2017 IEEE 37th International Conference on Distributed Computing Systems

Joint Optimization of Chain Placement and Request Scheduling for Network Function Virtualization

Qixia Zhang[1] Yikai Xiao[1] Fangming Liu*[1] John C.S. Lui[2] Jian Guo[1] Tao Wang[1]
[1]Key Laboratory of Services Computing Technology and System, Ministry of Education,
School of Computer Science and Technology, Huazhong University of Science and Technology
[2]The Chinese University of Hong Kong

***Abstract*—Compared with executing Network Functions (NFs) on dedicated hardwares, the recent trend of Network Function Virtualization (NFV) holds the promise for operators to flexibly deploy software-based NFs on commodity servers. However, virtual NFs (VNFs) are normally "chained" together to provide a specific network service. Thus, an efficient scheme is needed to place the VNF chains across the network and effectively schedule requests to service instances, which can maximize the average resource utilization of each node in service and simultaneously minimize the average response latency of each request. To this end, we formulate first VNF chains placement problem as a variant of bin-packing problem, which is NP-hard, and we model request scheduling problem based on the key concepts from open Jackson network. To jointly optimize the performance of NFV, we propose a priority-driven weighted algorithm to improve resource utilization and a heuristic algorithm to reduce response latency. Through extensive trace-driven simulations, we show that our methods can indeed enhance performance in diverse scenarios. In particular, we can improve the average resource utilization by 33.4% and can reduce the average total latency by 19.9% as compared with the state-of-the-art methods.**

I. Introduction

In today's enterprise and datacenter networks, middleboxes (e.g., firewall, load balancer, WAN accelerator)—also known as Network Functions (NFs)—play a critical role in ensuring security and enhancing performance [1]. Recently, the emerging Network Function Virtualization (NFV) technology shifts the way of how those NFs are implemented, by migrating them from dedicated hardwares to commodity servers. The trend of NFV makes it easy for operators to flexibly manage the network [2]–[4] and quickly deploy and scale up NFs to meet the traffic demand [5].

Typically, Virtual Network Functions (VNFs) are chained together—known as *NF chaining*—to provide a specific network service [6]–[8]. For instance, in datacenters, some flows need to traverse a firewall function and a load balancer function, while other flows need only to traverse the firewall function for processing. Since datacenter traffic exhibits high volume and high variation in both temporal and spatial dimensions [9], [10], an appropriate method is needed to place various VNF chains in datacenter networks so that they can serve requests effectively.

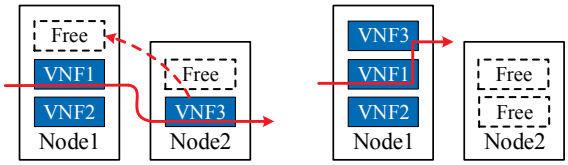
*This work was supported in part by the National 973 Basic Research Program under Grant 2014CB347800, and in part by NSFC under Grant 61520106005. (Corresponding author: Fangming Liu)

To flexibly process VNFs so as to achieve *high resource utilization* and *low response latency* in datacenters, we need to deal with two important tasks: (1) efficiently placing VNFs on commodity servers to achieve high resource utilization and (2) effectively scheduling requests to achieve low response latency, which includes both queuing latency and processing latency [11]. For the first task, Fig. 1 shows that low utilization of computing resource often increases the number of computing nodes in service, which furthermore increases the propagation delay and transmission cost of network flows [12]. For the second task, Fig. 2 shows that effectively scheduling requests can reduce the average queuing delay and processing latency of service instances. Here, a *service instance* means an instance of a VNF that is set up on a computing node and can serve requests with a positive service rate [13]. Besides, the second way of scheduling requests in Fig. 2 also lowers the job rejection rate, where *job rejection rate* refers to the ratio of requests rejected by the service instances among all requests due to the admission control mechanism.

Considering several essential characteristics of VNF chains and requests, we meet three major challenges when dealing with these two tasks:

- Since a request's arrival process at a VNF is associated with the service process at the former VNF (if there is any), this "chaining" requirement makes it inappropriate to place each VNF independently. Besides, different requests often require different VNF chains, which makes this problem even more challenging. Hence, we need to find an appropriate model so as to capture these important properties.
- Due to the difference in resource capacity among computing nodes and the difference in resource demand among VNFs, it is usually computational expensive to find out the optimal solution for placing all VNFs. Hence, we aim to find a near-optimal solution which can improve the average resource utilization of each computing node and reduce the execution cost simultaneously.
- Since each service instance of a VNF can be shared by multiple requests, improper scheduling of requests will lead to frequent congestions and high job rejection rate. Thus, an effective method is needed to schedule multiple requests to VNF instances, which would reduce queuing latency and job rejection rate.

1063-6927/17 $31.00 © 2017 IEEE
DOI 10.1109/ICDCS.2017.232



(a) Inter-server processing (b) Intra-server processing

Fig. 1: An example of two ways to place a VNF chain. By moving VNF3 from Node2 to Node1 (which has sufficient remaining resources to serve these three VNFs), the VNF chain is converted from (a) inter-server processing to (b) intra-server processing [11].

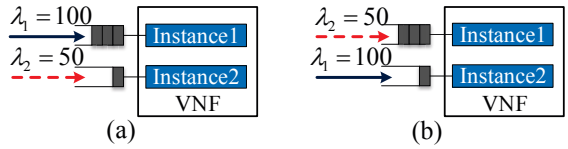


Fig. 2: An example of two ways to schedule two requests, where λ_1 and λ_2 represent the average packet arrival rate of each request. Instance1 and Instance2 are two service instances of a VNF, which are serving different numbers of requests respectively.

Differing from previous works, our work comprehensively addresses these three challenges. We tackle the first challenge by applying the theory of open Jackson network to model VNF chains in datacenter networks. For the second challenge, we formulate the VNF placement problem as a variable sized bin packing problem to prove its NP-hardness, and then we propose a cost-effective scheme to ensure near-optimal placement of VNFs. We also carefully design a heuristic algorithm to solve the third challenge.

In summary, we study the VNF chain placement and request scheduling problem, which is a timely important problem in datacenters. The contributions of this paper are as follows:

- We apply the theory of open Jackson network to model VNF chains. Our model not only captures the actual network traffic characteristics in datacenter networks, but also includes network congestions (reflected by packet loss rate) and job rejection rate.
- In order to maximize the average resource utilization, we propose a priority-driven weighted algorithm to achieve near-optimal placement of VNF chains with a theoretically-proved worst-case performance bound; and in order to minimize the average response latency, we propose a heuristic algorithm to schedule requests cost-effectively.
- Through theoretical analysis and extensive trace-driven simulations, we show that our methods improve the average resource utilization by 33.4% and reduce the average total latency by 19.9% as compared with the state-of-the-art methods.

II. Related Work

At present, previous works on VNF placement problem tend to map it into two NP-hard problems: Virtual Network Embedding (VNE) problem and location-routing problem [13]–[15]. Usually they do such mapping for showing NP-hardness and then solve the VNF placement problem accordingly. Cohen et al. [14] propose the NFV location problem and map it to two NP-Hard problems: the facility location problem and the Generalized Assignment Problem, and they solve the problem by jointly placing network functions and calculating path in the embedding process. Similarly, Mehraghdam et al. [13] hierarchically solve the placement of VNFs and then the chaining problem. In [15], Moens and Turck decouple the legacy VNE problem into VNF chaining and VM embedding problem, and they specify both VM requests and service requests. Xia et al. [16] provide an NFV network model for ISP operations.

