Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2017.DOI

Design and Placements of Virtualized Network Functions for Dynamically Changing Service Requests based on a Core/Periphery Structure

YUKI TSUKUI, SHIN'ICHI ARAKAWA (Member, IEEE), SHIORI TAKAGI, MASAYUKI MURATA (Member, IEEE)
Graduate School of Information Science and Technology, Osaka University, 1–5 Yamadaoka, Suita, Osaka, 565–0871 Japan
e-mail: {y-tsukui, s-takagi, arakawa, murata}@ist.osaka-u.ac.jp

Corresponding author: Shin'ichi Arakawa (e-mail: arakawa@ist.osaka-u.ac.jp).

This work was partially supported by Grant No. 19104 from the National Institute of Information and Communications Technology (NICT) in Japan.

ABSTRACT Network Function Virtualization (NFV) is a system that provides application services by connecting virtual network functions (VNFs), and is expected to accommodate new service requests through the development of new VNF and connection with existing ones. Because VNFs are implemented by software, their design and placement are important problems for the NFV system, which reduce the current and future system costs. In this paper, we investigate the design principles and the placement policies that reduce the cost of designing and developing VNFs for accommodating new service requests. As for the design policy, we introduce a Core/Periphery-Based Design (CPBD) that utilizes the core/periphery concept for developing VNFs. In CPBD, "core" VNFs are developed in advance and repeatedly used to accommodate future service requests. While "core" VNFs are common to current and future service requests, "periphery" VNFs are developed and customized for each service request. Next, we investigate the placement policies of VNFs for CPBD to fully utilize the nature of their core/periphery structure. In addition, we examine the Center-Located Core/Periphery placement (CLCP) policy and the Geographically-Distributed Core/Periphery placement (GDCP) policy, and evaluate the long-term cost of the NFV system under resource restrictions to run VNFs. Our results show that CPBD reduces the long-term cost of design and development of VNFs by 23% compared to the design with no core VNFs. Moreover, in the case of no resource restrictions, both CLCP and GDCP reduce the long-term costs of placing and connecting VNFs by 15% compared to the existing VNF placement algorithm. With resource constraints, GDCP reduces the long-term costs over CLCP by 11%.

INDEX TERMS Network Function Virtualization (NFV), Virtual Network Function (VNF), VNF Placement, Core/Periphery Structure, Software Design, Microservice

I. INTRODUCTION

As service demands become increasingly diverse, Network Function Virtualization (NFV) is gaining attention. NFV can implement network functions, such as firewall and proxy server, as a Virtual Network Function (VNF), which is developed using software. VNFs can run on general-purpose hardware shared with other VNFs. NFV flexibly accommodates various service requests by connecting VNFs over networks.

In operating NFV systems, it is important to reduce the costs of accommodating network services. Many previous studies conducted on NFV have discussed placement algorithms that minimize the costs of placing VNFs [1], [2]. For example, Kim et. al. [1] used a genetic algorithm to minimize the power consumption and satisfy the service delay requirements of users. Nam et. al. [2] minimized the end-to-end service time by placing VNFs based on Zipf's law which models the frequency of VNFs use. Although these studies used different algorithms or approaches, they implicitly assumed that VNFs are developed in advance.

However, in reality, service requests may change drastically and require VNFs that have not yet been developed. Therefore, we need a suitable software design of VNFs and its placement to accommodate the current and future service requests at a lower cost. If VNFs are not appropriately designed, new VNFs will be added frequently depending on changes in service requests, which will lead to an increase in their cumulative development cost. Moreover, appropriate placement of VNFs is required to reduce opportunities for changing placement, such as adding, moving, and removing VNFs. In this paper, we investigate a software design and placement method for VNFs that can reduce long-term development costs against changes in service requests.

In considering the software design of VNFs, we introduce a core/periphery structure [3], [4], which has been used to interpret the behaviors of biological systems, social networks, and internet systems. Some system components, called "core", do not change despite the composition of the entire system being changed with time and mediate the connection of non core system elements, called "periphery." We interpret VNFs based on a core/periphery structure and distinguish them into core and periphery VNFs. It is expected that designing core VNFs such that they can be repeatedly used will reduce the long-term development cost for accommodating future service requests. However, the development cost of each core VNF is higher than that of each periphery VNF, because core VNFs need to be generalized to be connected with other VNFs. Therefore, we introduce a model for deriving the development costs of NFV software systems and reveal the benefit of introducing core/periphery structures in VNF software design.

Next, we investigate how to place VNFs designed based on a core/periphery structure, as the deployment cost of VNFs can be reduced by appropriately placing the core VNFs in advance, so that they can be shared to accommodate future service requests. In fact, the existing method can reduce the number of VNFs to be placed by sharing common VNFs among the service requests [5]. We examineCenter-Located Core/Periphery placement (CLCP) policy and Geographically-Distributed Core/Periphery placement (GDCP) policy. CLCP places core VNFs at the center of physical networks, which increases the opportunity for core VNFs to accommodate many service-chain requests. In contrast, GDCP places core VNFs for each topological cluster, which prevents resource exhaustion resulting from accommodating service-chain requests. In addition, simulations are conducted for CLCP and GDCP, and the long-term cost of the NFV system is evaluated under resource restrictions to run VNFs.

The remainder of this paper is organized as follows. Section II explains our design and placement problems, as well as related work. Section III explains the software design of VNFs based on a core/periphery structure and shows its long-term cost through a numerical analysis. Section IV develops placement algorithms for VNFs designed based on the core/periphery structure. Section V presents our conclusions and future work.

II. DESIGN AND PLACEMENT PROBLEMS OF NFV SOFTWARE SYSTEMS

A. DESIGN PROBLEM

For the operation of an NFV system, it is important to design a VNF properly, to reduce costs. The NFV system comprises many VNFs and connects them to accommodate the service requests. There are some costs incurred in designing and developing VNFs, which disturb the flexible accommodation for service requests. A suitable software design of VNFs can reduce such costs.

A monolithic software design has been widely used for software such as networking software. In monolithic software design, multiple components form a single module [6]. These components are designed to compose a particular service and connect specific components. Thus, changing components can incur changes in other components, and thus, increase the development cost [7]–[11]. Moreover, tight coupling makes it difficult to use the components already developed for accommodating new services. Existing studies [7], [8] have analyzed how software components have been designed and developed in the long term, such as Linux and Mozilla, and indicated that large-scale refactoring to reduce tight coupling and increase the generality of components will contribute to fastening the application development.

