Master's Thesis

Title

On network function virtualization for dynamically changing service requests based on a Core/Periphery structure

Supervisor

Professor Masayuki Murata

Author

Yuki Tsukui

February 5th, 2020

Department of Information Networking

Graduate School of Information Science and Technology

Osaka University

Master's Thesis

On network function virtualization for dynamically changing service requests based on a Core/Periphery structure

Yuki Tsukui

Abstract

NFV (Network Function Virtualization) is a system which provides application services by connecting VNFs (Virtual Network Functions), and is expected to accommodate new service requests through a little development of new VNF and connecting with exiting VNFs. However, preparing a lot of VNFs in advance leads to increase a cost of NFV system. On the contrary, preparing a small set of VNFs leads to increase a development cost of new VNFs required for accommodating future service requests. Thus, it is important to design the NFV system that reduces the current and future system costs when service applications become diverse.

In this thesis, we first investigate the design principles that reduce the cost to design and develop VNFs for accommodating new service requests. For this purpose, we introduce CPBD (Core/Periphery Based Design) that utilizes a concept of core/periphery for developing VNFs. In the CPBD, "core" VNFs are developed in advance, and repeatedly used for accommodating future service requests. While "core" VNFs are common to current and future service requests, "periphery" VNFs are developed and customized to each service request. Next, we investigate placement policies of VNFs for CPBD to fully utilize the nature of core/periphery structure of VNFs. We examine the CLCP (Center-Located Core/Periphery placement) policy and the GDCP (Geographically-Distributed Core/Periphery placement) policy, and evaluate the long-term cost of NFV system under resource restrictions to run VNFs. Our results show that CPBD reduces the long-term cost about design and development VNFs by about 23 % compared to the design with no core VNFs. Moreover, in the case of no resource restrictions, both the CLCP and the GDCP reduce the long-term costs about place and connect VNFs by 15.83 % compared

to the existing VNF placement algorithm. With resource constraints, the GDCP reduce the long-term costs by 11.10 % over the CLCP.

Keywords

NFV (Network Function Virtualization)

VNF (Virtual Network Function)

Core/periphery structure

Software design

VNF placement

Microservices

Service chain

Contents

List of Figures

List of Tables

1 Introduction

Recent years, devices communicating with networks grow popular worldwide, and users enjoy various application services. As the service demands become diverse more and more, a system which accommodates various service requests at low cost is expected. NFV (Network Function Virtualization) [1, 2] is the system that implements VNFs (Virtual Network Functions), such as firewall and proxy server, by software and is expected to accommodate various service requests by connecting VNFs over a network in a flexible manner.

Many literatures of NFV have investigated VNFs placement algorithms to optimize a resource usage with given service requests. For example, in Ref. [3], a genetic algorithm is used to minimize the power consumption without violating delay requirements of service requests. Ref. [4] tries to minimize end to end delay of service requests by placing VNFs assuming that the usage of VNFs is governed by Zipf's law. These literatures assume that VNFs have already been designed and developed and also assume that service requests are given in advance. However, for an operational perspective of NFV system, service requests is not always accommodated by existing VNFs, and the NFV system may need to develop new VNFs to accommodate new kinds of service requests. In such a situation, when VNFs are not properly designed, new VNFs are added frequently depending on changes in service requests, and development cost increases severely. That is, a proper software design of VNFs is required to reduce a long-term cost, which is a cost to accommodate the current and future service requests, of NFV system. After a proper software design of VNFs is obtained, the placement problem of their VNFs is our next concern. Above existing placement methods [3, 4] may frequently change the placement of VNFs, because they did not concern about addition of VNFs depending on changing service requests. To reduce the long-term cost of NFV system, a proper placement of VNFs is required to reduce opportunities of changing placement, such as adding, moving, and removing VNFs [5]. In this thesis, we investigate software design and placement method of VNFs, and reduce long-term development cost and deployment cost against changes in service requests.

For the software design of VNFs, we introduce a core/periphery structure [6, 7]. The

core/periphery structure has been used to interpret a stable but flexible behaviors of biological systems. In the system that has the core/periphery structure, a main part of system components forms "core" components, which does not change despite changing a composition of an entire system with time, and other part of system components forms "periphery" components, which change and are mediated by the core components. We introduce a concept of the core/periphery to VNFs of the NFV system, and distinguish VNFs into core VNFs and periphery VNFs. By deploying core VNFs, it is expected that future service requests can repeatedly use core VNFs and requires less amount of new periphery VNFs, which reduces the long-term development cost. Note that, the larger the number of core VNFs becomes, the more frequently core VNFs are used for accommodating service requests. However, the larger core VNFs will lead to increase the cost to prepare core VNFs. In this thesis, we develop a model to represent the cost of NFV system and reveal the cost impact of setting the number of core VNFs to the long-term development cost.

Next, we investigate how to place VNFs that are designed based on the core/periphery structure. It is expected that the deployment cost will be reduced by properly placing core VNFs in advance so that they can be shared for accommodating multiple service requests. In fact, existing method in Ref. [8] tries to reduce the number of VNFs to place by sharing common VNFs among service requests. In our case, the placement of core VNFs under the resource restrictions is the most important because core VNFs will be commonly used for accommodating the current and future service requests. We investigate placement policies of VNFs for core/periphery based design to fully utilize the nature of core/periphery structure of VNFs. CLCP (Center-Located Core/Periphery placement) policy and the GDCP (Geographically-Distributed Core/Periphery placement) policy are examined for placements of core VNFs, and evaluate the long-term cost of NFV system under resource restrictions to run VNFs.

This thesis is organized as follows. Section 2 states our design and placement problems of NFV systems. Section 3 develops the cost model of NFV systems and show the reduction of the long-term cost by core/periphery-based design. Then, the placement algorithm for core/periphery-based design of VNFs are presented and evaluated in Section 4. Finally, we conclude our thesis in Section 5.

2 Design and Placement Problems of NFV Software Systems

2.1 Design problem

For operation of the NFV system, it is important to design VNFs properly to reduce costs. The NFV system has many VNFs and connects them for accommodating service requests. There are some costs to develop VNFs, and proper software design of VNFs reduces such costs. However, costly software design disturb flexible accommodation for service requests. We show several software design and their cost below.

The monolithic software design has been widely used for software. In the monolithic, multiple components form a single module [9]. These components are designed to accommodate a particular service and to be coupled with specific components. Thus, changing a componet leads to changes of other components, and thus increase a development cost [10–14]. Moreover, such tightly couplings between components make it difficult to use components already developed for accommodating new service. Existing works [10,12] analyze how components of software had been designed and developed in the long-term, such as Linux and Mozilla, and indicate that large-scale refactoring to reduce tightly couplings may contribute to the spread of Mozilla. In conclusion, monolithic has many problems.

In recent years, microservices are getting attention due to the possibility of reducing development cost [9, 14, 15]. In microservices, components are well independent and can be connected to other components to form a variety of services. Components already developed can be used for accommodating future services, thus microservices reduce the number of components and costs to design and develop.

Such discussions on software design have been active in the software engineering field, but have not been fully discussed on the NFV system. In this thesis, we discuss software designs of VNFs that has not attracted enough attention in existing works. To reduce development costs, like microservices, we design "core" VNFs to be used for accommodating new service requests.

2.2 Placement problem

Many literatures about NFV studied placement algorithms that accommodate service requests efficiently. Existing placement algorithm called AaP (Affiliation-aware vNF Placement) tries to reduce the number of VNFs to place by merging service requests [8]. AaP places VNFs on a shortest path between source-destination nodes in order of each merged request, and avoids bandwidth consumption. For example, in accommodating two service requests, such as VNF1 $\rightarrow$ VNF2 $\rightarrow$ VNF3 and VNF4 $\rightarrow$ VNF2 $\rightarrow$ VNF5, AaP merges these requests and assumes them as one service request, such as VNF1 $\rightarrow$ VNF4 $\rightarrow$ VNF2 $\rightarrow$ VNF3 $\rightarrow$ VNF5. Here, VNF2 is shared and the number of placing VNF2 is reduced from 2 to 1. Other methods use genetic algorithm to solve the problem that minimizes power consumption with satisfying service delay requirements [3], and places VNF to physical network based on Zipf's law, which models frequency of use, in order to minimize end to end service time [4]. These literature aim at optimization in terms of power consumption, end-to-end service time, or bandwidth consumption.

