, all of which are lightweight VNFs [17].

TABLE 3
The Normalized CPU and Memory Capacities for Heterogeneous Network Function Platforms in Google's Cloud [44], [45]

	1	2	3	4	5	6	7	8
CPU	0.50	0.50	0.50	**1.0**	0.25	0.50	0.50	0.50
Mem.	0.50	0.25	0.75	**1.0**	0.25	0.12	0.03	0.97

Experiment Results. We measure the maximum CPU and memory utilization using `linux top` command with different packet arrival rate for offline and online setting in Tab. 2. The 100% arrival rate corresponds to around 7 Gbps in our testbed. The resource utilization increases when the packet arrival rate becomes large. Specifically, the maximum CPU utilization using MVDP-Offline and MVDP-Online is 9.6% and 10.5%, respectively. The utilization is not very high since the computation complexity of these lightweight VNFs is around 60 CPU cycles per packet on average. From Tab. 2, we can observe that MVDP-Offline and MVDP-Online in general work well and their resource utilization can be relatively stable even the packet arrival rate becomes larger. This demonstrates that our algorithm takes multi-resource constraints into consideration and can make near optimal decision to deploy VNFs.

Discussions. Note that the solutions produced by our algorithms may violate the resource capacity. Based on the theoretical analysis above, the resource violation can be bounded. To guarantee the physical resource capacity cannot be violated, we can reserve a bounded number of resources in advance.

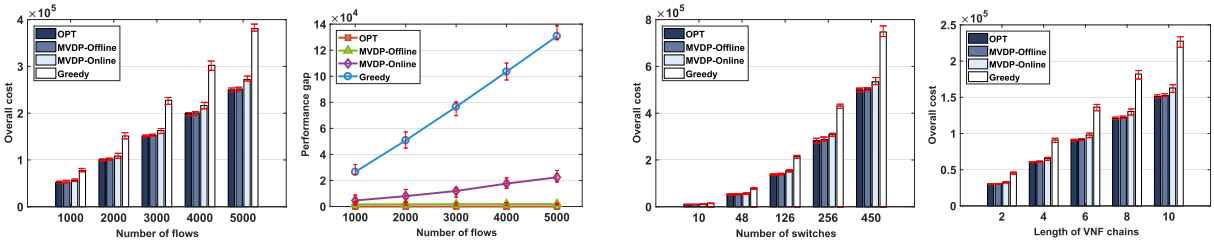
6.2 Simulation

We also conduct extensive simulations to thoroughly evaluate our algorithm at scale.

Setup. In addition to the OpenNetVM experiments in our testbed, here we use a large-scale synthetic network topology [40], which can be usually used for evaluating the performance of VNF deployment [41]. This topology can be divided into access layer, aggregation layer and core layer, where the switches in the access layer connect a large number of heterogeneous servers. Tab. 3 shows the normalized CPU and memory capacities for eight types of heterogeneous network function platforms. For each type, the cost used in our experiment is the product of the baseline cost [42], [43] and normalized CPU and memory capacities. The number of network platforms for each type varies with the experiments.

We run our algorithms on a server with Intel(R) Xeon(R) CPU E5-2650 and 64 GB memory and generate different numbers of flows to measure the performance.

Experiment Results. We first investigate the overall cost for different VNF deployment schemes generated by Greedy, MVDP-Offline and MVDP-Online. In addition, we compare



(a) Overall cost with different number of flows (b) Additive optimality gap with different number of flows (c) Overall cost with different number of switches in the access layer (d) Overall cost with different length of VNF chains

Fig. 2. Overall cost comparison.

our algorithms against a branch and bound method that solves the program (3) optimally, denoted as OPT.