Recently, in the field of software engineering, microservices have gained attention due to the possibility of reducing the development cost [6], [11], [12]. In microservices, components are well independent and can be connected with other components to form various services. The developed components can be used to accommodate future services; thus, microservices are expected to reduce the number of components and costs for design and development.

However, in the case of networked software, such as NFV systems, sufficient discussions have not been conducted on software. This paper discusses networked-software designs that have not attracted enough attention so far. To reduce development costs, such as those related to microservices, we design "core" VNFs to be used to accommodate new service-chain requests.

B. PLACEMENT PROBLEM

Many NFV studies have developed placement algorithms that can minimize the costs related to VNF placement and accommodate service requests efficiently . One such method [1] uses a genetic algorithm to solve the problem that minimizes power consumption while satisfying the service delay requirements of users. Another method [2] places VNF on a physical network based on Zipf's law, which models the frequency of use, to minimize the end-to-end service time. The existing placement algorithm, called affiliation-aware vNF placement (AaP), can reduce the number of VNFs to be placed by merging the service requests [5]. AaP places VNFs on the shortest path between the source and destination nodes

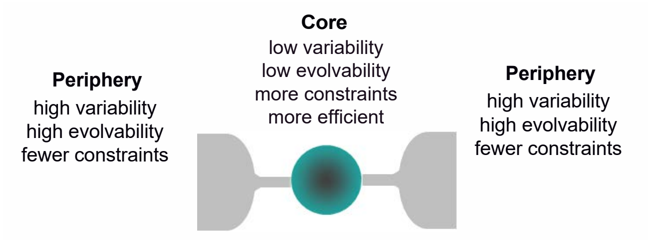

FIGURE 1: Concept of core/periphery structure

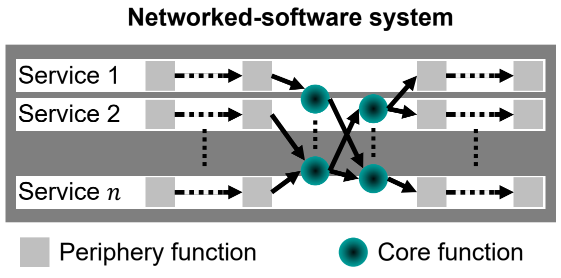

FIGURE 2: Example of NFV system

in order of each merged request, for avoiding bandwidth consumption.

For example, in accommodating two service requests, such as VNF1 → VNF2 → VNF3 and VNF4 → VNF2 → VNF5, AaP merges these requests and assumes them as one service request, such as VNF1 → VNF4 → VNF2 → VNF3 → VNF5. Here, VNF2 is shared and the number of VNF2 placements is reduced from 2 to 1. These studies aim at optimization in terms of power consumption, end-to-end service time, and bandwidth consumption.

However, considering the long-term operation of an NFV system, it is more important to reduce costs to additionally place VNFs and change the VNF location. For accommodating the service requests, the NFV system places VNFs into a physical network and connects them in a suitable order. However, there are some costs incurred in placeing and connecting VNFs, which are called deployment costs. In an operating NFV system, service requests may change variously and require VNFs that have not yet been placed and connected. These VNFs require additional costs to newly place and connect them. Moreover, changing the placement of VNFs, such as adding, moving, and removing them, suspends their execution and causes a delay in data communication [13]. Such factors increase the deployment cost. However, the above mentioned existing placement methods [1], [2], [5] may frequently change the VNF placement, because they do not consider changing the placement depending on the variation in the service requests. In this paper, we investigate a VNF placement method that can reduce long-term deployment costs against changes in service requests.

C. APPROACHES WITH CORE/PERIPHERY STRUCTURE

In considering the software design of VNFs, we introduce a core/periphery structure [3], [4], which has been used to interpret the behaviors of biological, social, and internet systems. A system possessing a core/periphery structure operates stably. This is because some system components, such as the "core", do not change despite changes in the composition of the entire system with time, and mediate the connection of non core system elements, such as the "periphery." Fig. 1 shows the basic concept of a core/periphery structure.

We interpret VNFs based on a core/periphery structure, and distinguish them into core and periphery VNFs. It is expected that the additional VNFs and their development cost will be reduced by designing VNFs such that the core VNFs can be used repeatedly to accommodate future service requests. Moreover, the deployment cost can be reduced by suitably placing VNFs designed based on a core/periphery structure. To reduce the deployment cost, we place the core VNFs in advance, so that they can be used repeatedly to accommodate future service requests, as in the case of software design. In fact, the above-explained AaP can reduce the number of VNFs to be placed by sharing common VNFs among the service requests [5].

III. CORE/PERIPHERY-BASED DESIGN OF NFV SYSTEMS

A. CORE / PERIPHERY BASED DESIGN

An NFV system has many VNFs and accommodates various service requests by appropriately connecting them. The connecting order of VNFs is called a service-chain. In general, some VNFs are frequently used for accommodating service-chain requests, and are regarded as core VNFs. Such a situation occurs when the VNFs are well-implemented, which is sufficient to be connected with many other VNFs. The other functions are regarded as periphery VNFs, which are used only for a specific service-chain request and implemented sufficiently to be connected with specific VNFs, such as receiving the process result of a VNF as input and passing it to another VNF as output. Fig. 2 illustrates the NFV system with core/periphery functions.

Such a VNF classification is based on a core/periphery structure, where the core part does not change despite the changes in service requests and mediates the connection of other system parts. The periphery part has higher variability and absorbs changes in service requests. Core VNFs are used to accommodate multiple service-chain requests and should not be changed frequently, owing to their generality. Periphery VNFs are used to accommodate service-chain requests that cannot be accommodated by core VNFs alone, and therefore, can absorb changes in service-chain requests. We call a software design that has both core VNFs and periphery VNFs as a Core/ Periphery-Based Design (CPBD).

Because core VNFs can be connected to other VNFs, they have more opportunity to accommodate service-chain requests. Preparing many core VNFs will lead to lesser development costs of periphery VNFs for accommodating new service-chain requests. This is because most of the service functionalities would be provided by core VNFs. However, the development cost of each core VNF will be higher than that of each periphery VNF because core VNFs should be generalized to be connected with other VNFs. Based on this observation, we model the development costs of NFV software systems, as presented in Sec. III-B, and compare them with those of CPBD, as presented in Sec. III-C.