However, considering the long-term operation of the NFV system, it may be more important to reduce costs to additionally place VNFs and to change the location of VNF one to another. For accommodating service requests, the NFV system places VNFs to the physical network and connects them in proper order. There are some costs to place and connect VNFs, and these costs are called deployment cost as a whole. Note that deployment cost includes costs to transfer a VM image to node, boot it, consume electric resource to run VNFs, and connect VNF in proper order. In operating the NFV system, service requests may variously change and need VNFs that have not yet been placed and connected. Those additional VNFs need additional deployment costs. Moreover, changes in a placement of VNFs, such as adding, moving, and removing VNFs, suspend the execution of VNFs, and cause a delay in data communications [5]. However, above existing placement method [3,4,8] may frequently change the placement of VNFs, because they did not concern about changing the placement of VNFs depending on variation of service requests. In this thesis, we investigate placement method of VNFs that reduces long-term deployment cost against the change of service requests.

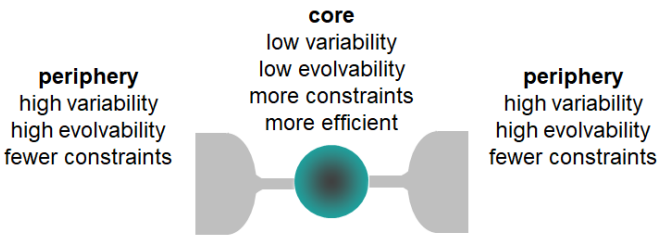

Figure 1: The concept of core/periphery structure

2.3 Approaches with core/periphery structure

In considering the software design of VNFs, we introduce a core/periphery structure [6, 7]. A core/periphery structure has been used to interpret behaviors of biological system, social network, and Internet systems. Fig. 1 shows the basic concept of a core/periphery structure. We distinguish VNFs into core VNFs and periphery VNFs. It is expected that the additional VNFs and the development cost will be reduced by designing VNFs so that core VNFs can be used repeatedly for accommodating future service requests. Moreover, the deployment will be also reduced by properly placing VNFs designed based on a core/periphery structures. In order to reduce deployment cost, we place core VNFs in advance. This is so that core VNF can be used repeatedly for accommodating future service requests, as in the case of software design. In fact, AaP described above tries to reduce the number of VNFs to place by sharing common VNFs among service requests [8].

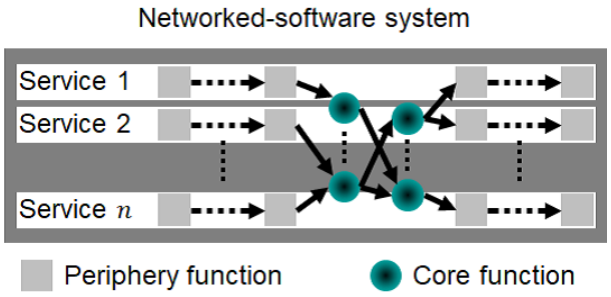

Figure 2: An example of NFV system

3 Core/Periphery-based Design of NFV Systems

3.1 CPBD: Core/Periphery Based Design

The NFV system has many VNFs and accommodates various kinds of service requests by appropriately connecting VNFs. Here, appropriately connecting order of VNFs is called service chain. It is natural that some VNFs are used frequently for accommodating service chain requests. We consider that such a situation occurs when the VNFs are well-implemented which is enough to be connected with many other VNFs. Such VNFs are regarded as core VNFs. The other VNFs are regarded as periphery VNFs, which are used only for one specific service chain requests and implemented just enough to be connected with specific VNFs, such as receiving the process result of a VNF as input and passing the process result to another VNF as output. Fig. 2 illustrates the NFV system with core/periphery VNFs.

Such a classification of VNFs is based on a core/periphery structure. In a core periphery structure, a core part does not change despite changing a composition of a entire system with time, and mediate the connection of other system parts. A periphery part has higher variability and absorbs changes of VNF requests. Then, core VNFs are used to accommodate multiple service chain requests, and should not be changed frequently due to its generalization. Periphery VNFs are used to accommodate service chain requests that can not be accommodated by core VNFs alone, therefore periphery VNFs absorb changes of VNF requests. The software design that has the both of above core VNFs and periphery VNFs is called CPBD: (Core/Periphery Based Design).

Because the core VNFs have an ability to be connected with other VNFs, they have more opportunity to accommodate service chain requests. Preparing many core VNFs will lead to have the less amount of development cost of periphery VNFs for accommodating new service chain requests because most of service's functionalities are provided by the core VNFs. However, development cost of each core VNF will be higher than periphery VNFs because core VNFs should be generalized in order to be connected with many other VNFs. In what follow, we define the development cost of the CPBD NFV system.

3.2 Cost definitions

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

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

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

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

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

$c_c(i)$ increases due to the ability to be connected with many other VNFs, such as core VNFs already implemented. Thus,

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

with a parameter α which determines how the development cost of newer core VNF increases as the number of core VNFs increases.

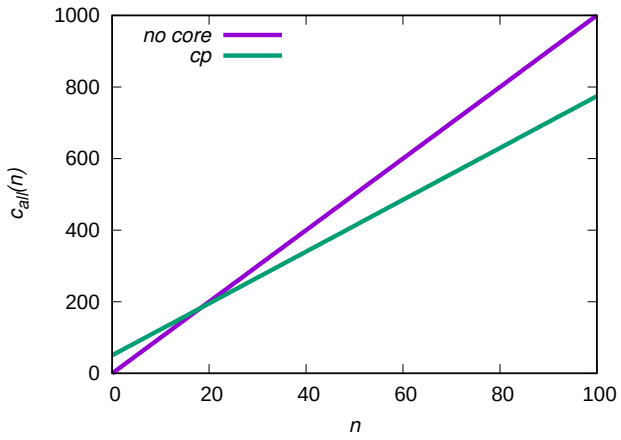

Figure 3: The development cost for two design scenarios

Implementing more core VNFs decreases $c_p(j)$ because more service's functionalities are provided by core VNFs instead of periphery VNFs. Thus,

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

with a parameter β which determines how the development cost of a periphery VNF decreases as the number of core VNFs increases. For example, in implementing a firewall as a periphery, its development cost can be reduced with using core VNFs such as pattern matching or session management. In such the case, the development cost of periphery VNFs is reduced more as the number of core VNFs increases. Note that not all implemented core VNFs can serve as a periphery VNF, thus Eq.4 forms negative exponential.

In order to observe the change of $c_{all}(n)$ by changing $f_c(n)$, we will represent $k_p(j)$ by $f_c(j)$. When j-th service chain request requires $k_c(j)$ core VNFs, $k(j)$ equals $k_c(j) + k_p(j)$. We introduce a parameter γ $(0 < \gamma < 1/f_c(j))$ that represents how often $f_c(j)$ core VNFs are repeatedly used among service chain requests, and assume that,

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

Then, we obtain $k_p(j)$ from following equation;

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

3.3 Results with long-term operations of the NFV system

We consider a scenario that n increases from 0 to 100. The increase of n represents that new service chain requests emerge dynamically over time. In this subsection, we set $k = 10$, $\alpha = 0.01$, $\beta = 0.001$, and $\gamma = 0.002$.