Overall Cost With Different Number of Flows. We can see that in Fig. 2a, as the number of flows increases, Greedy yields significantly much cost, while that of MVDP-Offline and MVDP-Online is below 2.8×10^5 all the time and can achieve near optimal. Specifically, the overall cost for Greedy, MVDP-Offline, MVDP-Online and OPT is 3.8×10^5, 2.5×10^5, 2.7×10^5 and 2.4×10^5, when the number of flows is 5000. We can observe that MVDP-Offline and MVDP-Online can reduce the overall cost by 33.3% and 28.4% respectively, compared to Greedy. This demonstrates that our algorithms can reduce the overall cost by flexibly deploying different VNFs.

Additive Optimality Gap With Different Number of Flows. Fig. 2b shows the additive optimality gap for MVDP-Offline, MVDP-Online and Greedy compared to OPT. For this simulation we vary the number of flows from 1000 to 5000. Intuitively, a larger additive optimality gap indicates more deployment cost resulting from a worse solution. We can see that, as the number of flows increases, Greedy yields significantly larger optimality gap compared to MVDP-Offline and MVDP-Online, where MVDP-Offline and MVDP-Online can guarantee that the additive optimality gap is less than 1.3×10^4 and their overall cost is always less than 2.8×10^5, even though the number of flows becomes larger.

Overall Cost With Different Number of Switches in the Access Layer. Fig. 2c illustrates the overall deployment cost for MVDP-Offline, MVDP-Online, Greedy and OPT in large-scale networks, where the value of x-axis represents the number of switches and that of y-axis represents the overall cost. We fix the length of VNF chains at 15 in this setting. When the number of switches is 450, the overall cost for MVDP-Offline, MVDP-Online, Greedy and OPT is 5.0×10^5, 5.3×10^5, 7.5×10^5 and 4.9×10^5. We can see that MVDP-Offline and MVDP-Online can reduce the overall cost by 32.7% and 27.9% on average compared to Greedy respectively.

Overall Cost With Different Length of VNF Chains. Fig. 2d shows that the overall cost varies with the length of VNF chains. Essentially this measures the efficiency for different deployment solutions. We observe that the overall cost increases as the length of VNF chains becomes longer. When the length of VNF chains is 8, the overall cost for MVDP-Offline, MVDP-Online, Greedy and OPT is 1.2×10^5, 1.3×10^5, 1.8×10^5 and 1.2×10^5 respectively. MVDP-Offline and MVDP-Online can reduce the overall cost by 32.9% and 28.2% compared with Greedy on average.

Overload Resource Percentage. We measure the percentage of overload resources in Fig. 3. We define this percentage as the ratio between the number of overload resources and that of total resources among all servers (i.e., the ratio between the number of violated constraints and that of total constraints). In this setting, the number of resource types is 2 and the length of VNF chains is around 8 on average. The overloaded percentage becomes larger when the number of flows increases since large number of running VNFs require more resources. Specifically, the overload percentage for MVDP-Offline and MVDP-Online is 7% and 5% respectively, when the number of flows is 4000. We can reduce the flow demand or increase the resource capacity in practice to further make the overload percentage smaller.

The CDF of Overbook Ratio. Fig. 4 shows the CDFs of the normalized overbook ratio $\{\tau_{i,k}\}$ when the consumed resource is larger than the resource capacity. The number of flows is fixed at 5000. We define the overbook ratio $\tau_{i,k}$ as following.

$$\tau_{i,k} = \frac{\sum_{f \in F} \sum_{j \in VNF_f} r^k_{f,i,j} - r^k}{r^k}, \forall i \in M, \forall k \in \{1, 2\}.$$

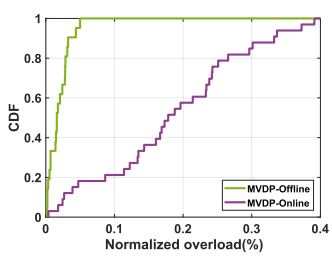


Fig. 4. Overbook ratio CDF.

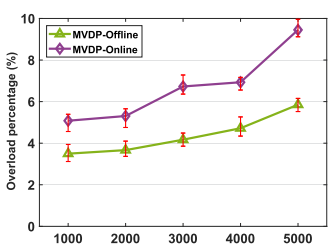


Fig. 3. Overload resource percentage.