B. COST DEFINITIONS

Let us consider an NFV system that accommodates n service-chain requests and the j-th service-chain request requires $k(j)$ VNFs on average. An NFV software system has $f_{all}(n)$ VNFs, which is the sum of the number of core VNFs, $f_c(n)$, and periphery VNFs, $f_p(n)$:

$$f_{all}(n) = f_c(n) + f_p(n). \tag{1}$$

The development cost of an NFV software system, $c_{all}(n)$, is

$$c_{all}(n) = \sum_{i=0}^{f_c(n)} c_c(i) + \sum_{j=0}^{n} k_p(j)c_p(j), \tag{2}$$

where $f_c(n)$ is the number of core VNFs and $c_c(i)$ is i-th core VNF's development cost. Moreover, the j-th service-chain request requires $k_p(j)$ periphery VNFs, and $c_p(j)$ is the development cost of each $k_p(j)$ periphery VNF. In Eq. (2), the first term represents the sum of the development costs of the core VNFs and the second term represents that of periphery VNFs because each periphery VNF serves only one service-chain request and the number of periphery VNFs equals $\sum_{j=1}^{n} k_p(j)$.

The variable $c_c(i)$ increases because of the ability to connect with many other VNFs, such as already implemented core VNFs. Thus,

$$c_c(i) = \alpha i, \tag{3}$$

where the parameter α determines how the development cost of newer core VNFs increases with the number of core VNFs.

Implementing more core VNFs decreases $c_p(j)$ because more service functionalities can be provided by the core VNFs as compared to the periphery VNFs. Thus,

$$c_p(j) = \exp(-\beta f_c(j)), \tag{4}$$

where β determines how the development cost of a periphery VNF decreases as the number of core VNFs increases. For example, if a firewall is implemented as a periphery, its development cost can be reduced using core VNFs, such as pattern matching and session management. Such cases occur more frequently as the number of core VNFs increases. Note that not all implemented core VNFs can serve for a service-chain request, thus Eq.4 forms negative exponential.

Then, we represent $k_p(j)$ by $f_c(j)$ to observe the change in $c_{all}(n)$ against $f_c(n)$. We introduce a parameter γ ($0 < \gamma < 1/f_c(j)$), which represents how often the $f_c(j)$ core VNFs are repeatedly used among the service-chain requests, and $k_c(j)$ is written as

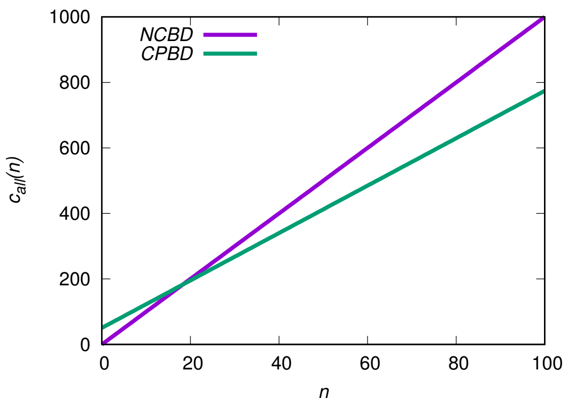

FIGURE 3: The development cost for two design scenarios

$$k_c(j) = k(j)\gamma f_c(j). \tag{5}$$

Then, $k_p(j)$ is obtained as k=$k_c(j)$ + $k_p(j)$;

$$k_p(j) = k - k\gamma f_c(j). \tag{6}$$

C. BENEFIT OF CPBD FOR LONG-TERM DEVELOPMENT COSTS OF NFV SOFTWARE SYSTEM

We consider a scenario in which n increases from 0 to 100, which indicates that new networking services emerge dynamically over time. In this section, we set $k = 10$, $\alpha = 0.01$, $\beta = 0.001$, and $\gamma = 0.002$.

We examine two design scenarios – noncore based design (NCBD) and CPBD – and discuss the development of each scenario by comparing $c_{all}(100)$ values of both NCBD and CPBD. NCBD uses only periphery VNFs to accommodate services without designing and developing core VNFs. That is, NCBD maintains $f_c(n) = 0$ regardless of the value of n. CPBD designs and develops 100 core VNFs even when n=0; that is, none of the service-chain requests are accommodated. Setting $k = 10$ and $\gamma = 0.002$ means that 2 among the 100 core VNFs are used, on average, to accommodate each future service-chain request.

Figure 3 shows the development cost of the NFV system $c_{all}(n)$ for each n. Note that the figure does not consider the addition of core VNFs; that is, $f_c(n)$ is always 100. The figure shows that the $c_{all}(0)$ of CPBD is 50 times higher than that of NCBD. This is because CPBD requires more costs to design and develop core VNFs before accommodating the service-chain request. However, CPBD can reduce the development cost by 23% compared to NCBD. This result suggests that CPBD can reduce long-term development costs by using the developed core VNFs to accommodate future service requests.