We examined two design scenarios, NCBD (Non-Core Based Design) and CPBD. The NCBD uses only periphery VNFs to accommodate service chain requests without developing core VNFs. This design keeps $f_c(n) = 0$ regardless of n. The CPBD developed 100 core VNFs when none of service chain requests were accommodated, that is, when $n = 0$. These 100 core VNFs are used on average by 2 per future service chain request accommodation, because we set $k = 10$ and $\gamma = 0.002$. Note that our evaluation here do not consider adding core VNFs, thus this design keeps $f_c(n) = 100$ regardless of n

Figure 3 shows the development cost of the NFV system $c_{all}(n)$ for each n. Looking at the results of such the design scenarios, the $c_{all}(0)$ of the CPBD is about 50 higher than that of the NCBD. This is because the CPBD require more costs to develop core VNFs before accommodating the service chain request. However, the CPBD reduces development cost by about 23% compared to the NCBD. This result suggests that the core/periphery structure of the NFV system reduces the development cost of the system.

4 Placement Methods of Designed VNFs

In Section 3, we show that CPBD reduces the long-term development cost by using developed core VNFs to accommodate future service chain requests. Based on these results, we investigate placement algorithm of core VNFs to reduce additional VNFs to place in accommodating new service chain requests.

4.1 Core placement problem

It is expected that properly placing core VNFs in advance reduces deployment cost. When the NFV system accommodate a new service chain request, new VNFs may be placed additionally, and connected in order of requested service chain. Because deployment costs increase in proportion to the number of VNFs to be placed in general, additionally placed VNFs should be reduced. Additionally placed VNFs are reduced by using already placed VNFs to accommodate service chain requests, and such a VNF is defined as a core VNF in CPBD. When core VNFs are used for accommodations, additionally placed VNFs need to be sequentially placed and connected from the source node to the destination node via nodes where core VNFs are located.

However, connecting from the source node to the destination node via nodes where core VNFs are placed increase hop count and consume more bandwidth resources. Moreover, when a lot of service chain requests use the same core VNF, an overhead occurs in processing. By duplicating core VNFs and placing them to multiple nodes, less overheads occur in processing, bandwidth resource consumptions are reduced, but node resource consumptions and deployment cost increase. Therefore, in order to reduce long-term deployment cost, we must decide the number of core VNFs to be placed in advance while considering the trade-off between bandwidth resource restrictions, the number of service chain requests that use the same core VNFs without delay, and the node resource restrictions. Such a problem to decide the number of core VNFs is called "core placement problem".

4.2 Placement algorithms for a core/periphery-based software system

One of the solutions to the "core placement problem" is to solve as optimization problem that minimizes the deployment cost at each time step. However, as the number of nodes

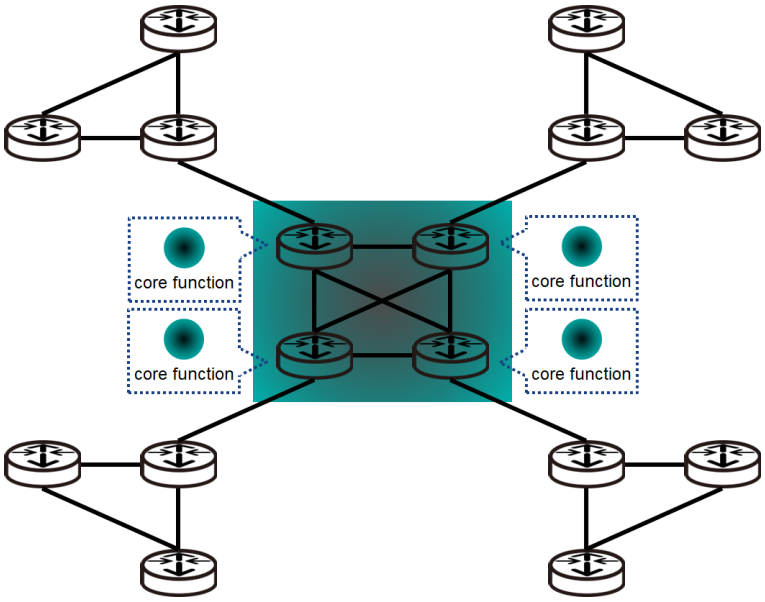

Figure 4: An example of CLCP

and VNF large, some nonlinear functions require an enormous amount of calculation time to solve as an optimization problem. Therefore, we focus on the topology of the physical network, and present two heuristics:

- CLCP (Center-Located Core/Periphery placement) policy to place core VNFs to center of physical network.
- GDCP (Geographically-Distributed Core/Periphery placement) policy to place core VNFs to be spread over the physical network.

4.2.1 Notations

The physical network is represented as $G = (V, E)$, where V is the node set and E is the link set. 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 initially allocated node and bandwidth resources, respectively. We pre-calculate the path set, P, that consists of shortest paths between each source-destination pair in $G = (V, E)$. Here, $P_{u,v} \in P$ represents shortest paths between each source node $u \in V$ and each destination node $v \in V$.

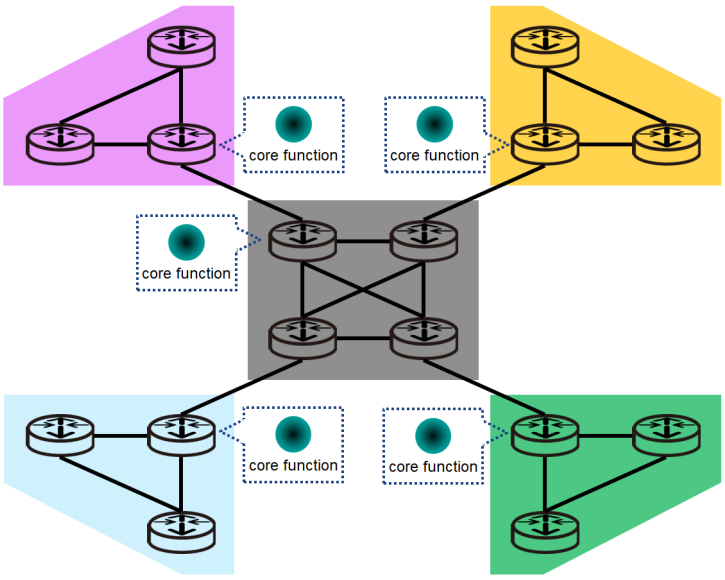

Figure 5: An example of GDCP

At each time slot t, λ type of service chain requests are generated, and required VNFs belonging to set of all VNFs, M, are placed to the physical network. When VNF $m \in M$ is placed on a node v, $\hat{c}_{m,v}$ node resources is consumed from $C_v(t)$. In this thesis, 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, when VNF $m \in M$ is placed on a node v, $\omega_{m,v}$ deployment cost is required. We also assume that $\omega_{m,v}$ is uniform for any VNF $m \in M$ and node $v \in V$, and denote it by ω. Note that the deployment cost of the entire NFV system increases in proportion to the number of VNFs placed

Each VNF $m \in M$ is classified into core VNF or periphery VNF. X is the core VNF set, and Y is the periphery VNF set. Here, $M = X \cup Y$ and $\emptyset = X \cap Y$. For each core VNF $x \in X$, w_x is the number of duplications of x.

$R = \{R_1, R_2, ..., R_T\}$ is the set of all service chain requests, and R_t is the set of service chain requests at each time slot 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 is the bandwidth resources that are consumed from $B_e(t)$ of each link $e \in E$ used for accommodating r. $U_{m,v}(t)$ is the remaining number of service chain requests that use a VNF m placed to node v without processing overhead.

We assume that $U_{m,v}(t)$ is uniform for any VNF $m \in M$ and node $v \in V$ and $U_{m,v}(0)$ represents the maximum number. If VNF $m \in M$ placed on node $v \in V$ has been already used for accommodations $U_{m,v}(0)$ times, such a VNF cannot be used for accommodating a new service chain requests. Table. 1 shows notations defined in this subsubsection.