Nevertheless, these NP-hard problems do not integrate some key characteristics in the VNF chain placement problem. For instance, the VNE problem mainly focuses on how to deploy VNFs on physical networks, specifically on different templates of VMs, while a request usually requires a chain of VNFs. In other words, chaining requirements are not addressed well in the VNE problem [14]. Furthermore, combining the VNE problem and the location-routing problem still does not address the problem well. Even though [13] and [15] consider the chaining requirements, their proposed solutions do not scale well for large problem instances. Moreover, most existing works just assume that the network is uncongested and do not consider packet loss situations. In fact, congestions do occur in datacenter networks, hence queuing delay and packet losses should not be ignored. In addition, retransmission of packets will bring in feedbacks among requests, which makes previous approaches not applicable. To cover these aspects, request scheduling should be jointly considered when placing VNF chains. Thus, an effective scheme to schedule requests is urgent needed in datacenter networks.

Differing from existing works, we apply the theory of open Jackson network to capture the actual network traffic characteristics in datacenter networks. Our model includes both network congestions (reflected by packet loss rate) and job rejection rate. We jointly optimize VNF chain placement and request scheduling by proposing two heuristic algorithms to achieve *high resource utilization* and *low response latency*, both of which outperform the state-of-the-art methods.

III. Model and Formulation

In this section, we first present our model as key notations are listed in Table I and II. Then we elaborate on why we apply the open Jackson network to model VNF chains. Lastly, we formally present our objectives combined with constraints.

A. Mathematical Model

We model the datacenter network as a connected graph $G = (V, E)$, where V is the set of computing nodes and E is the set of edges (or links) for connecting computing nodes through switch nodes (which are not included in set V). Since datacenter network provides high bi-sectional bandwidth [17], queuing latency on switch nodes is relatively low. We assume that there are sufficient switch capacities to ensure

the connectivity of the network, and we only consider placing VNFs on computing nodes in our model.

The computing resource consumption of a VNF can be expressed in terms of CPU, memory and network bandwidth. According to the related works, such as [3], [13] and [16], we find that CPU is usually defined as the bottleneck resource in most VNFs, while other hardware resources are relatively sufficient in most cases. Consequently, we define A_v as the CPU-bounded resource capacity of computing node $v \in V$, while other resources (e.g., memory, network bandwidth) are modeled as additional constraints.

A VNF $f \in F$ can be placed at any computing node $v \in V$ if it has sufficient resource capacity, where F is the set of VNFs. We use a binary variable x_v^f to indicate whether VNF $f \in F$ is deployed at node $v \in V$ (1 if so, 0 otherwise). Since multiple VNFs can be placed at the same computing node, we also define a binary variable y_v indicating whether computing node $v \in V$ has deployed any VNF $f \in F$. The relationship between y_v and x_v^f can be expressed in Eq. (1).

$$\forall v \in V : y_v = \begin{cases} 0, \sum_{f \in F} x_v^f = 0, \\ 1, \sum_{f \in F} x_v^f > 0. \end{cases} \tag{1}$$

In fact, multiple service instances of a VNF can be deployed at the same computing node to deal with multiple requests. We use M_f to indicate the number of service instances that VNF $f \in F$ can deploy. To satisfy the constraint of the integrity of each VNF, we suggest placing all service instances of a VNF at one computing node, which actually helps the operators to save the setup cost. If all the service instances still cannot cope with all the requests, we can then place some replicas of the VNF on different nodes, and regard each replica as a new VNF. Hence, we have Eq. (2).

$$\forall f \in F : \sum_{v \in V} x_v^f = 1. \tag{2}$$

We use a notation R to represent the set of requests. A chain of VNFs should be applied to a request $r \in R$ in a specific order, hence we use a symbol U_r^f to indicate whether VNF $f \in F$ is required in request $r \in R$ (1 if so, 0 otherwise). Since some service instances can be shared by multiple requests, we have inequality (3).

$$\forall f \in F : M_f \leq \sum_{r \in R} U_r^f. \tag{3}$$

We assume that the service time for packets on each service instance of VNF $f \in F$ follows an exponential distribution with a parameter μ_f. We distinguish the service instances by a positive integer $k \leq M_f$. We define a binary variable $z_{r,k}^f$ to indicate whether a request $r \in R$ uses the k-th service instance of VNF $f \in F$. We also define a binary variable η_v^r to indicate whether a request $r \in R$ traverses any VNF on node $v \in V$.

TABLE I: Set

Symbol	Description
G	The graph $G = (V, E)$ representing the datacenter network
V	The set of computing nodes within the network
E	The set of edges (or links) within the network
F	The set of Virtual Network Functions (VNFs)
R	The set of requests, where each request needs to traverse a specific VNF chain

TABLE II: Parameter and Variable

Symbol	Description
A_v	Resource capacity of computing node $v \in V$
D_f	Resource demand of each service instance of VNF $f \in F$
M_f	Number of service instances of VNF $f \in F$ that can be deployed
U_r^f	1 if request $r \in R$ uses VNF $f \in F$, 0 otherwise
μ_f	Average service rate of VNF $f \in F$, $\mu_f > 0$
λ_r	Average packet arrival rate of request $r \in R$, $\lambda_r > 0$
Λ_k^f	Equivalent total arrival rate of packets at the k-th service instance of VNF $f \in F$, $\Lambda_k^f > 0$
P_r	Probability of packets of request $r \in R$ that are received correctly by the destination, $0 < P_r \leq 1$
η_v^r	1 if request $r \in R$ needs to traverse any VNF placed at computing node $v \in V$, 0 otherwise
x_v^f	1 if VNF $f \in F$ is placed at computing node $v \in V$, 0 otherwise
y_v	1 if there is any VNF $f \in F$ placed at computing node $v \in V$, 0 otherwise
$z_{r,k}^f$	1 if request $r \in R$ uses the k-th ($k < M_f$) service instance of VNF $f \in F$, 0 otherwise

Their relationship can be described as follows:

$$\forall r \in R, v \in V : \eta_v^r = \begin{cases} 0, \sum_{f \in F} x_v^f U_r^f = 0, \\ 1, \sum_{f \in F} x_v^f U_r^f > 0. \end{cases} \tag{4}$$

For each request $r \in R$ using VNF $f \in F$, it should be mapped to exactly one service instance of f. In other words, if it does not traverse a VNF, none of its service instances can be used by this request. Hence, we have Eq. (5).

$$\forall r \in R, f \in F : \sum_{k=1}^{M_f} z_{r,k}^f = U_r^f. \tag{5}$$

A request can be allocated to any service instance if needed. Hence, the resource demand D_f of each service instance of VNF $f \in F$ can be estimated by the number of requests allocated to it. Since we can place multiple VNFs at the same computing node if and only if it has sufficient resource capacity, we express this constraint in inequality (6).

$$\forall v \in V : \sum_{f \in F} x_v^f \cdot M_f \cdot D_f \leq A_v. \tag{6}$$

For each request, packets arrive as a Poisson stream with an arrival rate λ_r. Let P_r ($0 < P_r \leq 1$) be the probability

that packets of request $r \in R$ are received correctly by the destination. Lost or incorrectly-received packets would be retransmitted from source to destination as a feedback. We use Λ_k^f to indicate the equivalent total arrival rate of the packets at the k-th service instance of VNF f with a packet loss rate $(1-P_r)$. Hence, we have the relationship between λ_r and Λ_k^f:

$$\forall f \in F : \Lambda_k^f = \sum_{r \in R} (\lambda_r / P_r) \cdot z_{r,k}^f. \quad (7)$$

B. Applying the Theory of Open Jackson Network

First of all, we explore the *Input Process*, *Queuing Discipline* and *Service Process* of our problem according to the queuing network theory [18].