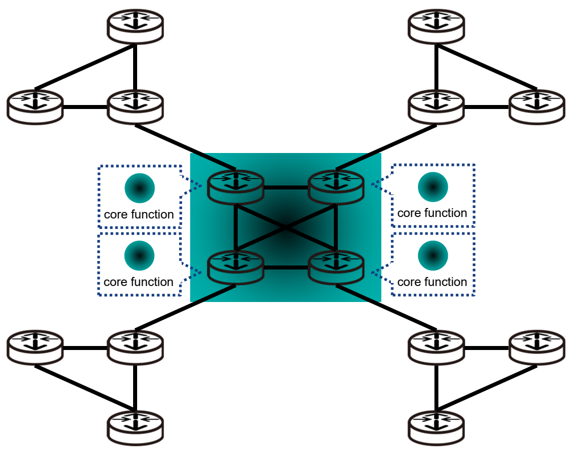

FIGURE 4: Example of CLCP

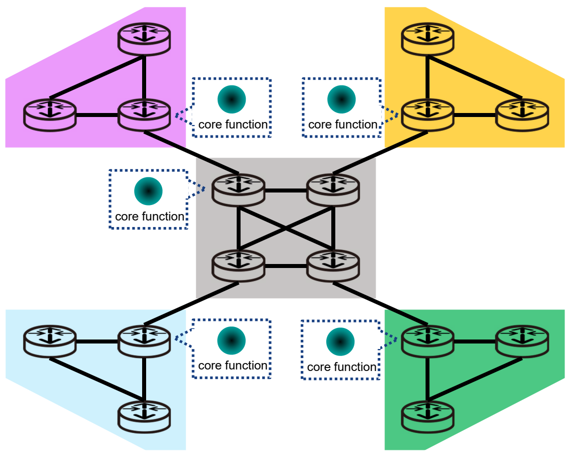

FIGURE 5: Example of GDCP

IV. PLACEMENT METHODS OF CORE/PERIPHERY VNFS

A. PLACEMENT ALGORITHMS FOR A CORE/PERIPHERY-BASED SOFTWARE SYSTEM

In the previous section, we revealed that CPBD reduces long-term development costs. Our next concern is where to deploy the core and peripheral VNFs in the physical network.

Because core VNFs are developed such that other VNFs can be reused, their placement is the most crucial problem in accommodating new service requests with lesser long-term costs. A method for obtaining a suitable placement of core VNFs is to solve the optimization problem that minimizes the deployment cost each time a service-chain request arises. Here, the deployment cost is the cost to place and connect VNFs, and not the development cost used in the previous section. Minimizing the long-term deployment costs is problematic because the placement of core VNFs affects the additionally placed VNFs for future service requests. Therefore, we use the following two heuristics, which focus on the topology of the physical network, and compare them.

- Duplications of core VNFs are placed at the center of the physical network. (CLCP: Center-Located Core/Periphery placement Policy)
- Duplications of core VNFs are distributed over the physical network. (GDCP: Geographically-Distributed Core/Periphery placement Policy)

Figures 4 and 5 illustrate the placement of core VNFs in cases of CLCP and GDCP, respectively, which are detailed in the following sections.

1) Notations

Table 1 shows the notations used in CLCP and GDCP.

The physical network is represented as $G = (V, E)$, where V is the node set and E is the link set. Given a G, a path set P that comprises the shortest paths between each source destination pair is calculated. Here, $P_{u,v} \in P$ represents the shortest paths between each source node $u \in V$ and each destination node $v \in V$.

TABLE 1: Table of notations

V	node set
E	set of links
P	set of all pre-calculated shortest path
$P_{u,v}$	set of shortest paths between $u, v \in V$, and $P_{u,v} \in P$
t	time slot index
T	max size of time slot
$C_v(t)$	remined node resources for each node $v \in V$
$B_e(t)$	remined bandwidth resources for each link $e \in E$
$\hat{c}$	node resource consumption when VNF is placed on node
R	set of all service-chain requests ($r = \{s_r, d_r, b_r, n_r\}, r \in R$)
R_t	set of service-chain requests at each time slot t
s_r	source node of $r \in R$
d_r	destination node of $r \in R$
b_r	bandwidth consumption when link is used for accommodating $r \in R$
$\vec{n}_r$	service-chain of $r \in R$
M	set of all VNF
X	core VNF set
Y	periphery VNF set
w_x	the number of duplications of $x \in X$
$U_{m,v}$	the remaining number of service-chain requests that can use a VNF m placed to node v at each time t
α	deployment cost to place a VNF on a node

In each time slot t, $C_v(t)$ represents the remaining node resources for each node $v \in V$ and $B_e(t)$ represents the remaining bandwidth resources for each link $e \in E$. Note that $C_v(0)$ and $B_e(0)$ represent the initially allocated node and bandwidth resources, respectively.

In each time slot t, λ type of service-chain requests are generated, and the required VNFs belonging to the set of all VNFs, M, are placed in the physical network. When VNF $m \in M$ is placed on a node v, $\hat{c}_{m,v}$ node resources are consumed from $C_v(t)$. In this paper, we assume that $\hat{c}_{m,v}$ is uniform for any VNF $m \in M$ and node $v \in V$, and denote it by $\hat{c}$. Moreover, the deployment cost, $\alpha_{m,v}$, is required to place VNF $m \in M$ on a node v. We also assume that $\alpha_{m,v}$ is uniform for any VNF $m \in M$ and node $v \in V$, and denote it by α. Note that the total deployment cost is the sum of $\alpha_{m,v}$ and increases with the time slot t.

R_t is the set of new service-chain requests at each time slot t, and R is a cumulative set of R_t over the time slot. That is, $R = R_1, R_2, \ldots, R_T$. Each service-chain request $r \in R$ is represented as $r = \{s_r, d_r, b_r, \vec{n}_r\}$, where $s_r \in V$ is the source node of a service-chain request r, $d_r \in V$ is the destination node of r, and b_r indicates the bandwidth resources consumed by $B_e(t)$ of each link $e \in E$.

Each VNF $m \in M$ is classified into either the core VNF or the periphery VNF. Denoting X as the core VNF set and Y as the periphery VNF set, X and Y satisfy $X \cup Y = M$ and $X \cap Y = \emptyset$. For each core VNF $x \in X$, w_x represents the number of duplications of x placed in the physical network. $U_{m,v}(t)$ is the remaining number of service-chain requests that can use a VNF m placed at node v without processing overhead. When VNF $m \in M$ is placed on node $v \in V$, there is a limitation on running VNF m on that node. We denote $U_{m,v}(t)$ as the remaining number of service-chain requests that can use a VNF m placed on node v. $U_{m,v}(0)$ denotes the maximum number of service-chain requests that can use a VNF m on node v.

2) CLCP

Nodes located at the geographic center of the physical network have more opportunities for paths to go through. Placing core VNFs on such nodes increases the opportunity for them to accommodate many service-chain requests.

CLCP places the duplication of core VNFs on nodes in descending order of their *efficiency* [14], which is a metric for measuring how efficiently information exchange is performed on a node. A node with a high *efficiency* has a short hop count from/to other nodes, and is located at the center of the physical network.