Table 1: Notations

V	node set
E	set of links
t	time slot
T	max size of time slot
P	set of all pre-calculated shortest path
$P_{u,v}$	set of shortest paths between $u, v \in V$, and $P_{u,v} \in P$
$C_v(t)$	remaining node resources for each node $v \in V$
$B_e(t)$	remaining bandwidth resources for each link $e \in E$
M	set of all VNF
X	core VNF set
Y	periphery VNF set
w_x	the number of duplications of $x \in X$
$\hat{c}$	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 $r \in R$
$U_{m,v}$	the remaining number of service chain requests that use a VNF m placed to node v at each time t
ω	deploy cost to place a VNF to a node

Algorithm 1 The 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 been already 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**

4.2.2 CLCP: Center-Located Core/Periphery placement Policy

Nodes located at the geographic center of the physical network have more opportunity that paths go through. Placing core VNFs to such nodes may further increase opportunity that core VNFs are used for accommodating many service chain requests.

CLCP places duplication of core VNFs to nodes in descending order of the *efficiency* [16] of nodes. Efficiency is one of the metrics to measure how efficient information exchange is. A node with a high efficiency has an average short hop count from any other node, and is located at the center of the physical network.

Algorithm 1 shows the core placement algorithm of CLCP. For loop from line 2 to line 10 place core VNFs. From line 5 to line 7 get node v that has highest efficiency to place x. However, there is a node resource restriction. In placing core VNF x on node v, $\hat{c}$ resources will be consumed from the $C_v(t)$ resources remaining on node v. $\hat{c}$ should not exceed $C_v(t)$, otherwise x cannot be placed on v due to lack of node resources. If gotten v cannot satisfy node resource restriction or x has already been placed on gotten v, Algorithm 1 sets v to a node with next higher efficiency. Such processing for placing the core VNF x is repeated w_x times, which is the number of duplications of the core VNF x.

Next, we show how to place periphery VNFs. In accommodating a service chain request

with using core VNFs placed based on Algorithm 1, we place and connect periphery VNFs sequentially from the source node to the destination node via nodes where core VNFs are located. Basically, each section from the source node to the node that has a core VNF, the node with core VNF to another one, and the node with core VNF to the destination node, follow the shortest path.

Algorithm 2 obtains such a set of paths of each section, P_{avail}, that periphery VNFs are placed on. In order to avoid processing delays, this algorithm determines which nodes are gone through to use core VNFs, while considering the restriction of the number of service chain requests that use the same core VNF in line 7. If $U_{m,v}(t)$ is 0, m placed on v cannot be used to accommodate a new service chain request. Moreover, Algorithm 2 consider the bandwidth resource restriction in line 13. In using a link e for accommodating a service chain requests r, b_r bandwidth resources will be consumed from the $B_e(t)$ resources remaining on link e. b_r should not exceed $B_e(t)$, otherwise r cannot use e due to lack of bandwidth resources. If $U_{m,v}$ of all the placed core VNFs is 0 or Algorithm 2 does not obtain P_{avail} with the remaining bandwidth resources bigger than b_r, a service chain request r is rejected.

Algorithm 3 places periphery VNFs to P_{avail} obtained byAlgorithm 2. However, if the hop count between nodes in P_{avail} is too short, periphery VNFs may not be additionally placed. Shorter path go through less nodes, thus there are less remaining node resource to run additional VNFs. In particular, nodes with core VNFs have few remaining resources. To avoid such a situation that periphery VNFs cannot be placed, from line 11 to line 16 add a detour to shortest path of each section. In more detail, when m is placed to v that owns core VNFs and do not leave enough resource, p will additionally traverse a detour that go through neighbor nodes of v to reach a node remaining enough resource.

Algorithm 2 The algorithm to get path 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 any 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 line 11 to line 17

Algorithm 3 The periphery placement algorithm of CLCP

Input: $G = (V, E)$, X, P, R

for each $r \in R$ in descending order of b_r **do**
  call Algorithm 2 and obtain P_{avail}
  $p \Leftarrow$ first path of P_{avail}
  $v \Leftarrow$ source node of p
  for each $m \in \vec{n}_r$ **do**
    if $m \in X$ **then**
      $p \Leftarrow$ next path of P_{avail}
      $v \Leftarrow$ source node of p
      continue
    end if
    if v owns any core VNF **then**
      while $C_v(t) < \hat{c}$ **do**
        $v \Leftarrow$ neighbor node of v with the highest remaining node resource $C_v(t)$
      end while
      add a detour route that reaches from v to p
    end if
    while $C_v(t) < \hat{c}$ **do**
      if v is destination node of p **then**
        reject r
      end if
      $v \Leftarrow$ next node of p
    end while
    place m to v
  end for
end for

4.2.3 GDCP: Geographically-Distributed Core/Periphery placement Policy

By placing core VNFs in the center of a physical network as in CLCP, it is expected that core VNFs are used more frequently for accommodating many service chain requests. However, resources of nodes with core VNFs and neighbors of them run dry quickly because such nodes are intensively used for accommodations. Therefore, we examine another placement algorithm that spread core VNFs on the physical network.

CLCP divides the physical network into clusters to maximize the modularity [17, 18], and place duplication of core VNFs to each cluster. Modularity is one of the metrics that reflects the density of the cluster. When the modularity is maximized, there are many links in each cluster and few links between clusters.

Algorithm 4 shows the core placement algorithm of GDCP. Line 2 divide the physical network into clusters. Here, ζ is the number of clusters, and Here, the louvain algorithm [19] is used for the clustering algorithm based on the modularity, and ζ, which is the number of clusters, is determined so that it maximizes the modularity. Thus, line 3 set w_x to ζ that is the number of duplications of core VNF x. To place a duplication of core VNF x to each cluster, line 6 use Algorithm 1 that places x on the center of the cluster.

To place periphery VNFs, Algorithm 3 is used as in CLCP. Then, periphery VNFs are placed to reduce deployment cost while considering bandwidth resource restrictions, restriction of the number of service chain requests that use the same core VNF without delay, and the node resource restrictions. Note that a path may traverse multiple clusters, and there are no usage limitations that core VNFs belonging to a particular cluster must be used for accommodating service chain requests. For example, if the source node belongs to one cluster and the destination node belongs to another cluster, core VNFs placed in either of clusters are used for accommodating service chain requests.

4.3 A model for service chain requests

In the simulation to evaluate placement algorithms, service chain requests are dynamically generated. Each service chain request consists of a total of k VNFs, which is the sum of the number of core VNFs, k_c, and the number of periphery VNFs, k_p. Here, k_c and k_p are obtained by using the Eq. 5 and the Eq. 6 in the section 3.2, respectively.

Algorithm 4 The core placement algorithm of GDCP

Input: $G = (V, E)$, X

1: **if** $t = 1$ **then**
2: divide G into ζ 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 *loopcounter*-th cluster by using Algorithm 1
7: **end for**
8: **end for**
9: **end if**

When the time slot t is incremented, λ new type of service chain requests described above is generated. Both the source node and destination node of each service chain request are chosen by using uniform random. k_c core VNFs are chosen from all the $|X|$ types of core VNFs with using uniform random. Note that a service chain requests do not have duplicate VNFs. Moreover, we examine the case that the increase of t does not prevent existing service chain requests by being used.

4.4 Results

We perform simulations to evaluate CLCP policy and GDCP policy. AaP [8] is used as a placement policy for comparison. In this subsection, in order to reveal the basic characteristics of each placement policy , we first perform simulations when the resources are ∞. Next, we consider the case that the only node resources are finite and become a bottleneck for accommodating service chain requests. Finally, we show simulation results when the bandwidth resources are also finite as well as node resources.

4.4.1 Results with no resource restriction

We use 7×7 grid network as the physical network. Both initial node resource $C_v(0)$ and bandwidth resource $B_e(0)$ are ∞. The number of VNFs consisting a service chain request,

that is chain length k, is decided with using uniform random from the range of $[4, 8]$. When t is incremented, new $\lambda = 10$ service chain requests are generated. Here, we set λ to a value bigger than 1 because AaP presupposes the accommodation for multiple service chain requests at the same time. Because louvain algorithm divide 7×7 grid network into 5 clusters, we set both of CLCP and GDCP w_x that is the number of duplications of each core VNF $x \in X$ to 5. The deployment cost for each VNF, ω, is decided with using uniform random from the range of $[1, 1.2]$.