- *Input Process* - Packets of a request r arrive stochastically as a Poisson stream with an arrival rate λ_r. The arrival process of each packet is independent of each other.
- *Queuing Discipline* - When a packet arrives at an idle service instance, it will get served immediately; Otherwise, the packet will be queuing in a buffer. Packets are served on a first-come, first-served basis.
- *Service Process* - Each service instance of a VNF handles packets independently. We assume that the service time is exponentially distributed and each service instance of VNF f has a single server fixed rate μ_f.

Based on the discussion above, we elaborate on how we apply the theory of open Jackson network to model requests.

A request with a packet loss feedback. For instance, as shown in Fig.3, packets of a request traverse two VNFs from source to destination, named by VNF_1 and VNF_2 respectively. Packets arrive as a Poisson stream with an arrival rate λ_0. The service time of both VNFs are exponentially distributed with a parameter μ_1 and μ_2 respectively. When a packet arrives, it is served by these two VNFs successively and then leaves the network. A NACK is sent by the destination when a packet has lost or not been properly received [11]. If so, the packet will be retransmitted as soon as the NACK is received by the source. We denote by P the probability of a packet being received correctly. Hence, packet loss rate can be expressed as $(1-P)$.

Since the total stream entering the network must be equal to the total stream leaving the network, we have:

$$\lambda_0 + (1-P)\lambda_2 = \lambda, \lambda \ = \lambda_1 = \lambda_2,$$

According to Burke's Theorem [18], when the network reaches its steady state, we have:

$$\lambda = \lambda_0 / P.$$

Let $E[N_i]$ be the average number of packets in the queue of VNF_i and $E[T_i]$ be the average response latency of each packet in the queue (i.e., the buffer) of VNF_i. Based on Jackson's theorem [18], we have:

$$E[N_i] = \frac{\lambda_0}{P\mu_i - \lambda_0}, \quad i = 1, 2,$$
$$E[T_i] = \frac{1}{P\mu_i - \lambda_0}, \quad i = 1, 2.$$

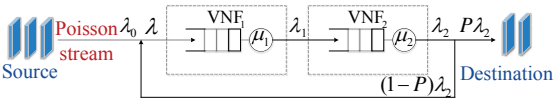


Fig. 3: An example of a request traversing two VNFs from source to destination with an Poisson arrival rate λ_0 and a packet loss rate $(1-P)$.

Besides, the total response time of this request is:

$$E[T] = \sum_{i=1}^{2} E[T_i] = \frac{1}{P\mu_1 - \lambda_0} + \frac{1}{P\mu_2 - \lambda_0}.$$

In conclusion, we notice that this queuing network must satisfy two conditions: (1) the interval time distribution of arriving requests follows a Poisson distribution; and (2) the service time of each service instance follows an exponential distribution. Based on Jackson's Theorem, we can model each service instance as an M/M/1 queue with the same packet arrival rate. Since an M/M/1 queue captures the growth in delay for low loads and high costs near system capacity, this model suits our hypothesis well. This way, we consequently model each request as an open Jackson network. Besides, with the growth of multi-processing capabilities, powerful network processors manage to keep the average response latency under a manageable threshold on traditional network nodes, despite the increased time that it takes for complex packet processing functions [19].

Multiple requests in a datacenter network. In a datacenter network, different VNFs can be distributed on different computing nodes. Since different requests may require different VNF chains, and some VNFs can be shared by multiple requests, several flows of packets may merge at the same node. Accordingly, all requests in a datacenter network can be modeled as a large interconnected network. Considering the influence among multiple requests, we believe that it is valid to apply the theory of open Jackson network.

Fig. 4 describes an example of this situation, where there are three requests r_1, r_2 and r_3, with their average packet arrival rate λ_1, λ_2 and λ_3. As we can see, packets of different requests traverse different VNFs. Besides, this figure also depicts how the requests are allocated to the service instances. For example, request r_2 uses the second service instance of VNF_2, and it shares the second service instance of VNF_3 with request r_1. Due to the sharing characteristic of service instances, an effective method is needed to merge the flows. Based on Kleinrock's Approximation [18], we define λ_i as the equivalent total arrival rate at a service instance i, which can be expressed by:

$$\lambda_i = \lambda_i^0 + \sum_{j=1}^{k} \lambda_j P_{ji}, \quad i = 1, ..., k.$$

Where λ_i^0 refers to external flows of requests and $\lambda_j P_{ji}$ refers to internal flows merging into service instance i.

This way, we merge several flows of requests into an service instance as one flow. Each service instance i behaves as

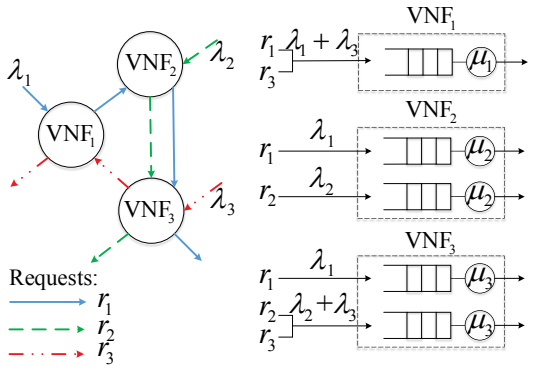


Fig. 4: An example of scheduling three requests to multiple service instances of three VNFs.

if the arrival stream Λ_i were Poissonian. Due to the state-independent service rate of each instance, we can calculate the probability of having n packets in the queue, $\pi(n)$, and the utilization of a service instance, ρ_k^f, which can be expressed as follows.

$$\forall f \in F : \pi(n) = (1 - \frac{\Lambda_k^f}{\mu_f})(\frac{\Lambda_k^f}{\mu_f})^n, \quad n \in \mathbb{N} \tag{8}$$

$$\forall f \in F : \rho_k^f = \Lambda_k^f / \mu_f. \tag{9}$$

where k is the k-th service instance of VNF f, $k = 1, ..., m_f$; and ρ_k^f should satisfy $\rho_k^f < 1$. When the arrival rate is larger than the service rate, the admission control mechanism will drop some requests to ensure the normal operation of the services. As mentioned in Sec. I, we use the job rejection rate to measure this metric.

In an open Jackson network, when each service instance reaches its steady state, the *average number of packets* $N(f,k)$ can be expressed in Eq. (10).

$$\forall f \in F : N(f,k) = \frac{\rho_k^f}{1 - \rho_k^f}. \tag{10}$$

Based on the additivity property of Poisson streams [20] and Little's formula [18], we can express the *average response latency* $W(f,k)$ in Eq. (11), which contains both queuing latency and processing latency.

$$\forall f \in F : W(f,k) = \frac{N(f,k)}{\sum_{r \in R} \lambda_r z_{r,k}^f} = \frac{\rho_k^f}{(1 - \rho_k^f) \sum_{r \in R} \lambda_r z_{r,k}^f}. \tag{11}$$

When $P_r = P, \forall r \in R$, we have:

$$W(f,k) = \frac{1/P}{\mu_f - \sum_{r \in R} \Lambda_k^f} = \frac{1}{P\mu_f - \sum_{r \in R} \lambda_r z_{r,k}^f}. \tag{12}$$

where $k = 1, ..., m_f$ is the k-th service instance of VNF f.

To sum up, considering various requests in a datacenter network, we apply the theory of open Jackson network for modeling. Based on Kleinrock's Approximation and Jackson's Theorem, we merge several flows of requests into an service instance as one flow, and model each service instance as an M/M/1 queue. Packets loss rate and job rejection rate are both included in our model.