Algorithm 1 shows the CLCP core placement algorithm of CLCP. For a loop from line 2 to line 10, the core VNFs are placed. From lines 5 to 7, we obtain a node v that has the highest *efficiency* to place x considering the resource restriction of the node. When placing the core VNF x on node v, $\hat{c}$ resources are consumed, and the remaining resource, $C_v(t)$, decreases by $\hat{c}$. When $C_v(t)$ is lower than $\hat{c}$, VNF x cannot be placed on v. When the node v, which exhibits the highest efficiency, does not satisfy the node resource restriction or x has already been placed on v, Algorithm 1 sets v to a node with the next highest *efficiency*. This process of placing the core VNF x is repeated w_x times, which is the number of duplications of the core VNF x.

Next, we explain where to place the periphery VNFs. Given the placements of core VNFs by Algorithm 1, we place and connect periphery VNFs sequentially from the source node to destination node via nodes where core VNFs are deployed. Note that a service-chain is composed of periphery-core-periphery VNFs as depicted in Fig. 2. Thus, there are three path segments between the source and destination nodes: a segment for periphery VNFs (source side), that for core VNFs, and that for periphery VNFs (destination side).

Algorithm 1 Core placement algorithm of CLCP

```
Input: $G = (V, E)$, $X$, $w_x$
 1: if $t = 1$ then
 2:   for each $x \in X$ in descending order of $\hat{c}$ do
 3:     $v \Leftarrow$ node with the highest efficiency
 4:     for loopcounter $= 1$ to $w_x$ do
 5:       while $\hat{C}_v(t) < \hat{c}$ or $x$ has already been placed to
          $v$ do
 6:         $v \Leftarrow$ node with a next higher efficiency
 7:       end while
 8:       place $x$ to $v$
 9:     end for
10:   end for
11: end if
```

Algorithm 2 calculates a set of available paths for each path segment, P_{avail}, and determines the possible nodes on which core VNFs can be placed, under the resource restriction of the node in line 7. In addition, it considers the bandwidth resource restriction in line 13. In using link e for accommodating a service-chain request r, b_r bandwidth resources are consumed by $B_e(t)$, which is defined as the resources remaining on link e. b_r should not exceed $B_e(t)$; otherwise, r cannot use e because of the lack of bandwidth resources. When $U_{m,v}$ is 0 for all m and v, or Algorithm 2 cannot obtain P_{avail} with the remaining bandwidth resources larger than b_r, a service-chain request r is rejected.

Algorithm 3 places the periphery VNFs along the P_{avail} obtained by Algorithm 2. However, when the hop count between the nodes in P_{avail} is too short, periphery VNFs cannot be placed because there are fewer opportunities to find a node to run them. To avoid such a situation, we consider detour paths other than the shortest path for each path segment from lines 11 to 16 . In more detail, when m is placed on v, which deploys core VNFs and does not retain sufficient resources, a detour path, p, is selected using the neighboring nodes of v.

3) GDCP

By placing the core VNFs at the center of a physical network, as in CLCP, they can be used more frequently for accommodating many service-chain requests. However, using the central nodes that deploy core VNFs and their neighbors leads to resource exhaustion because such nodes are intensively used for accommodating service-chain requests. Therefore, we examine another placement algorithm that exhibits a distributed placement of core VNFs on the physical network.

GDCP divides the physical network into clusters to maximize modularity [15], and place duplication of core VNFs in each cluster. Modularity is a metric that reflects the density of the cluster density; a higher modularity leads to an increased ratio of the number of links between clusters and that in each cluster. Algorithm 4 shows the core placement algorithm of GDCP. In line 2, we divide the physical network

Algorithm 2 Algorithm to obtain available paths to place periphery VNFs

```
Input: $G = (V, E)$, $X$, $P$, $r$
Output: $P_{avail}$
 1: $P_{avail} \Leftarrow \emptyset$
 2: $s'_r \Leftarrow s_r$
 3: for each $m \in \vec{n}_r$ do
 4:     if $m \notin X$ then
 5:         continue
 6:     end if
 7:     if $U_{m,v}(t) = 0$ for all VNF $m$ and node $v$ then
 8:         reject $r$
 9:     end if
10:     $d'_r \Leftarrow$ node with the shortest path from $s'_r$, VNF $m$, and $\hat{r}_m(t) < \tilde{r}_m$
11:     Obtain $p_{s'_r,d'_r} \in P_{s'_r,d'_r}$ with the highest remaining bandwidth resources
12:     for each $e \in p_{s'_r,d'_r}$ do
13:         if $\hat{B}_e(t) < b_r$ then
14:             reject $r$
15:         end if
16:     end for
17:     add $p_{s'_r,d'_r}$ to $P_{avail}$
18:     $s'_r \Leftarrow d'_r$
19: end for
20: $d'_r \Leftarrow d_r$
21: Repeat from lines 11 to 17
```

into clusters using the Louvain algorithm [16] and obtain the number of clusters ζ. The Louvain algorithm can obtain optimized ζ to maximize modularity. Thus, line 3 sets w_x, which is the number of duplications of the core VNF x, to ζ. Finally, these duplications are distributed to each cluster, and placed on the node having the highest efficiency in the cluster in line 6. For the placement of periphery VNFs on GDCP, Algorithm 3 is used.

B. A MODEL FOR SERVICE-CHAIN REQUESTS

In the simulation for evaluating the algorithms, service-chain requests are dynamically generated. Each request comprises k VNFs, which is the sum of the numbers of core VNFs, k_c, and periphery VNFs, k_p. Here, k_c and k_p are obtained using the Eqs. 5 and 6, respectively.

When the time slot t is incremented, λ new types of service-chain requests are generated. Both the source and destination nodes of each service-chain request are selected by using a uniform random. k_c core VNFs are selected from all $|X|$ types of core VNFs using a uniform random. Note that service-chain requests do not have duplicate VNFs.