Figure 6 shows deployment cost of each placement policy, when CLCP and GDCP place $|X| = 500$ core VNFs in advance and such core VNFs are used for accommodations at the frequency of $\gamma = 0.001$. Figure 7 and Figure 8 show nodes with core VNFs of CLCP and GDCP, respectively, by coloring blue. When resources are ∞, the deployment cost of the CLCP and GDCP is the same, so both of them are shown by the CLCP/GDCP line in the figure 6. In $t \leq 8$, the deployment cost of AaP is lower, but in $9 \leq t$, deployment cost of CLCP/GDCP become lower than that of AaP. This is because CLCP and GDCP reduce additional VNFs to be placed by using already placed core VNFs to accommodate new service chain requests.

Placing more core VNFs in advance raises the performance to reduce deployment cost of CLCP and GDCP. Figure 9 shows deployment cost of each placement policy, when CLCP and GDCP place more core VNFs in advance, $|X| = 800$. Comparing the deployment costs of CLCP/GDCP at $t = 150$, the deployment costs in Figure 9 are about 12.76% less than those in Figure 6. This is because placing more core VNFs increase opportunity to use core VNFs for accommodating a service chain request, and reduce opportunity to use periphery VNFs. Moreover, increasing the parameter γ also raises the performance to reduce deployment cost of CLCP and GDCP. This is because increasing γ makes it more frequent to use core VNFs for accommodating service chain requests. Figure 10 shows deployment cost of each placement policy, when γ is 1.5 times higher than that of Figure 6. Comparing the deployment costs of CLCP/GDCP at $t = 150$, the deployment costs in Figure 10 are smaller than those in Figure 6.

Table 2 shows the average hop count of paths used by each placement policy for accommodating service chain requests. The average hop count of GLCP is smaller than that of GDCP. CLCP places core VNFs to the center of the physical network that has

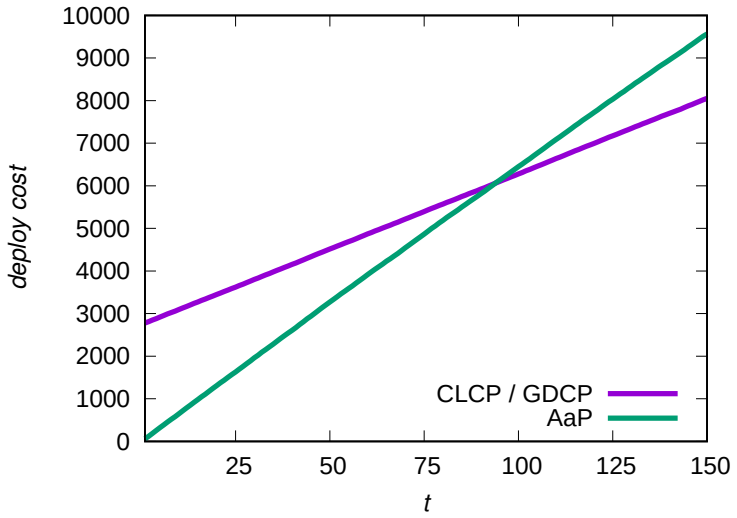

Figure 6: Deploy costs of each placement policy: $|X| = 500, \gamma = 0.001$

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

setting	placement policy	average hop count
$\|X\| = 500, \gamma = 0.001$	CLCP	6.65
	GDCP	7.15
	AaP	6.50
$\|X\| = 700, \gamma = 0.001$	CLCP	6.63
	GDCP	7.20
	AaP	6.49
$\|X\| = 500, \gamma = 0.0015$	CLCP	6.63
	GDCP	6.70
	AaP	6.49

an average short hop count from any node, CLCP tends to be able to use shorter paths through nodes with core VNFs than GDCP. Note that AaP is the placement policy with an average shortest hop count. This is because AaP uses the shortest path from the source node to the destination node, while CLCP and GDCP add detours to a path to go through a node with core VNFs.

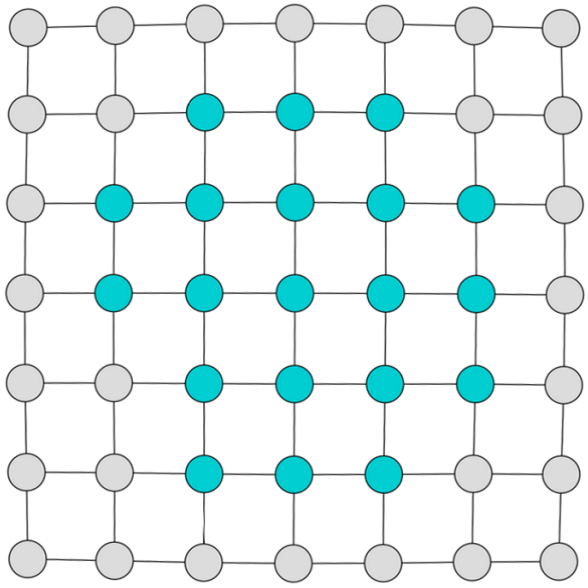

Figure 7: Location of core VNFs: CLCP, 7 × 7 grid network

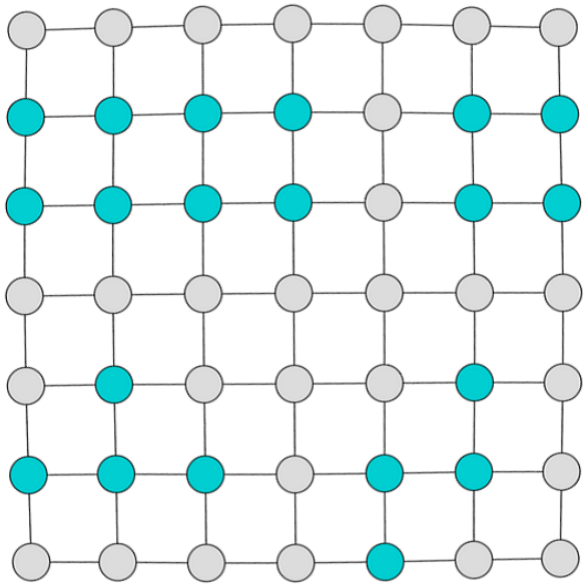

Figure 8: Location of core VNFs: GDCP, 7 × 7 grid network

4.4.2 Results with restrictions of computing and bandwidth resources

As the NFV system is operated for a long time and accommodates a lot of service chain requests, resources are gradually dried up. In such a situation, some service chain requests