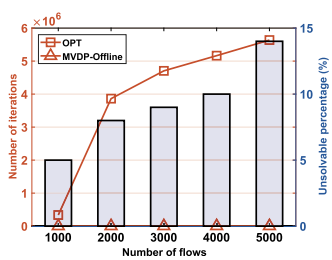


Fig. 5. Solver iterations for OPT.

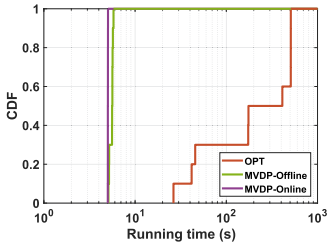


Fig. 6. Running time.

We can observe that the overbook ratio for MVDP-Online is always less than 0.4%, while that for MVDP-Offline is less than 0.05%. This indicates that the resource violation is very small even though the network is saturate. We can reserve a bounded number of resources to make sure our solution can be put into practice.

Solver Iterations for OPT. We now look at the number of iterations and the percentage of solvability when we obtain the optimal solution OPT using standard solver. In Fig. 5, the number of flows varies from 1000 to 5000 at the increment of 1000 for each run. We compare 200 different instances at each run. We found that the number of unsolvable instances increases when the number of flows becomes large. Specifically, when the number of flows is 5000, around 13% instances cannot be solved by standard solver after 10 hours. This demonstrates that OPT cannot perfectly solve all instances, and it's going to get worse especially when the number of flows is large. In addition, Fig. 5 also shows the number of solver iterations for MVDP-Offline and OPT. We can see that the number of iterations for OPT increases significantly than that for MVDP-Offline, when the number of flows becomes large. The convergence rate of MVDP-Offline in general can be exponentially faster than that of OPT. This demonstrates that OPT does not work well and cannot be used in practice in large-scale settings.

Running Time. Finally, we evaluate the running time of MVDP-Offline, MVDP-Online and OPT which is illustrated as CDFs in Fig. 6. In this setting, the number of flows is fixed at 5000 and the number of required VNFs for each flow is 8 on average. We can see that most cases using MVDP-Offline and MVDP-Online finish within 6 seconds and OPT takes 500 seconds. The running time of OPT is 100 times than that of MVDP-Offline and MVDP-Online in the worst case. As discussed above, MVDP-Offline and MVDP-Online can be able to offer near equivalent performance and run faster than OPT.

7 Conclusion

We studied the problem of minimizing VNF deployment cost under multi-resource constraints in a heterogeneous cloud and proved its hardness. We proposed an offline bicriteria approximation algorithm and a competitive online algorithm to solve our problem. Large-scale simulations and DPDK-based OpenNetVM implementation show that our algorithms can significantly reduce the deployment cost and improve the performance in terms of multi-resource allocation.

Acknowledgments

The authors would like to thank the anonymous reviewers for their helpful comments on drafts of this article. This work was supported in part by the National Key Research and Development Program of China under Grant 2018YFB1004700, and in part by China NSF under Grant 61802172, Grant 61972254, Grant 61832005, Grant 61672353, Grant 61772265, and Grant 62072228, in part by China NSF of Jiangsu Province under Grant BK20201248, and in part by the Open Fund of PDL under Grant WDZC20205500109.

References

[1] T. Wang, G. Zhang, A. Liu, M. Z. A. Bhuiyan, and Q. Jin, "A secure IoT service architecture with an efficient balance dynamics based on cloud and edge computing," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4831–4843, Jun. 2019.

[2] A. Gember-Jacobson, R. Viswanathan, C. Prakash, R. Grandl, J. Khalid, S. Das, and A. Akella, "OpenNF: Enabling innovation in network function control," in *Proc. ACM SIGCOMM Comput. Commun. Rev.*, 2014, pp. 163–174.