C. Objectives

In the sections above, we have formulated some relations between parameters and variables. We assume all these constraints should be satisfied once defined. In general, we have the following two objectives.

Objective 1: *Maximize the average resource utilization of each computing node*. To this end, we aim to take full advantage of the resource capacity of each computing node in service, thus we have:

$$\begin{aligned} &\max \sum_{v \in V} ((\sum_{f \in F} x_v^f \cdot M_f \cdot D_f)/A_v) / \sum_{v \in V} y_v \\ &\text{s.t.} \begin{cases} \sum_{v \in V} x_v^f = 1, \forall f \in F \\ \sum_{f \in F} x_v^f \cdot M_f \cdot D_f \le A_v, \forall v \in V \end{cases} \end{aligned} \tag{13}$$

Insight: To achieve Eq. (13), we find another objective that is complementary to Objective 1— *minimizing the total number of computing nodes in service*. We express this objective in Eq. (14).

$$\min \sum_{v \in V} y_v \tag{14}$$

To reduce the total number of computing nodes in service, we have to fully utilize the resources of each used computing node, which means to $\max \sum_{f \in F} x_v^f$ for $\forall v \in V$, when $y_v = 1$. This in return improves the resource utilization of each computing node. Thus, we can conclude that Eq. (14) and Eq. (13) are complementary to each other. Besides, from the operators' perspective, using fewer computing nodes (i.e., commodity servers) is beneficial for saving operation cost.

Objective 2: *Minimize the average response latency of each service instance*. We express this objective in Eq. (15).

$$\begin{aligned} &\min \sum_{k=1}^{M_f} W(f,k)/M_f \\ &\text{s.t.} \sum_{k=1}^{M_f} z_{r,k}^f = U_r^f, \forall r \in R, f \in F \end{aligned} \tag{15}$$

Insight: In Eq. (12), we notice that $W(f,k)$ is positively correlated with $\sum_{r \in R} \lambda_r z_{r,k}^f$ when $P_r = P$ is a constant. Since each VNF f can deploy M_f service instances to serve $\sum_{r \in R} U_v^f$ requests, we aim to find an appropriate way to allocate the requests effectively. Considering Eq. (7), we find it advisable to balance the $\sum_{r \in R} \lambda_r z_{r,k}^f$ of each service instance as nearly equal as possible. This way, we can minimize the average response latency $W(f,k)$ of each service instance of VNF f.

The coordination of Objective 1 and Objective 2. The utilization and response time may usually be conflicting goals, hence we want to find a method to balance the tradeoff. To achieve Objective 1, we suggest minimizing the total number

of computing nodes in service, as declared in Eq. (15). This is conducive to reducing the interval traffic cost so as to reduce the total propagation and transmission delay of all requests. To achieve Objective 2, we aim to minimize the total response latency of all service instances. Therefore, by jointly achieving Objective 1 and Objective 2, we can minimize the total latency of all requests, which can be expressed in Eq. (16).

$$\min \sum_{r\in R}(\sum_{f\in F}\sum_{k=1}^{M_f} z_{r,k}^f U_r^f W(f,k) + (\sum_{v\in V}\eta_v^r - 1)L) \quad (16)$$

Where L is the sum of average propagation delay and transmission delay on the link between two computing nodes [11].

In short, Eq. (16) adds up two independent parts of the total latency of all requests (the total response latency on computing nodes plus the sum of communication latency on links). This way, we explore the coordination of these two objectives. By jointly achieving Objective 1 and Objective 2, we can flexibly place VNFs with *high resource utilization* and process requests with *low response latency* in datacenters.

IV. Algorithm Design

In order to jointly optimize VNF chain placement and request scheduling, we manage to solve the problem in a two-phase way. In phase one, we prove that the VNF chain placement (VNF-CP) problem is NP-hard and propose a priority-dirven weighted algorithm. Then we propose a heuristic algorithm to solve the request scheduling problem. Finally, we analyze the optimality and complexity of both algorithms.

A. VNF Chain Placement

To achieve Eq.(13), we find it helpful to refer to a NP-hard problem, the Variable Sized Bin Packing (VSBP) problem [21]. Even consider the simplified version of the VNF-CP problem, where each computing node $v \in V$ has the same resource capacity $A_v = a$, we can prove it still NP-hard.

Theorem 1. *The VNF-CP problem defined in Eq.* (13) *is NP-hard.*

Proof: First of all, coming back to the bin packing problem, there is a set of pieces with different sizes $W = \{w_1, w_2, ..., w_n | w_i \in (0,1], i = 1, ..., n\}$. Let Q be the set of bins with each size of 1. The target is to put all the pieces in these bins and minimize the total number of the bins used:

$$\min \sum_{i=1}^{n} y_i$$

$$\text{s.t.} \begin{cases} \sum_{j=1}^{n} w_i x_{ij} \le 1, x_{ij} \in \{0,1\} \\ y_i = 0, if \sum_{j=1}^{n} x_{ij} = 0;\ y_i = 1, if \sum_{j=1}^{n} x_{ij} > 1 \end{cases} \quad (17)$$

Given an instance $I = (w_1, ..., w_n, n, q)$ of the bin packing problem, we map an instance of Eq. (13), $I' = (|F| = n, M_f D_f = w_i, (A_v = a) = q)$ to I, where $M_f D_f$ stands for the total resource demand of each VNF. Obviously, we can do such mapping in polynomial time. We denote the total demand of all requests by a constant $\mathbf{C} = \sum_{f\in F} M_f D_f$. Considering Eq. (2), we have:

$$\begin{aligned} &\max \sum_{v\in V}((\sum_{f\in F} x_v^f \cdot M_f \cdot D_f)/A_v)/\sum_{v\in V} y_v \\ &\Rightarrow \max (\sum_{v\in V}\sum_{f\in F} x_v^f \cdot M_f \cdot D_f / \sum_{v\in V} a)/\sum_{v\in V} y_v \\ &\Rightarrow \max (\mathbf{C}/|V|a)/\sum_{v\in V} y_v \\ &\Rightarrow \max 1/\sum_{v\in V} y_v \\ &\Rightarrow \min \sum_{v\in V} y_v \end{aligned}$$

Therefore, if there exists an solution for I in the bin packing problem, then it also solves the VNF-CP problem, and *vice versa*. As a result, the VNF-CP problem can be formulated as a variant of the bin packing problem, which is NP-hard as well. ■

Although we prove the NP-hardness of the VNF-CP problem by mapping it to the bin packing problem, we notice some significant differences between them, which makes existing solutions to the VSBP problem not applicable to the VNF-CP problem. In the VSBP problem, there are unlimited bins of each size; while in the VNF-CP problem, each computing node is regarded as one and only node with a unique resource capacity. Hence, we carefully design a priority-driven weighted algorithm BFDSU (Best Fit Decreasing using Smallest Used nodes with the largest probability) to find a near optimal solution cost-effectively.

We use a set *Used_list* to save the computing nodes in service and a set *Spare_list* to save the spare computing nodes. Another set, *VNF_list* saves the VNFs that haven't been placed. Let RST(v) be the remaining resource capacity of computing node v. In BFDSU, we place VNFs from the most resource-demanding one to the least. When placing a VNF f, we try to find out a subset $V_{rst}(f) = \{v \in V | \text{RST}(v) \ge D_f^{sum} = D_f M_f\}$ that contains all nodes with sufficient resources RST(v) for placing it. We first search the *Used_list* for these nodes, and place f at one of the most suitable nodes in $V_{rst}(f)$ (if $V_{rst}(f) \neq \emptyset$); Otherwise, we attempt to place f at one of the most suitable spare node in the *Spare_list* and move that node from the *Spare_list* to the *Used_list*.