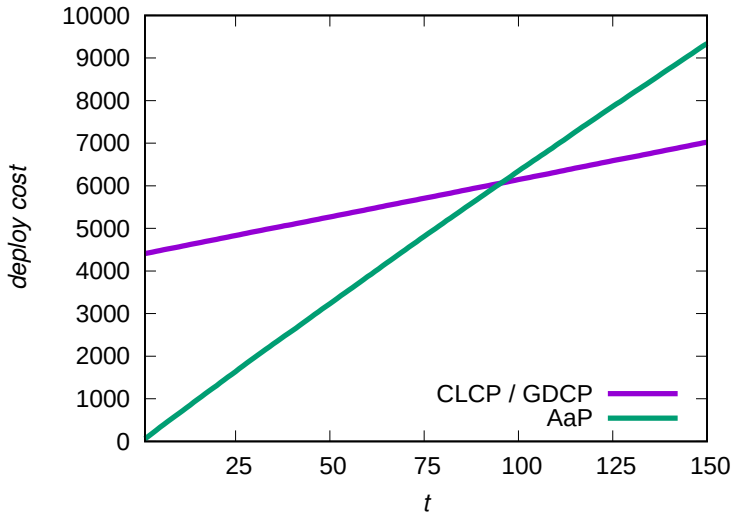

Figure 9: Deploy costs of each placement policy: $|X| = 800, \gamma = 0.001$

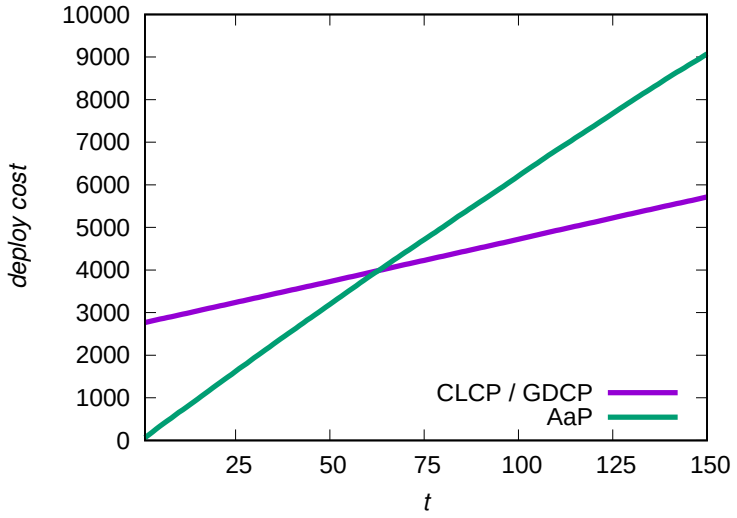

Figure 10: Deploy costs of each placement policy: $|X| = 500, \gamma = 0.0015$

cannot be accommodated due to insufficient resources. Therefore, it is difficult to evaluate the placement policies by comparing the deployment costs simply. Thus, when resources are limited, we normalize the deployment cost by the number of service chain requests accommodated. Moreover, in order to evaluate how efficiently each placement policy uses the finite resources, we show the amount of resources consumed by the placed VNFs to

accommodate service chain requests. The amount of resources consumed by placed VNFs is normalized, as well as the deployment cost, by the number of service chain requests accommodated.

First, we consider a case when only the node resources are finite and become a bottleneck to accommodate service chain requests. We set initial node resources $C_v(0)$ to 100 for any node $v \in V$ and initial bandwidth resource $B_e(0)$ to ∞ for any link $e \in E$. Node resource consumed by a placed VNF, $\hat{c}$, is decided with using uniform random from the range of $[0.4, 1]$. The upper number of service chain requests that use a VNF m placed to node v without processing overhead, $U_{m,v}(0)$, is decided with using uniform random from the range of $[4, 40]$. When t is incremented, new $\lambda = 10$ service chain requests are generated. Chain length, k, is decided with using uniform random from the range of $[4, 8]$. The deployment cost for each VNF, ω, is decided with using uniform random from the range of $[1, 1.2]$.

Figure 11 shows deployment cost of each placement policy, when CLCP and GDCP place $|X| = 500$ core VNFs to 7×7 grid network in advance and such core VNFs are used for accommodations at the frequency of $\gamma = 0.001$. Referring to that the louvain algorithm divides 7×7 grid networks into 5 clusters, we set both CLCP and GDCP w_x that is the number of duplications of each core VNF $x \in X$ to 5 as well as the case when resource is ∞. Deployment costs per service chain request accommodated by CLCP and GDCP decreases as t increases. This results indicates an increase in the performance to reduce deployment cost of CLCP and GDCP due to the repeated use of the core VNFs placed in advance for accommodating future service chain requests. When t is small and few service chain requests are accommodated, the performance to reduce deployment cost cannot be sufficiently exhibited, and core VNFs placed in advance increase the total deployment costs. When t is larger and more service chain requests are accommodated, core VNFs will be used repeatedly enough for accommodations, and the performance to reduce deployment cost is more remarkably exhibited. In Figure 11, the deployment cost of AaP is lower than that of CLCP or GDCP in about $t < 100$. For example, deployment costs per service chain requests accommodated by AaP at t = 1 is 6.82, whereas that of CLCP and GDCP is 278.69. However, in $100 \leq t$, deployment costs per service chain request accommodated by CLCP and GDCP become less than

that of AaP. Note that the deployment costs of GDCP become lower than that of CLCP. This is because they accommodate different number of service chain requests due to the node resource restriction. CLCP places core VNFs only nodes at the center of the physical network and intensively use those nodes to accommodate service chain requests, thus node resource restrictions are likely to occur. Whereas GDCP spreads core VNFs on the physical network, thus can use geographically distributed nodes and accommodate more service chain requests than CLCP. Moreover, in the long-term perspective, GDCP accommodates more service chain requests than AaP. For example, at t = 150, AaP accommodates 1193 service chain requests, however the number of service chain requests accommodated GDCP is 1346. Results similar to that of deployment cost is obtained in Figure 12 that shows the amount of node resources consumed by the placed VNFs per accommodated service chain requests. Therefore, GDCP is the best placement policy that accommodates many service chain requests with low deployment cost while using finite node resources efficiently. Note that Placing more core VNFs in advance raises the performance to reduce deployment cost of CLCP and GDCP as well as the case when resources is ∞. However, the number of core VNFs to be placed in advance is restricted by the initial node resources. Placing too many core VNFs consumes too much resources, and makes it difficult to place additional VNFs for accommodating new service chain requests. In such a case, the performance to reduce deployment cost cannot be sufficiently exhibited, and the entire deployment cost increases greatly due to placing many core VNFs in advance.

Even if the physical network topology changes, we obtain results similar to the case of grid network. When physical network is larger scale 9×9 grid network, the deployment cost and the amount of node resources consumed by the placed VNFs per accommodated service chain requests are shown in Figure 13 and Figure 14, respectively. In order to compare with model base network, we use 49-node BA topology and show results in Figure 15 and Figure 16. Figure 17 and Figure 18 show nodes with core VNFs of CLCP and GDCP of 49-node BA topology, respectively, by coloring blue. Here, in order to generate 49-node BA topology, we set the number of nodes to 49 and number of edges to attach from a new node to existing nodes to 2, so that the number of nodes and the number of edges are similar to the 7×7 grid network. Moreover, we use a 40-node ternary tree as the physical node to compare the grid topology with a very different network topology,

where 40-node ternary tree has a similar number of nodes as a grid network. In a 40-node ternary tree, each node has three or no children, and the height is 3. Results of a 40-node ternary tree are shown in Figure 19 and Figure 20. Figure 21 and Figure 22 show nodes with core VNFs of CLCP and GDCP of a 40-node ternary tree, respectively, by coloring blue. In the 9×9 grid network and 49-node BA topology, $|X| = 500$, $\gamma = 0.001$, and $w_x = 5$. In a 40-node ternary tree, $|X| = 250$, $\gamma = 0.002$, and $w_x = 7$. $|X|$ of a 40-node ternary tree is smaller than that of other network topology, because a 40-node ternary tree has less nodes and less node resources. w_x of a 40-node ternary tree is different from that of other network topology, because louvain algorithm divide a 40-node ternary tree into a different number of clusters. In any of above network topologies, the performance to reduce deployment cost and node resource consumption of CLCP and GDCP increases as t increases as well as the case when 7×7 grid network.