C. RESULTS

We perform simulations to evaluate the CLCP and GDCP. AaP [5] is used as a placement policy for comparison. In this section, to reveal the basic characteristics of each

Algorithm 3 The CLCP periphery placement algorithm

```
Input: $G = (V, E)$, $X$, $P$, $R$
 1: for each $r \in R$ in descending order of $b_r$ do
 2:     call Algorithm 2 and obtain $P_{avail}$
 3:     $p \Leftarrow$ first path of $P_{avail}$
 4:     $v \Leftarrow$ source node of $p$
 5:     for each $m \in \vec{n}_r$ do
 6:         if $m \in X$ then
 7:             $p \Leftarrow$ next path of $P_{avail}$
 8:             $v \Leftarrow$ source node of $p$
 9:             continue
10:         end if
11:         if $v$ owns any core VNF then
12:             while $C_v(t) < \hat{c}$ do
13:                 $v \Leftarrow$ neighbor node of $v$ with the highest remaining node resource $C_v(t)$
14:             end while
15:             add a detour that can reach $v$ to $p$
16:         end if
17:         while $C_v(t) < \hat{c}$ do
18:             if $v$ is the destination node of $p$ then
19:                 reject $r$
20:             end if
21:             $v \Leftarrow$ next node of $p$
22:         end while
23:         place $m$ to $v$
24:     end for
25: end for
```

Algorithm 4 Core placement algorithm of GDCP

```
Input: $G = (V, E)$, $X$
 1: if $t = 1$ then
 2:     divide $G$ into $\zeta$ cluster by using Louvain algorithm
 3:     $w_x \Leftarrow \zeta$
 4:     for each $x \in X$ do
 5:         for $loopcounter = 1$ to $w_x$ do
 6:             Place a duplication of core VNF $x$ to the node that has the highest efficiency in the $loopcounter$-th cluster
 7:         end for
 8:     end for
 9: end if
```

placement policy, we first perform simulations when the resources are infinite. Next, we consider the case in which only node resources are finite and become a bottleneck for accommodating service-chain requests. Finally, we show the simulation results when both the bandwidth resources and node resources are finite.

1) Results with no resource restriction

We use a 7×7 grid network as the physical network, where both the initial node resources $C_v(0)$ and bandwidth resources $B_e(0)$ are infinite. The number of VNFs

TABLE 2: Average hop counts of paths used by each placement policy

setting	placement policy	average hop count
$\vert X\vert = 500, \gamma = 0.001$	CLCP	6.65
	GDCP	7.15
	AaP	6.50
$\vert X\vert = 500, \gamma = 0.0015$	CLCP	6.63
	GDCP	6.70
	AaP	6.49

comprising a service-chain request, that is, chain length k, is decided by using a uniform random from the range $[4, 8]$. When t is incremented, new $\lambda = 10$ service-chain requests are generated. Because the Louvain algorithm divide the 7×7 grid network into five clusters, we set w_x of both CLCP and CDCP to 5. The deployment cost, α, is decided using a uniform random from the range $[1, 1.2]$.

Figure 6 shows the deployment cost of each placement policy when CLCP and GDCP place $|X| = 500$ core VNFs in advance, which are used for accommodations at a frequency of $\gamma = 0.001$. When the resources are infinite, the deployment costs of the CLCP and GDCP are the same, and thus, both are indicated by the CLCP / GDCP line in the figure. At $t \leq 80$, the deployment cost of AaP is lower, but at $80 \leq t$, the deployment cost of CLCP / CDCP reduces below that of AaP. This is because CLCP and GDCP reduce the number of additional VNFs to be placed, by using the already placed core VNFs to accommodate new service-chain requests.

Placing more core VNFs in advance reduces the deployment costs of CLCP and GDCP. Figure 7 shows the deployment cost of each placement policy at $|X| = 700$; that is, CLCP and GDCP place more core VNFs in advance. The deployment costs of CLCP/GDCP at $t = 150$ are 12.76% less than those of AaP. This is because placing more core VNFs increases the opportunity to use them to accommodate a service-chain request and reduce the opportunity to use periphery VNFs. Moreover, the parameter γ affects the deployment costs of CLCP and GDCP. As γ increases, the core VNFs are used more frequently to accommodate the service-chain requests.

Table 2 shows the average hop count of paths used by each placement policy to accommodate the service-chain requests. The average hop count of CLCP is smaller than that of GDCP. CLCP places core VNFs at the center of a physical networ, which has, on average, a short hop count from any node. In addition, it tends to use shorter paths through nodes that deploy core VNFs as compared to GDCP. Note that AaP is the placement policy with the shortest hop count because it uses the shortest path from the source node to destination node, while CLCP and GDCP use a detour path.

2) Results with restrictions on computing and bandwidth resources

First, we consider a case in which only the node resources are finite and become a bottleneck to accommodate service-chain

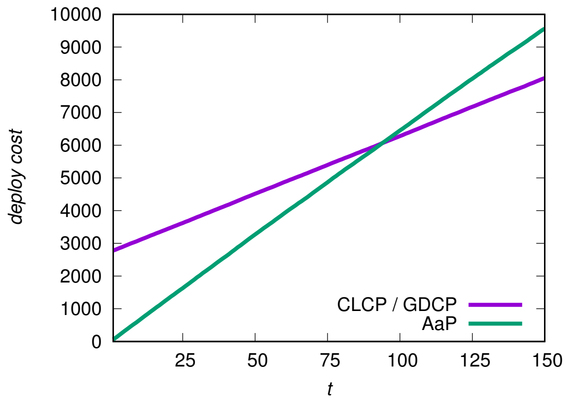

FIGURE 6: Deployment costs of each placement policy ($|X| = 500, \gamma = 0.001$)

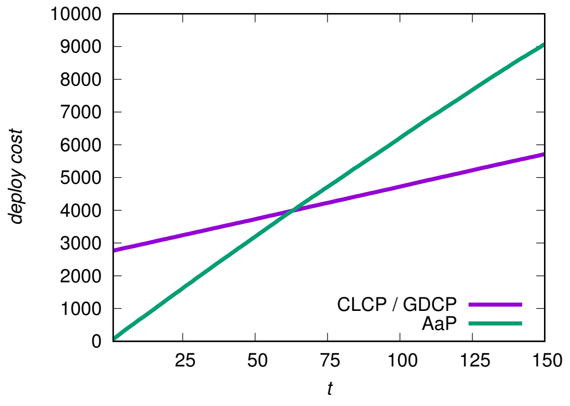

FIGURE 7: Deployment costs of each placement policy ($|X| = 700, \gamma = 0.001$)

requests. A 7×7 grid network is again used for the physical network. We set the initial node resources $C_v(0)$ to 100 for each node $v \in V$ and the initial bandwidth resource $B_e(0)$ to infinity for each link $e \in E$. The node resource consumed by a placed VNF, $\hat{c}$, is decided using a uniform random from the range $[0.4, 1]$. The upper number of service-chain requests that can use a VNF m placed at node v without processing overhead, $U_{m,v}(0)$, which is the maximum number of service-chain requests that can use a VNF m placed at node v, is set using a uniform random from the range $[4, 40]$. When the time slot t is incremented, $\lambda = 10$ service-chain requests are newly generated. The chain length, k, is decided using a uniform random from the range $[4, 8]$, and the deployment cost for each VNF, α, is decided using a uniform random from the range $[1, 1.2]$.