Intuitively, the most suitable node v for placing VNF f should be the one with minimal RST(v) in $V_{rst}(f)$. However, placing f at such node may not ensure a feasible solution. Instead, we introduce a weighted probability strategy, where we place the VNF at such node with the maximum probability. Specifically, for all the nodes $v \in V_{rst}(f)$, we calculate the probability of placing f at node v by its reciprocal of RST(v).

Definition $P_{rst}(v)$ refers to the weight of $v \in V_{rst}(f)$ (assuming all computing nodes in $V_{rst}(f)$ have been sorted in ascending order by their $RST(v)$) to place VNF $f \in F$ and $Prob\_sum$ refers to the sum of all weights. Then we have the

upper bound of the probability of node v_k, $Prob\_bound(v_k)$:

$$P_{rst}(v) = 1/(1 + RST(v) - D_f^{sum})$$
$$Prob\_sum = \sum_{v \in V_{rst}(f)} (1/(1 + RST(v) - D_f^{sum}))$$
$$Prob\_bound(v_k) = \sum_{i=1}^{k} P_{rst}(v)/Prob\_sum$$

where $\forall f \in F, v \in V_{rst}(f), k = 1, 2, ..., |V_{rst}(f)|$. Note that a constant, 1 is added to the denominator of $P_{rst}(v)$ to make it nonzero. We also assume there is a virtual node v_0 and its $Prob\_bound(v_0) = 0$ for simplifying the procedure. When we place VNF f, we first generate a random number ξ within $Prob\_sum$. If ξ belongs to [$Prob\_bound(v_{k-1})$, $Prob\_bound(v_k)$), then place VNF f at node v_k.

In addition, we have considered dynamically adding or removing VMs. However, this work needs to cooperate with underlying mechanism of SDN [22]. Besides, placing a VNF at a computing node suffers from large setup cost (i.e., to get domain isolation, we would have to run each middlebox inside a Linux virtual machine and this will take around five seconds to boot [23]). Some existing works have solved this problem (e.g., ClickOS manages to reduce the setup time within 30 ms [23]). Differing from these works, we aim to find out an efficient scheme to place VNF chains on commodity servers, with a fixed number of VMs on each server. To avoid introducing this sizable setup cost, we should not frequently add or remove VMs.

B. Request Scheduling

To achieve Eq. (15), let us consider another NP-hard problem, the Multi-Way Number Partitioning (MWNP) problem [24]. It aims to divide a set of integers into a collection of subsets so that the sum of the integers in each subset is as equal as possible. Even the simplest version, a 2-way number partitioning is proved NP-hard [24]. There are some existing approximation algorithms for solving the MWNP problem, such as CGA (Complete Greedy Algorithm) and CKK (Complete Karmarkar-Karp) algorithm [24]. However, they do not scale well as the number of instances increases. Thus we carefully design a heuristic algorithm RCKK (Reverse Complete Karmarkar-Karp) to solve the request scheduling problem effectively.

Note that $\forall f \in F$, $W(f,k) = 1/(\mu_f - \Lambda_k^f)$, hence $W(f,k)$ is positively correlated with Λ_k^f, which can be represented by $W(f,k) \propto \Lambda_k^f$. Hence, when we allocate each λ_r of $\sum_{r \in R} U_v^f$ requests to M_f instances, we balance the $\sum_{r \in R} \lambda_r z_{r,k}^f$ of each service instance as equal as possible. This way, we can minimize the average response latency $W(f,k)$ of each service instance of VNF f.

We use a set $R_f = \{r | \forall r \in R, U_r^f = 1\}$ to save the requests requiring VNF $f \in F$. For each request $r \in R_f$, we initiate a partition in form of (λ_r,0,..,0) consisted of m values (one λ_r and $m-1$ zeros) in each position. Here, position i ($i = 1, 2, ..., m$) represents the i-th service instance. We maintain the state of each service instance by a state table, which saves the number of requests allocated to it. Then we add all the partitions into a set named by *Partition_list*. We use a set s_i to save the requests that allocated to the i-th instance. We search the *Partition_list* for two partitions with the largest value at the first position, namely a = (a_1,a_2,...,a_m) and b = (b_1,b_2,...,b_m). Then we combine them into a new partition ($a_1 + b_m$,$a_2 + b_{m-1}$,...,$a_m + b_1$). We resort it, normalize it by subtracting the value at the m-th position from each position and replace a and b by it. Meanwhile, we combine the request sets accordingly. For instance, if $a_i + b_{m-i}$ is the value in the i-th position of the new partition, we should also combine the request set s_i of partition a and s_{m-i} of b into a new set s_i'.

Algorithm 1 BFDSU: VNF Chain Placement Procedure

Input: The set of resource capacity of each computing node, $A = \{A_v | \forall v \in V\}$;
The set of total resource demand of each VNF, $D = \{D_f^{sum} | \forall f \in F\}$;
Other sets: *Used_list*, *Spare_list*, *VNF_list* and $V_{rst}(f)$;
Output: The set of placement result of each VNF, $X = \{x_v^f | \forall f \in F, v \in V\}$;

```
Begin: Initiate Used_list = ∅, Spare_list = V, VNF_list = F;
Sort all VNFs in the VNF_list in descending order by their total resource demand;
while VNF_list ≠ ∅ do
    Get the first VNF f in the VNF_list, reset V_rst(f) = ∅;
    Search the Used_list and add each computing node v into V_rst(f) if it satisfies RST(v) ≥ D_f^sum;
    if V_rst(f) = ∅ then
        Search the Spare_list and add each computing node v into V_rst(f) if it satisfies RST(v) ≥ D_f^sum;
    end if
    if V_rst(f) = ∅ then
        Go back to Begin;
    end if
    Sort all computing nodes in V_rst(f) in ascending order by their RST(v);
    Calculate the weighted probability P_rst(v) of each node in V_rst(f) and each probability upper bound Prob_bound(v_k);
    Generate a random number ξ within Prob_sum;
    if ξ ∈ [Prob_bound(v_{k-1}), Prob_bound(v_k)) then Place VNF f at node v_k, x_{v_k}^f = 1;
    end if
    Remove VNF f from the VNF_list and move the node v_i to the Used_list if it's from the Spare_list;
end while
return Z.
```

C. Optimality Analysis

First of all, we summarize some properties of the BFDSU algorithm: (1) we use two sets, *Used_list* and *Spare_list* to distinguish whether a computing node has placed any VNF and preferentially place a VNF at a computing node in service; and (2) we bring in weighted probability for finding a feasible solution; hence we try to place a VNF at a computing node with the highest probability if its remaining resource is minimal. Both methods help to improve the resource utilization of each computing node in service and meanwhile reduce the total number of the computing nodes in service.