Next, we consider the case when the bandwidth resources are also finite as well as node resources, and set $B_e = 500$. Bandwidth resources of any link $e \in E$ consumed by each each service chain requests $r \in R$ is decided with using uniform random from the range of [1, 5]. Other settings are the same as in the case when only the node resources are finite.Other settings are the same as in the case when only the node resources are finite. Figure 23 shows deployment cost of each placement policy, when the physical network is 7×7 grid network, $|X| = 500$, $\gamma = 0.001$, and $w_x = 5$. Comparing the deployment costs at $t = 150$ between Figure 11 and Figure 23, that in Figure 23 is larger for both CLCP and GDCP. This result indicate that the performance to reduce deployment cost was weakened, because adding the bandwidth resource restriction decrease the number of service chain requests accommodated by CLCP and GDCP. However, at $t = 150$ in Figure 23, deployment cost per service chain requests by GDCP is still the lowest. That of CLCP is by about 12.99% larger than Figure 11, but that of GDCP is by only about 4.26% larger. In CLCP, nodes at the geographic center of the physical network are intensively used, thus the links that reaches those nodes are also intensively used. As a result, bandwidth resource restrictions are likely to occur as well as node resource restrictions, and both of restrictions greatly reduce the number of service chain requests accommodated by CLCP. Whereas GDCP uses geographically distributed nodes, and less bandwidth resource restrictions occur than CLCP. Similar results is obtained in Figure 24 that shows the amount of node

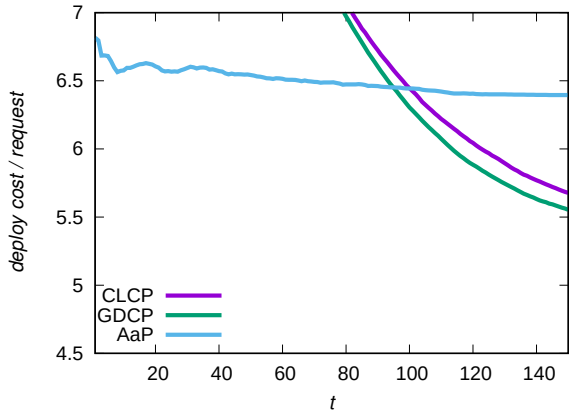

Figure 11: Deployment cost: 7×7 grid network, $C_v = 100, B_e J = \infty$, $|X| = 500, \gamma = 0.001, w_x = 5$

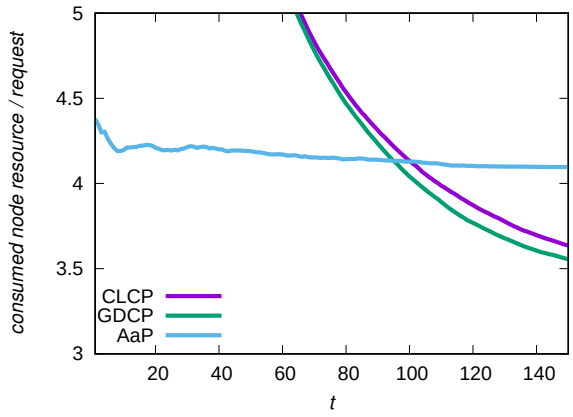

Figure 12: The amount of node resources consumed by the placed VNFs: 7×7 grid network, $C_v = 100, B_e = \infty$, $|X| = 500, \gamma = 0.001, w_x = 5$

resources consumed by the placed VNFs per accommodated service chain requests. Note that AaP is superior to CLCP and GDCP in terms of bandwidth resource consumption, for the same reason that AaP has an average shortest hop count in section 4.4.1.

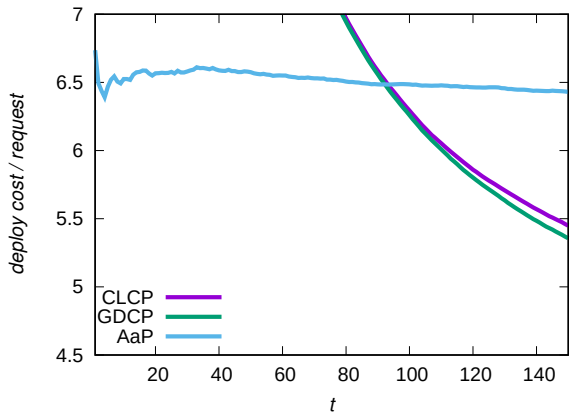

Figure 13: Deployment cost: 9×9 grid network, $C_v = 100, B_e = \infty$, $|X| = 500, \gamma = 0.001, w_x = 5$

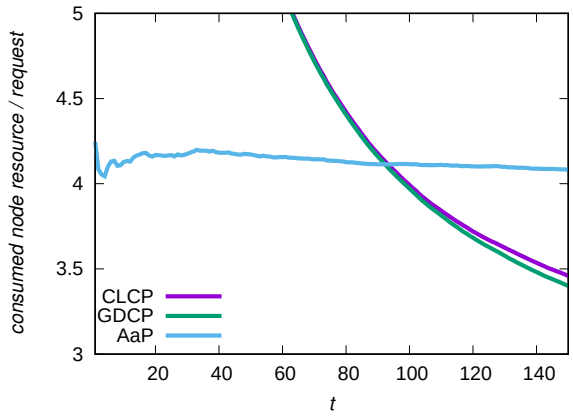

Figure 14: The amount of node resources consumed by the placed VNFs: 9×9 grid network, $C_v = 100, B_e = \infty$, $|X| = 500, \gamma = 0.001, w_x = 5$

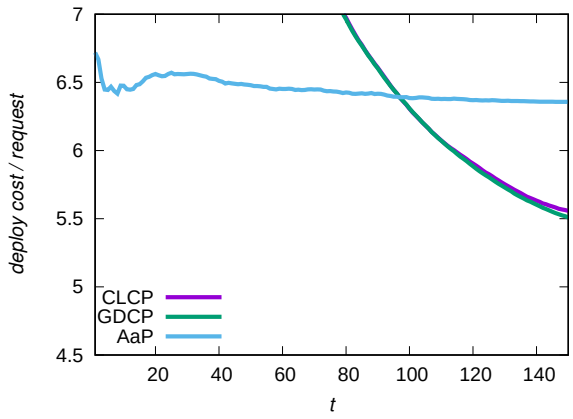

Figure 15: Deployment cost: 49-node BA topology, $C_v = 100, B_e = \infty$, $|X| = 500, \gamma = 0.001, w_x = 5$

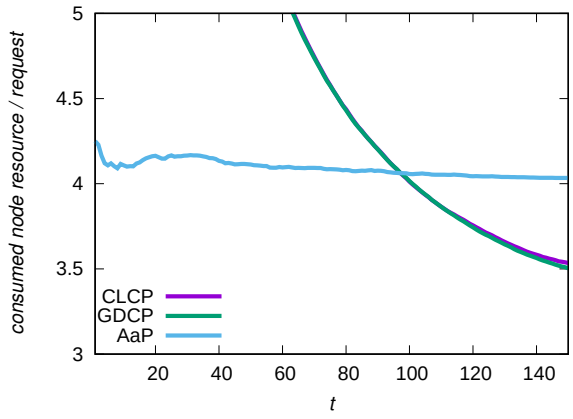

Figure 16: The amount of node resources consumed by the placed VNFs: 49-node BA topology, $C_v = 100, B_e = \infty$, $|X| = 500, \gamma = 0.001, w_x = 5$