Figure 8 shows the deployment cost of each placement policy when $|X| = 500$, $\gamma = 0.001$ and $w_x = 5$. In the figure, the deployment costs per service-chain request accommodated by CLCP and GDCP decreases as t increases. This result indicates that CLCP and GDCP reduce the deployment cost by repeatedly using the core VNFs.

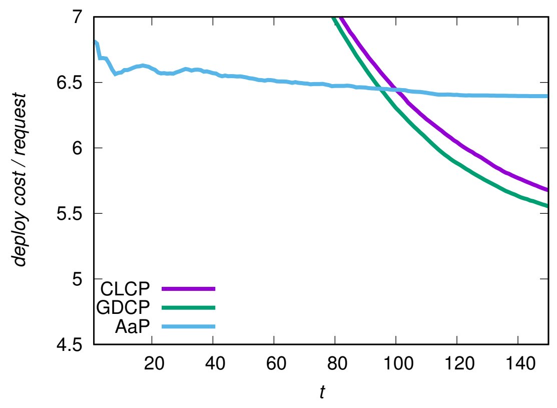

FIGURE 8: Deployment cost: $(C_v = 100, B_e$ is infinite, $|X| = 500, \gamma = 0.001, w_x = 5$

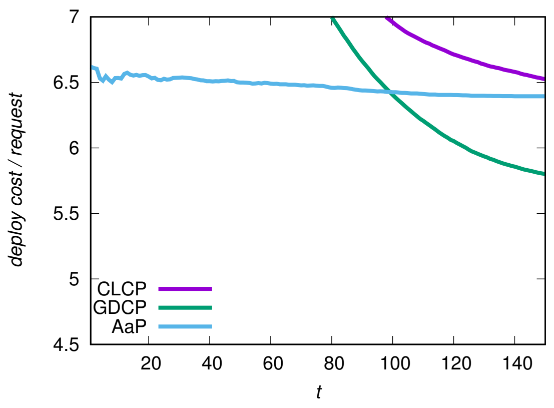

FIGURE 10: Deployment cost: $C_v = 100, B_e = 500, |X| = 500, \gamma = 0.001, w_x = 5$

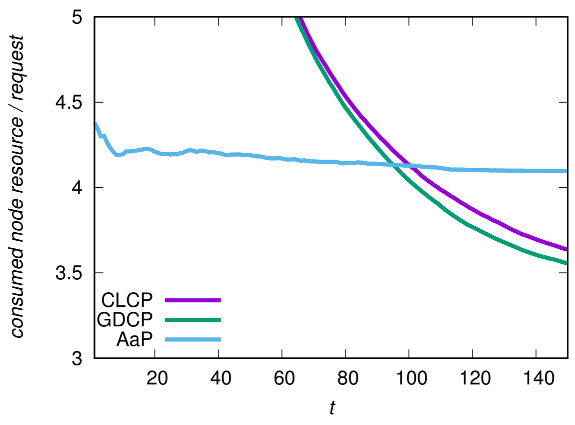

FIGURE 9: Amount of node resources consumed by the placed VNFs: $C_v = 100, B_e$ is infinite, $|X| = 500, \gamma = 0.001, w_x = 5$

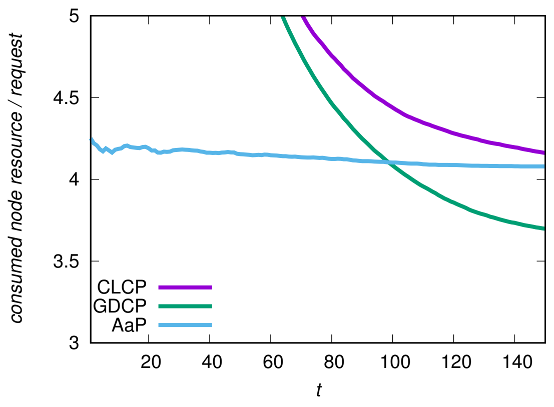

FIGURE 11: Amount of node resources consumed by the placed VNFs: $C_v = 100, B_e = 500, |X| = 500, \gamma = 0.001, w_x = 5$

Note again that a reduction in the deployment costs appears when many service-chain requests are accommodated ($t \geq 90$). Otherwise, the initial costs to deploy the core VNFs are significant; the deployment cost of AaP at t = 1 is 6.82, whereas that of CLCP and GDCP is 278.69.

The deployment costs of GDCP are lower than those of CLCP, because they accommodate different numbers of service-chain requests due to the node resource restriction. CLCP places core VNFs only on nodes at the center of the physical network and intensively uses these nodes to accommodate service-chain requests; thus, node resource restrictions are likely to occur. In contrast, GDCP distributes core VNFs on the physical network, and thus, can use geographically distributed nodes and accommodate more service-chain requests than CLCP. This can be observed from Figure 9, which shows the amount of node resources consumed by the placed VNFs per accommodated service-chain request.

Next, we consider the case in which both the bandwidth resources and node resources are finite. Other settings are the same as those in the case where only the node resources are finite. Figure 10 shows the deployment cost of each placement policy. We set B_e to 500, keeping all other parameters same as those in Fig. 8. Looking at $t = 150$ in Figure 10, the deployment cost per service-chain request by GDCP exhibits the lowest value. The deployment cost of CLCP is 12.99% larger than that in Figure 8, while that of GDCP is only 4.26% larger. Figure 11 shows the amount of node resources consumed by the placed VNFs per accommodated service-chain request. In CLCP, nodes at the geographic center of the physical network are intensively used; thus, the links that can reach these nodes are also intensively used. As compared to CLCP, GDCP can use geographically distributed nodes and incurs fewer bandwidth resource restrictions. We have conducted simulations on 9 $\times$ 9 grid network, 49-node BA networks, and 40-node ternary trees, which are not shown here. Similar tendencies are observed for AaP/CLCP/GDCP.