Due to the NP-hardness of the VNF-CP problem, it is computational expensive to find out the optimal solution. In order to analyze the optimality of our algorithm, we derive

Algorithm 2 RCKK: Request Scheduling Procedure

Input: Number of instances that VNF $f \in F$ deploys, $m = M_f$;
The set of requests using VNF $f \in F$, $R_f = \{r | \forall r \in R, U_r^f = 1\}$, assume that $n = |R_f|$;
The set of partitions, *Partition_list* $= \{(\lambda_r, 0, .., 0) | \forall r \in R_f\}$;
The set of arrival rates of requests which require VNF $f \in F$, $\Omega = \{\lambda_r | \forall r \in R_f\}$;
Output: The set of scheduling results for each request, $Z = \{z_{r,k}^f | \forall r \in R_f, k = 1, ..., M_f\}$;

1: Sort all partitions in descending order by their value at the 1-th position ($\lambda_r \in \Omega$) in the *Partition_list*;
2: **while** *Partition_list* has more than one partition **do**
3: Combine the first two partitions P_a and P_b (with their sets) by adding each position's value in reverse order and get a new partition P;
4: Resort P by the value at each position in descending order;
5: Normalize P by subtracting the value at the m-th position from each position and get $P^{'}$;
6: Replace P_a and P_b by $P^{'}$ and add $P^{'}$ into the *Partition_list* according to the value at the 1-th position;
7: **end while**
8: **for** $i = 1$ to m **do**
9: For all requests r in set s_i, $z_{r,i}^f = 1$;
10: **end for**
11: **return** Z.

the *asymptotic worst-case performance bound* of algorithm BFDSU, which is defined as $\mathrm{S}_{\mathrm{BFDSU}}^{\infty}$ [25].

$$\mathrm{S}_{\mathrm{BFDSU}}^{\infty} = \lim_{n \to \infty} \sup\{\mathrm{SUM}(V)/\mathrm{OPT}(V)\}$$

where $\mathrm{SUM}(V)$ refers to the sum of computing nodes used in BFDSU and $\mathrm{OPT}(V)$ refers to the sum of computing nodes used in the optimal solution. In other words, $\mathrm{OPT}(V)$ stands for the minimal number of nodes used for placing all VNFs.

Theorem 2. *The asymptotic worst-case performance bound of Algorithm BFDSU is 2, that is* $S_{BFDSU}^{\infty} = 2$.

Proof: We use $m = \sum_{v \in V} y_v$ to represent the sum of nodes in service. We normalize A_v in (0,1] and normalize D_f^{sum} accordingly. Thus, we have $d(v) = \sum_{f \in F} x_v^f D_f^{sum}$. We sort the set of computing nodes $\{v_1, v_2, ..., v_m\}$ by A_v in descending order.

If $m = 1$, only one node is used, thus $\mathrm{SUM}(V) = \mathrm{OPT}(V)$.

Else if $m \geq 2$, for $i = 1, ..., m-1$, we have $d(v_i) + d(v_{i+1}) > A_{v_i}$ and $d(v_1) + d(v_m) > A_{v_1}$; Otherwise, a shift from $d(v_{i+1})$ to $d(v_i)$ must be done. Then we have:

$$\begin{aligned} 2\mathrm{OPT}(V) &\geq 2\sum_{i=1}^{m} d(v_i) = d(v_1) + (d(v_1) + d(v_2)) + ... \\ &+ (d(v_{m-1}) + d(v_m)) + d(v_m) \\ &> d(v_1) + \sum_{i=1}^{m-1} A_{v_i} + d(v_m) \\ &> \sum_{i=1}^{m-1} A_{v_i} + A_{v_1} \geq \sum_{i=1}^{m} A_{v_i} \\ &\geq \mathrm{SUM}(V) \end{aligned}$$

Therefore, $\mathrm{SUM}(V)/\mathrm{OPT}(V) \leq 2$. We can infer that $\mathrm{S}_{\mathrm{BFDSU}}^{\infty} = \lim_{n \to \infty} \sup\{\mathrm{SUM}(V)/\mathrm{OPT}(V)\} = 2$. ■

Let ε be a dimensionless, assume that we only have pieces of size $1/2 + \varepsilon$ and bins of size 1 and $1/2 + \varepsilon$. In this case, we can conclude that the *asymptotic worst-case performance bound* is 2. Although this theoretical bound gives a performance guarantee, this worst case hardly occurs in real scenarios. As we evaluated in Sec. V, the simulation results proves that our methods improve the resource utilization rate by around 30% as compared with the state-of-the-art methods.

As for algorithm RCKK, there is usually no optimal k-way partitioning, where each sum of partition is equal. Even if there is an optimal solution, due to the NP-hardness of the problem and the diversity of the partitions, it is computational expensive to figure out. More details about the optimality can be found in [24]. In fact, for a m-way partitioning, there are $m!$ ways to combining two partitions. In order to get a high-quality solution, but not increase too much execution time, we attempt to combine two normalized partitions in reverse order. This way, we achieve a very cost-effective solution to the request scheduling problem.

D. Complexity Analysis

For BFDSU, note that there are $m = |F|$ VNFs and $n = |V|$ computing nodes. First, sorting m VNFs in descending order requires $O(m\log m)$ computation. Note that *Used_list* $\cap$ *Spare_list* $= V$, searching both sets terminates in n iterations. Besides, sorting the $V_{rst}(f)$ terminates in $\log n$ iterations and placing a VNF costs needs $O(n)$ computation. Thus, the time complexity of BFDSU is $O(m(\log m + n\log n))$.

For RCKK, as we have defined in Sec. V, we have $m = M_f$ service instances and $n = |R_f|$ requests. First, we sort the requests in descending order, which can be finished in $O(n\log n)$. Note that there are $n-1$ iterations in the loop. In each iteration, step 3 needs $O(m)$ computation, step 4 needs $O(m\log m)$ computation and step 6 needs at most $O(n)$ computation. Due to $n \geq m$, as stated in Eq. (3), the total time complexity of RCKK is $O(nm\log m)$.

V. Evaluation and Analysis

We conduct extensive trace-driven simulations to evaluate the performance of both algorithms as compared with the state-of-the-art methods. Our experiment setup is based on real-world traces in datacenter networks.

A. Simulation Setup

1) Diverse VNF chains: First, Li and Chen in [26] summarize more than thirty commonly-used VNFs and classify them into nine categories. Traced by this survey, we scale the number of VNFs from 6 to 30, including at least six commonly-deployed VNFs, such as Network Address Translator (NAT), Firewall (FW), Intrusion Detection System (IDS), Load Balancer (LB), WAN Optimizer and Flow Monitor (FM). The number of requests ranges from 30 to 1000. Each request traverses a VNF chain consisted of at most 6 VNFs.

2) Scale-up network topologies: We adopt a connected graph to model the datacenter network based on [27], which contains

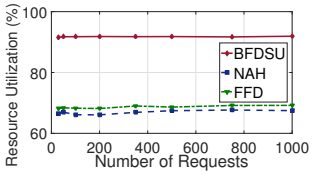


Fig. 5: The average resource utilization of 10 nodes under three algorithms.

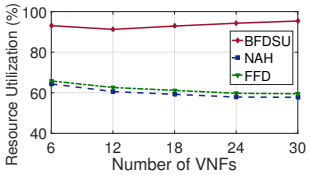


Fig. 6: The average resource utilization of used nodes handling 1000 requests under three algorithms.

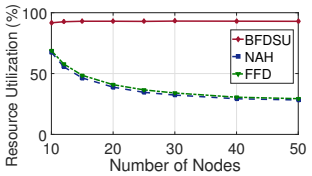


Fig. 7: The average resource utilization of used nodes for placing 15 VNFs under three algorithms.

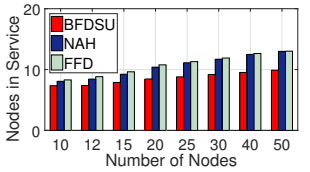


Fig. 8: The average number of nodes in service for placing 15 VNFs under three algorithms.