[3] G. Liu, Y. Ren, M. Yurchenko, K. K. Ramakrishnan, and T. Wood, "Microboxes: high performance NFV with customizable, asynchronous TCP stacks and dynamic subscriptions," in *Proc. Conf. ACM Special Interest Group Data Commun.*, 2018, pp. 504–517.

[4] Z. Xu, F. Liu, T. Wang, and H. Xu, "Demystifying the energy efficiency of network function virtualization," in *Proc. IEEE/ACM 24th Int. Symp. Qual. Service*, 2016, pp. 1–10.

[5] "Intel DPDK," 2019. [Online]. Available: https://dpdk.org/

[6] "SR-IOV for NFV solutions practical considerations and thoughts," 2019. [Online]. Available: https://www.intel.com/content/dam/www/public/us/en/documents/technology-briefs/sr-iov-nfv-tech-brief.pdf

[7] K. Thimmaraju, S. Hermak, G. Rétvári, and S. Schmid, "MTS: Bringing multi-tenancy to virtual networking," in *Proc. Annu. Techn. Conf.*, 2019, pp. 521–536.

[8] S. P. Crago, *et al.*, "Heterogeneous cloud computing," in *Proc. IEEE Int. Conf. Cluster Comput.*, 2011, pp. 378–385.

[9] "Docker," 2018. [Online]. Available: https://www.docker.com/

[10] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica, "Dominant resource fairness: Fair allocation of multiple resource types," in *Proc. 8th USENIX Conf. Netw. Syst. Des. Implementation*, 2011, pp. 323–336.

[11] R. Cohen, L. Lewin-Eytan, J. Naor, and D. Raz, "Near optimal placement of virtual network functions," in *Proc. IEEE Conf. Comput. Commun.*, 2015, pp. 1346–1354.

[12] 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 *Proc. IEEE Conf. Comput. Commun.*, 2017, pp. 1–9.

[13] H. Feng, J. Llorca, A. M. Tulino, D. Raz, and A. F. Molisch, "Approximation algorithms for the NFV service distribution problem," in *Proc. IEEE Conf. Comput. Commun.*, 2017, pp. 1–9.

[14] M. Ghaznavi, A. Khan, N. Shahriar, K. Alsubhi, R. Ahmed, and R. Boutaba, "Elastic virtual network function placement," in *Proc. IEEE 4th Int. Conf. Cloud Netw.*, 2015, pp. 255–260.

[15] X. Zhang, C. Wu, Z. Li, and F. C. M. Lau, "Proactive VNF provisioning with multi-timescale cloud resources: Fusing online learning and online optimization," in *Proc. IEEE Conf. Comput. Commun.*, 2017, pp. 1–9.

[16] R. Potharaju and N. Jain, "Demystifying the dark side of the middle: A field study of middlebox failures in datacenters," in *Proc. Conf. Internet Meas. Conf.*, 2013, pp. 9–22.

[17] S. G. Kulkarni, *et al.*, "NFVnice: Dynamic backpressure and scheduling for NFV service chains," in *Proc. Conf. ACM Special Interest Group Data Commun.*, 2017, pp. 71–84.

[18] A. Tootoonchian, *et al.*, "ResQ: Enabling SLOs in network function virtualization," in *Proc. 15th USENIX Conf. Netw. Syst. Des. Implementation*, 2018, pp. 283–297.

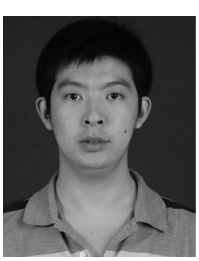
[19] "Amazon EC2," 2020. [Online]. Available: https://aws.amazon.com/

[20] W. Zhang, G. Liu, W. Zhang, N. Shah, P. Lopreiato, G. Todeschi, K. K. Ramakrishnan, and T. Wood, "OpenNetVM: A platform for high performance network service chains," in *Proc. Workshop Hot Topics Middleboxes Netw. Function Virtualization*, 2016, pp. 26–31.