Our results show that GDCP mostly reduces the deployment costs for CPBD when there are many service-chain requests. As CPBD also reduces the development costs, the core/periphery-based software design and distributed placement are suitable for NFV systems for accommodating future service-chain requests.

V. CONCLUSION

In this paper, we investigated the software design and placement method of VNFs to reduce the long-term development and deployment costs against the change in service requests. We first considered designing an NFV system based on a core/periphery structure, and repeatedly used core VNFs to accommodate future service requests. Our evaluation results indicated that such software design based on the core/periphery structure can accommodate service-chain requests with lower development costs than that without core VNFs. Moreover, we investigated where to place the core VNFs in the physical network by examining CLCP and GDCP. Our results showed that GDCP is the best placement policy that can accommodate many service-chain requests with low deployment cost, and the difference between GDCP and CLCP is significant when there are resource constraints on nodes and/or links.

In this paper, the incremental development of VNFs was considered. However, in reality, some of core VNFs would be no longer necessary as the time proceeds, because of the changes in service-chain requests. One of the future works is to consider removing the not required VNFs from the nodes.

REFERENCES

[1] S. Kim, S. Park, Y. Kim, S. Kim, and K. Lee, "VNF-EQ: Dynamic placement of virtual network functions for energy efficiency and QoS guarantee in NFV," *Cluster Computing*, vol. 20, no. 3, pp. 2107–2117, Sep. 2017.

[2] Y. Nam, S. Song, and J.-M. Chung, "Clustered NFV service chaining optimization in mobile edge clouds," *IEEE Communications Letters*, vol. 21, no. 2, pp. 350–353, Oct. 2017.

[3] P. Csermely, A. London, L.-Y. Wu, and B. Uzzi, "Structure and dynamics of core/periphery networks," *Journal of Complex Networks*, vol. 1, no. 2, pp. 93–123, Oct. 2013.

[4] V. Miele and R. R.-J. D. P. Vázquez, "Core-periphery dynamics in a plant-pollinator network," *Journal of Animal Ecology*, pp. 1–8, Mar. 2020.

[5] Q. Sun, P. Lu, W. Lu, and Z. Zhu, "Forecast-assisted NFV service chain deployment based on affiliation-aware vNF placement," in *Proceedings of IEEE Global Communications Conference (GLOBECOM)*, Dec. 2016, pp. 1–6.

[6] M. Richards, *Software Architecture Patterns*. O'Reilly Media, Inc., Feb. 2015.

[7] A. MacCormack, J. Rusnak, and C. Y. Baldwin, "Exploring the structure of complex software designs: An empirical study of open source and proprietary code," *Management Science*, vol. 52, no. 7, pp. 1015–1030, Jul. 2006.

[8] A. MacCormack, "The architecture of complex systems: Do "core-periphery" structures dominate?" *Academy of Management Proceedings*, vol. 2010, no. 1, pp. 1–6, Nov. 2010.

[9] H. P. Breivold, I. Crnkovic, and P. J. Eriksson, "Analyzing software evolvability," in *32nd Annual IEEE International Computer Software and Applications Conference*. IEEE, Aug. 2008, pp. 327–330.

[10] A. MacCormack and D. J. Sturtevant, "Technical debt and system architecture: The impact of coupling on defect-related activity," *Journal of Systems and Software*, vol. 120, pp. 170–182, Oct. 2016.

[11] W. Hasselbrin, "Microservices for scalability: Keynote talk abstract," in *Proceedings of the 7th ACM/SPEC on International Conference on Performance Engineering*. ACM, Mar. 2016, pp. 133–134.

[12] D. Sabella, V. Sukhomlinov, L. Trang, S. Kekki, P. Paglierani, R. Rossbach, X. Li, Y. Fang, D. Druta, F. Giust, L. Cominardi, W. Featherstone, B. Pike, and S. Hadad, "Developing software for multi-access edge computing," *ETSI, White Paper*, vol. 20, no. 2, Feb. 2019.

[13] M. Wajahat, B. Balasubramanian, A. Gandhi, G. Jung, and S. P. Narayanan, "A model-driven graybox approach to rehoming service chains," in *Proceedings of IEEE 26th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, Sep. 2018, pp. 116–122.

[14] V. Latora and M. Marchiori, "Efficient behavior of small-world networks," *Physical Review E 87, 19801*, no. 19, Oct. 2001.

[15] M. E.J. Newman, "Modularity and community structure in networks," *Proceedings of the national academy of sciences*, vol. 103, no. 23, pp. 8577–8582, Jun. 2006.

[16] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *Journal of statistical mechanics: theory and experiment*, vol. 2008, no. 10, p. P10008, Oct. 2008.

YUKI TSUKUI received the B.E. degree and M.E. degree in information and computer science from Osaka University, Osaka, Japan, in 2018 and 2020, respectively.

SHIN'ICHI ARAKAWA (M'02) received his M.E. and D.E. in informatics and mathematical science from Osaka University, Osaka, Japan, in 2000 and 2003, respectively.

From August 2000 to March 2006, he was an Assistant Professor with the Graduate School of Economics, Osaka University. In April 2006, he moved to the Graduate School of Information Science and Technology, Osaka University. He has been an Associate Professor since October 2011.

Dr. Arakawa is a member of the IEEE and the IEICE.

SHIORI TAKAGI received the B.E. degree and M.E. degree in information and computer science from Osaka University, Osaka, Japan, in 2018 and 2020, respectively. She is currently pursuing the M.E. degree in information science and technology at Osaka University.

MASAYUKI MURATA (M'89) received the M.E. and D.E. in information science and technology from Osaka University, Osaka, Japan, in 1984 and 1988, respectively.

In April 1984, he joined the Tokyo Research Laboratory IBM Japan as a Researcher. From September 1987 to January 1989, he was an Assistant Professor with the Computation Center, Osaka University. In February 1989, he moved to the Department of Information and Computer Sciences, Faculty of Engineering Science, Osaka University. From 1992 to 1999, he was an Associate Professor with the Graduate School of Engineering Science, Osaka University, and since April 1999, he has been a Professor. He moved to the Graduate School of Information Science and Technology, Osaka University, in April 2004.

Prof. Murata is a fellow of the IEICE and a member of the IEEE, the Association for Computing Machinery (ACM), the Internet Society, and the Information Processing Society of Japan.