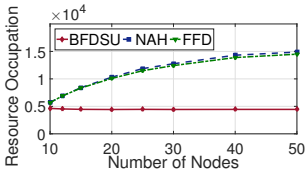


Fig. 9: The average resource occupation for placing 15 VNFs under three algorithms.

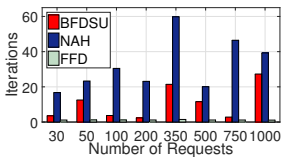


Fig. 10: The iterations of executing three algorithms for placing 15 VNFs.

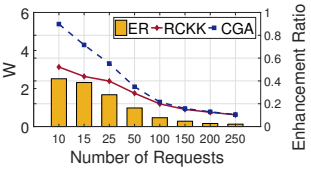


Fig. 11: The average response time of two algorithms and their enhancement ratio (P = 0.98).

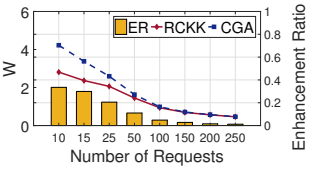


Fig. 12: The average response time of two algorithms and their enhancement ratio (P = 1.00).

from four to fifty computing nodes. There are sufficient switch capacities and network bandwidth for serving each request. The resource capacity of each computing node scales from 1 to 5000. One unit of resource capacity refers to the ability to handle one unit of workload per second, precisely, 64 Bytes packets at 10 kpps in our simulations. According to the reference [28], one CPU core can handle 64 Bytes packets at 1.5 Mpps, which equals to 150 units of resource capacity. Hence, in our simulations, a computing node with 5000 units of resource capacity indicates that it needs 34 CPU cores. Currently, most providers (e.g., Amazon EC2) can provide such VMs with up to 64 CPU cores, which should be enough to host our peak-time workload [29].

3) Arrival process and service process: Measured in datacenters, the arrival rate of requests follows the flow inter-arrival time distribution [9]. We assume that any external arrival to the network follows Poisson distribution with an arrival rate λ ranging from 1 to 100 pps. Each service instance of VNF f has the same exponential service rate μ_f. We estimate the latency of a request by its arrival time and the service rate that it gets at a service instance. The probability of packets being received correctly by the destination, P scales from 0.98 to 1. Besides, one unit of workload can be obtained by estimating (1) the arrivals of requests (from 1 to 100 pps), (2) the number of requests scheduled to each service instance (from 1 to 200) and (3) the number of service instances deployed on a computing node (from 1 to 25). Hence, we can adjust the grain size of the workload according to the actual demand.

B. Performance Evaluation for VNF Chain Placement

To evaluate the performance of BFDSU, we compare it with two state-of-the-art algorithms, FFD (First Fit Decreasing) and NAH (Node Assignment Heuristic algorithm for VNF placement) [12]. As introduced in [12], for each VNF chain, NAH first places the most resource-demanding VNF at the node with the largest remaining resource capacity. It then tries to place the other VNFs of that service chain at the same node as many as possible. Both FFD and NAH do not save the state of whether a computing node has placed any VNF or not.

Average resource utilization. As illustrated in Fig. 5, when the number of requests scales from 30 to 1000, all three algorithms' average resource utilization of used nodes remains stable. Precisely, it is 91.76%, 68.63% and 66.89% for BFDSU, FFD and NAH. We also observe this trend in Fig. 6, as we scale the number of VNFs from 6 to 30 and the number of nodes from 4 to 20. Our algorithm enhances the performance by 31.61% as compared with FFD and 33.41% with NAH. Fig. 7 also demonstrates that as the number of computing nodes scales from 6 to 30, the average resource utilization of FFD and NAH decreases while BFDSU stabilizes.

Insight: BFDSU improves the average resource utilization of computing nodes in service by around 30% as compared with two state-of-the-art algorithms, FFD and NAH.

Total computing nodes in service. We also have some interesting findings about the total number of computing nodes in service, as the number of computing nodes available increases. As shown in Fig. 8, with more computing nodes available, the average total number of used nodes increases slightly. We observe that BFDSU always uses fewest nodes while FFD uses most. According to the figure, BFDSU, NAH and FFD uses 8.56, 10.55 and 10.80 computing nodes in average. Besides, we also evaluate the total resource occupation of all computing nodes in service as a performance metric. As illustrated in Fig. 9, we find that our method maintains a stably low resource occupation, while FFD and NAH both show a growing trend in the resource occupation as the number of computing nodes increases.

Insight: As compared with the state-of-the-art methods, BFDSU achieves the minimal total number of computing nodes in service and the minimal resource occupation.

Execution Cost. To evaluate the execution cost of the three

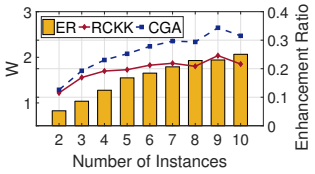


Fig. 13: The average response time of two algorithms and their enhancement ratio (P = 0.98).

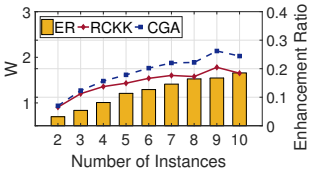


Fig. 14: The average response time of two algorithms and their enhancement ratio (P = 1.00).

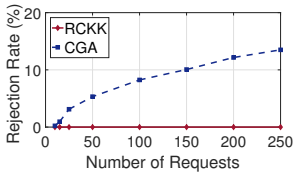


Fig. 15: The average job rejection rate of two algorithms under a low packet loss rate (P = 0.997).

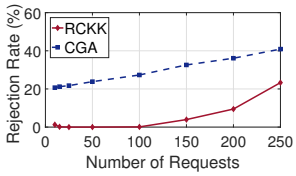


Fig. 16: The average job rejection rate of two algorithms under a high packet loss rate (P = 0.984).

algorithms, we measure their numbers of iterations for finding a feasible solution. Fig. 10 plots that the average number of iterations of BFDSU, NAH and FFD is 11, 32 and 1 respectively. As the number of requests increases, the iterations of FFD stay constantly lowest, while NAH takes nearly triple execution time than BFDSU.

Insight: Considering both the performance metrics and execution cost, BFDSU performs as the most cost-effective algorithm as compared with the state-of-the-art algorithms.

C. Performance Evaluation for Request Scheduling

For the request scheduling problem, we compare RCKK with CGA. We execute both algorithms for 1000 times and calculate the average values as the simulation results.

Average response time. We fix the number of service instances at 5, while the number of requests scales from 15 to 250. W refers to the *average response latency*, which is the main performance metric in Eq. (15). The enhancement ratio is defined as $(W_{\text{CGA}} - W_{\text{RCKK}})/W_{\text{CGA}}$ to show the improvement of W from CGA to RCKK. We scale μ_f with the number of requests to eliminate its dominant influence. To reflect the network congestion degree, we set the probability of a packet being received correctly, P, at two values, precisely 1 and 0.98 (i.e., packet loss rates are 0% and 2%).

Fig. 11 and Fig. 12 plot the average response time of five instances with P = 0.98 and 1.00 respectively. As illustrated in these two figures, RCKK always outperforms CGA in W as the number of requests increases under different packet loss rates. With P = 0.98 and 1.00, the enhancement ratio between CGA and RCKK, W, is reducing from 41.89% to 2.10% and from 33.49% to 1.17% respectively.

Different from the simulations above, Fig. 13 and Fig. 14 plot the performance metrics when the number of service instances scales from 2 to 10. As shown in Fig. 13, when the number of service instances grows, RCKK reduces the average response time by 5.24% to 25.05% as compared with CGA. When P = 1.00, as illustrated in Fig. 14, the enhancement ratio is from 3.16% to 18.53%. We can also conclude that with a higher packet loss rate, the average response time is increasing accordingly and so is the enhancement ratio.