[21] H. Moens and F. D. Turck, "VNF-P: A model for efficient placement of virtualized network functions," in *Proc. 10th Int. Conf. Netw. Service Manage. Workshop*, 2014, pp. 418–423.

[22] B. Addis, D. Belabed, M. Bouet, and S. Secci, "Virtual network functions placement and routing optimization," in *Proc. IEEE 4th Int. Conf. Cloud Netw.*, 2015, pp. 171–177.

[23] W. Ma, O. Sandoval, J. Beltran, D. Pan, and N. Pissinou, "Traffic aware placement of interdependent NFV middleboxes," in *Proc. IEEE Conf. Comput. Commun.*, 2017, pp. 1–9.
[24] C. Sun, J. Bi, Z. Zheng, H. Yu, and H. Hu, "NFP: enabling network function parallelism in NFV," in *Proc. Conf. ACM Special Interest Group Data Commun.*, 2017, pp. 43–56.
[25] J. Zhang, Z. Wang, C. Peng, L. Zhang, T. Huang, and Y. Liu, "RABA: Resource-aware backup allocation for a chain of virtual network functions," in *Proc. IEEE Conf. Comput. Commun.*, 2019, pp. 1918–1926.
[26] S. G. Kulkarni, G. Liu, K. K. Ramakrishnan, M. Arumaithurai, T. Wood, and X. Fu, "REINFORCE: achieving efficient failure resiliency for network function virtualization based services," in *Proc. CoNEXT*, 2018, pp. 41–53.
[27] C. You, "Hierarchical multi-resource fair queueing for network function virtualization," in *Proc. IEEE Conf. Comput. Commun.*, 2019, pp. 406–414.
[28] B. Gavish and H. Pirkul,"Algorithms for the multi-resource generalized assignment problem," *Manage. Sci.*, vol. 37, no. 6, pp. 695–713, 1991.
[29] Y. Guo, A. L. Stolyar, and A. Walid, "Shadow-routing based dynamic algorithms for virtual machine placement in a network cloud," *IEEE Trans. Cloud Comput.*, vol. 6, no. 1, pp. 209–220, First quarter 2018.
[30] S. Palkar, *et al.*, "E2: A framework for NFV applications," in *Proc. 25th Symp. Operating Syst. Princ.*, 2015, pp. 121–136.
[31] T. Wang, H. Xu, and F. Liu, "Multi-resource load balancing for virtual network functions," in *Proc. IEEE 37th Int. Conf. Distrib. Comput. Syst.*, 2017, pp. 1322–1332.
[32] Y. Liu, Z. Zeng, X. Liu, X. Zhu, and M. Z. A. Bhuiyan, "A novel load balancing and low response delay framework for edge-cloud network based on SDN," *IEEE Internet Things J.*, vol. 7, no. 7, pp. 5922–5933, Jul. 2020.
[33] D. P. Williamson and D. B. Shmoys, *The Design of Approximation Algorithms*. Cambridge, UK: Cambridge Univ. Press, 2011.
[34] R. Diestel, *Graph Theory*. Springer, 2012, vol. 173.
[35] R. J. Vanderbei *et al.*, *Linear Programming*. Berlin, Germany: Springer, 2015.
[36] N. Buchbinder and J. Naor, "The design of competitive online algorithms via a primal-dual approach," *Foundations Trends Theor. Comput. Sci.*, vol. 3, no. 2–3, pp. 93–263, 2009.
[37] S. Arora, E. Hazan, and S. Kale, "The multiplicative weights update method: A meta-algorithm and applications," *Theory Comput.*, vol. 8, no. 1, pp. 121–164, 2012.
[38] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge, UK: Cambridge Univ. Press, 2004.
[39] X. Meng, V. Pappas, and L. Zhang, "Improving the scalability of data center networks with traffic-aware virtual machine placement," in *Proc. IEEE INFOCOM*, 2010, pp. 1154–1162.
[40] N. Ron and T. Naveh, "Wireless backhaul topologies: Analyzing backhaul topology strategies," Ceragon White Paper, pp. 1–15, 2010.
[41] X. Jin, L. E. Li, L. Vanbever, and J. Rexford, "Softcell: Scalable and flexible cellular core network architecture," in *Proc. CoNEXT*, 2013, pp. 163–174.
[42] "A simple model for determining true total cost of ownership for data centers," 2018. [Online]. Available: http://tinyurl.com/kznlhn2/
[43] P. Patel, *et al.*, "Ananta: Cloud scale load balancing," in *Proc. ACM SIGCOMM Comput. Commun. Rev.*, 2013, pp. 207–218.
[44] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch, "Heterogeneity and dynamicity of clouds at scale: Google trace analysis," in *Proc. 3rd ACM Symp. Cloud Comput.*, 2012, pp. 1–13.
[45] C. Reiss, J. Wilkes, and J. L. Hellerstein, "Google cluster-usage traces," 2019. [Online]. Available: http://code.google.com/p/googleclusterdata/



Jiaqi Zheng (Member, IEEE) is currently a research assistant professor with the Department of Computer Science and Technology, Nanjing University, China. His research interests include computer networking, particularly data center networks, SDN/NFV, machine learning system, and online optimization. He was a research assistant at the City University of Hong Kong, in 2015 and collaborated with Huawei Noah's Ark Lab. He visited CIS center at Temple University, in 2016. He has received the Best Paper Award from IEEE ICNP 2015, Outstanding Doctorial Dissertation Award from ACM SIGCOMM China 2018, the First Prize of Jiangsu Science and Technology Award in 2018, and Outstanding Doctorial Dissertation Award from CCF, Jiangsu Province and Nanjing University, in 2019. He is a member of ACM.

Zixuan Zhang currently working toward the graduate degree from the School of Cyber Science and Engineering, Shanghai Jiao Tong University, P.R. China. His research interests include social network mining, deep reinforcement learning, and natural language processing.

Qiufang Ma received the BS degree from the the College of Information Engineering, Nanjing University of Finance and economics, Nanjing, China, in 2016, and the master's degree, in 2019. She is currently working toward the master's degree with the Department of Computer Science and Technology, in Nanjing University. Her research interests focus on network function virtualization.

Xiaofeng Gao (Member, IEEE) is currently a professor with the Department of Computer Science and Engineering, Shanghai Jiao Tong University, China. Her research interests include wireless communications, data engineering, and combinatorial optimizations. She has published more than 160 peer-reviewed papers, including well-archived international journals such as the *IEEE Transactions on Computers*, *IEEE Transactions on Knowledge and Data Engineering*, *IEEE Transactions on Parallel and Distributed Systems*, *Theoretical Computer Science*, and also in well-known conference proceedings such as INFOCOM, SIGKDD, and ICDCS. She has served on the editorial board of *Discrete Mathematics*, *Algorithms and Applications*.

Chen Tian (Member, IEEE) received the BS, MS, and PhD degrees from the Department of Electronics and Information Engineering, Huazhong University of Science and Technology, China, in 2000, 2003, and 2008, respectively. He is an associate professor with the State Key Laboratory for Novel Software Technology, Nanjing University, China. He was previously an associate professor with the School of Electronics Information and Communications, Huazhong University of Science and Technology, China. From 2012 to 2013, he was a postdoctoral researcher with the Department of Computer Science, Yale University. His research interests include data center networks, network function virtualization, distributed systems, Internet streaming, and urban computing.

Guihai Chen (Senior Member, IEEE) He is a distinguished professor at Nanjing University, China. He had been invited as a visiting professor by many universities including the Kyushu Institute of Technology, Japan, in 1998, the University of Queensland, Australia, in 2000, and Wayne State University during September 2001 to August 2003. He has a wide range of research interests with focus on sensor networks, peer-to-peer computing, high-performance computer architecture, and combinatorics. His papers have been cited for more than 10 000 times according to Google Scholar. Besides, he is a CCF fellow, since 2017.