In addition, some tail statistics attract our attention, where tail refers to the 99^{th} percentile response time in 1000 simulation results. With the number of requests scaling from 10 to 200, we find that RCKK reduces the 99^{th} percentile response time of by 44.54% to 5.18% as compared with CGA. For instance, when we schedule fifty requests to five service instances with P = 0.98, we find that the 99^{th} percentile response time of RCKK is within 1.23 while CGA is within 1.60, and the enhancement ratio is 23.17%.

Insight: Generally, RCKK outperforms CGA in minimizing the total response latency under two packet loss rates when we successively vary the number of requests and service instances.

Job rejection rate: As declared in Sec. I, when the arrival rate of the requests is larger than the service rate of a service instance, the admission control mechanism drops some requests to ensure the normal operation of the services. We measure this metric under two packet loss rates $(1 - P)$. As shown in Fig. 15 and Fig. 16, we find that with a higher packet loss rate, the job rejection rate is consequently higher. Under a low packet loss rate, when P = 0.997, RCKK nearly maintains a zero job rejection rate while this rate of CGA rises as the number of requests increase. Fig. 16 plots that under a low packet loss rate, when P = 0.984, the average job rejection rate of RCKK and CGA is 4.87% and 28.28% respectively.

Insight: RCKK achieves a lower job rejection rate as compared with CGA under different packet loss rates.

VI. Conclusions

In this paper, we present a hierarchically two-phase solution for joint optimization of VNF chain placement and request scheduling problem. We apply the theory of open Jackson network to model VNF chains. Our model not only captures the actual network traffic characteristics in datacenter networks, but also includes network congestions (reflected by packet loss rate) and job rejection rate. In order to maximize the resource utilization rate, we formulate the VNF chain placement problem as a variant of variable-sized bin-packing problem, and propose a priority-driven weighted algorithm BFDSU to ensure a near-optimal solution with theoretically proved worst-case performance bound. To minimize the average response latency of each service instance, we also propose a heuristic algorithm RCKK to optimize request scheduling cost-effectively. Through extensive trace-driven simulations, we show that our methods scale well in diverse scenarios. In particular, BFDSU improves the average resource utilization by 31.6% and 33.4% as compared with FFD and NAH respectively. Besides, BFDSU achieves the minimal nodes in service for placing all VNFs. Under different packet loss rates, RCKK not only reduces the average response latency of each instance by 19.9%, but also lowers the average job rejection rate by 23.4% as compared with CGA.

REFERENCES

[1] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, and V. Sekar, "Making middleboxes someone else's problem: network processing as a cloud service," 2012.

[2] S. Palkar, C. Lan, S. Han, K. Jang, A. Panda, S. Ratnasamy, L. Rizzo, and S. Shenker, "E2: a framework for nfv applications," in *ACM Proc. of SOSP*, 2015.

[3] F. Callegati, W. Cerroni, C. Contoli, and G. Santandrea, "Dynamic chaining of Virtual Network Functions in cloud-based edge networks," in *Network Softwarization (NetSoft), 2015 1st IEEE Conference on*. IEEE, 2015, pp. 1–5.

[4] B. Li, K. Tan, L. Luo, Y. Peng, R. Luo, N. Xu, Y. Xiong, and P. Cheng, "ClickNP: Highly flexible and High-performance Network Processing with Reconfigurable Hardware," pp. 1–14, 2016.

[5] A. Gember-Jacobson, R. Viswanathan, C. Prakash, R. Grandl, J. Khalid, S. Das, and A. Akella, "OpenNF: Enabling innovation in network function control," 2015.

[6] S. K. Fayazbakhsh, L. Chiang, V. Sekar, M. Yu, and J. C. Mogul, "Enforcing network-wide policies in the presence of dynamic middlebox actions using flowtags," in *USENIX Proc. of NSDI*, 2014.

[7] P. Quinn and T. Nadeau, "Service function chaining problem statement," *draft-ietf-sfc-problem-statement-10 (work in progress)*, 2014.

[8] A. Panda, S. Han, K. Jang, M. Walls, S. Ratnasamy, and S. Shenker, "Netbricks: taking the v out of nfv," in *Usenix Conference on Operating Systems Design and Implementation*, 2016, pp. 203–216.

[9] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *ACM Proc. of IMC*, 2010.

[10] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The nature of data center traffic: measurements & analysis," in *ACM Proc. of IMC*, 2009.

[11] J. F. Kurose and K. W. Ross, *Computer Networking: A Top-Down Approach*.

[12] 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.

[13] S. Mehraghdam, M. Keller, and H. Karl, "Specifying and placing chains of virtual network functions," in *IEEE Proc. of CloudNet*, 2014.

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

[15] H. Moens and F. De Turck, "VNF-P: A model for efficient placement of virtualized network functions," in *IEEE Proc. of CNSM*, 2014.

[16] B. Addis, D. Belabed, M. Bouet, and S. Secci, "Virtual network functions placement and routing optimization," in *IEEE Proc. of CloudNet*, 2015.

[17] F. Liu, J. Guo, X. Huang, and J. C. S. Lui, "eBA: Efficient Bandwidth Guarantee Under Traffic Variability in Datacenters," *IEEEACM Transactions on Networking*, pp. 1–14, 2016.

[18] E. Gelenbe, G. Pujolle, and J. Nelson, *Introduction to queueing networks*. Wiley Chichester, 1998.

[19] A. Dwaraki and T. Wolf, "Adaptive Service-Chain Routing for Virtual Network Functions in Software-Defined Networks," in *Workshop*, 2016, pp. 32–37.

[20] G. M. Kamath, E. Şaşoğlu, and D. Tse, "Optimal haplotype assembly from high-throughput mate-pair reads," in *IEEE Proc. of ISIT*, 2015.

[21] R. Grandl, G. Ananthanarayanan, S. Kandula, S. Rao, and A. Akella, "Multi-resource packing for cluster schedulers," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4, pp. 455–466, 2014.

[22] T. Wang, F. Liu, J. Guo, and H. Xu, "Dynamic SDN controller assignment in data center networks: Stable matching with transfers," pp. 1–9, 2016.

[23] J. Martins, M. Ahmed, C. Raiciu, V. Olteanu, M. Honda, R. Bifulco, and F. Huici, "ClickOS and the art of network function virtualization," in *USENIX Proc. of NSDI*, 2014.

[24] R. E. Korf, "Multi-Way Number Partitioning," in *IJCAI*. Citeseer, 2009, pp. 538–543.

[25] E. G. Coffman Jr, M. R. Garey, and D. S. Johnson, "Approximation algorithms for bin packing: a survey," in *Approximation algorithms for NP-hard problems*. PWS Publishing Co., 1996, pp. 46–93.

[26] Y. Li and M. Chen, "Software-defined network function virtualization: a survey," *IEEE Access*, vol. 3, pp. 2542–2553, 2015.

[27] S. Orlowski, R. Wessaly, A. Tomaszewski, and R. Wess, "Sndlib 1.0 survivable network design library," in *Networks*, 2010, pp. 276–286.

[28] Z. Xu, F. Liu, T. Wang, and H. Xu, "Demystifying the energy efficiency of Network Function Virtualization," in *Quality of Service (IWQoS), 2016 IEEE/ACM 24th International Symposium on*. IEEE, 2016, pp. 1–10.

[29] Amazon EC2. [Online]. Available: http://aws.amazon.com/ec2/