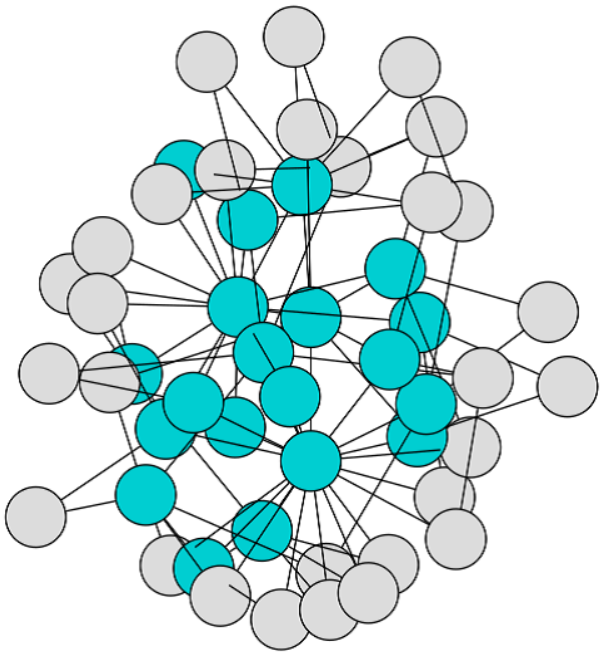

Figure 17: Location of core VNFs: CLCP, 49-node BA topology

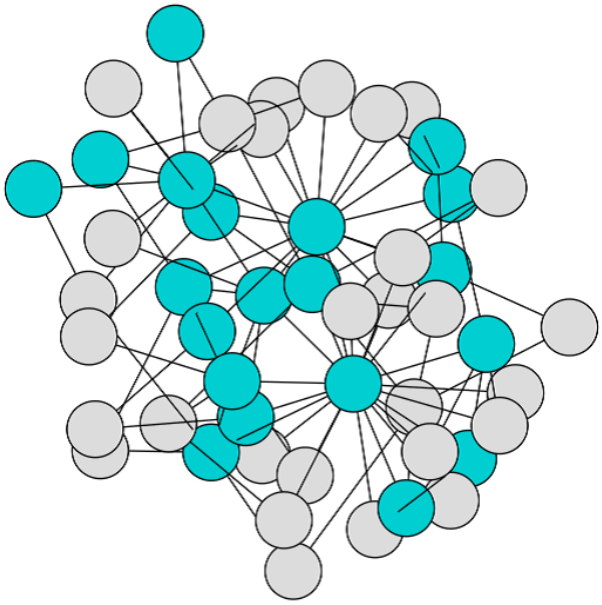

Figure 18: Location of core VNFs: GDCP, 49-node BA topology

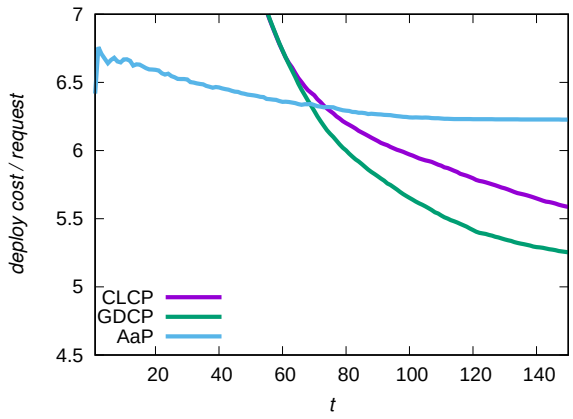

Figure 19: Deployment cost: 40-node ternary tree, $C_v = 100, B_e = \infty$, $|X| = 500, \gamma = 0.001, w_x = 5$

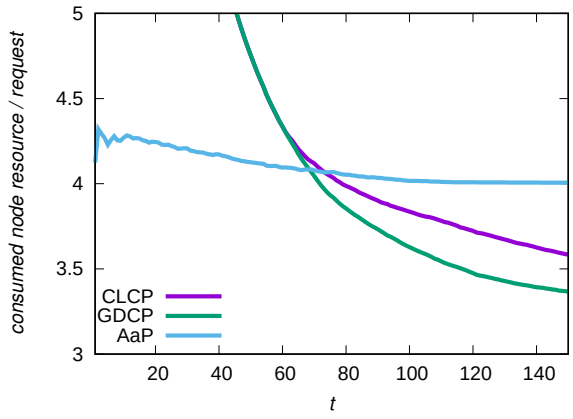

Figure 20: The amount of node resources consumed by the placed VNFs: 40-node ternary tree, $C_v = 100, B_e = \infty$, $|X| = 500, \gamma = 0.001, w_x = 5$

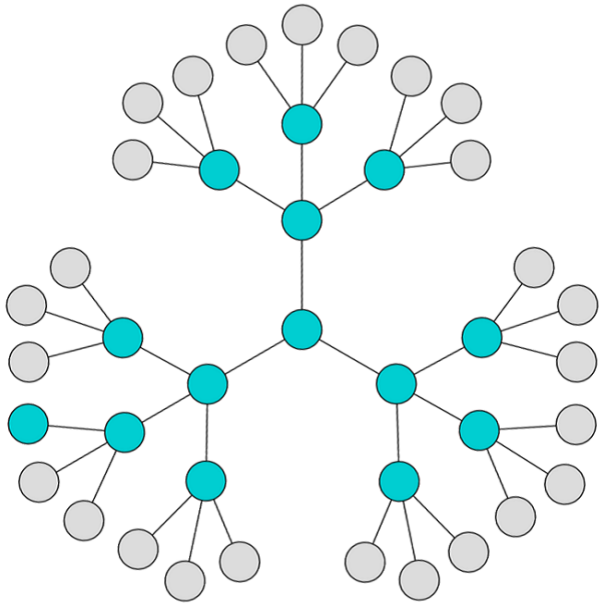

Figure 21: Location of core VNFs: CLCP, 40-node ternary tree

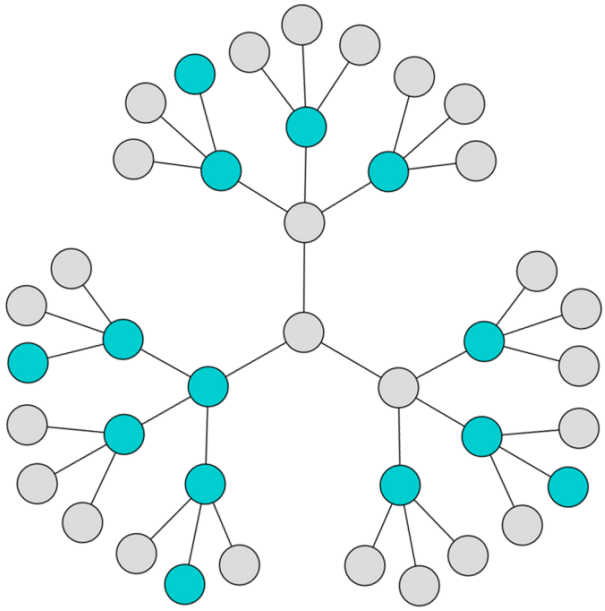

Figure 22: Location of core VNFs: GDCP, 40-node ternary tree

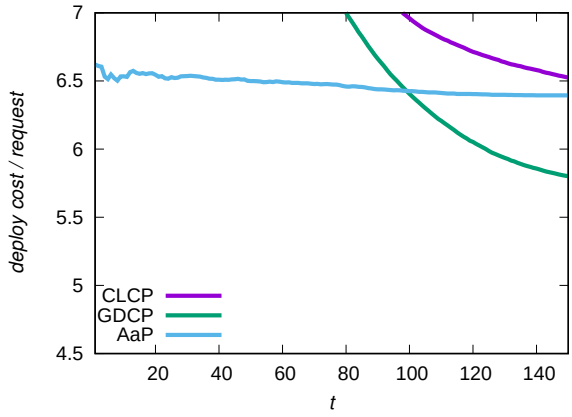

Figure 23: Deployment cost of 7×7 grid network with restriction of link bandwidth

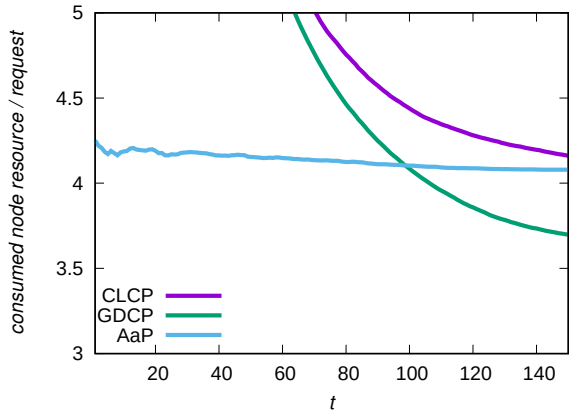

Figure 24: The amount of node resources consumed by the placed VNFs: 7×7 grid network, $C_v = 100, B_e = 500, |X| = 500, \gamma = 0.001, w_x = 5$

5 Conclusion

In this thesis, we investigate software design and placement method of VNFs to reduce long-term development cost and deployment cost against the change of service requests. We introduce CPBD that is the software design of the NFV system based on a core/periphery structure. CPBD repeatedly uses core VNFs for accommodating future service requests, and develops periphery VNFs specifically for each service request. Our evaluation results indicate that CPBD accommodates service chain requests with lower development cost than that of software design without core VNFs. Then, we investigate the placement problem of VNFs which are designed by CPBD. Considering the nature of core/periphery structure, we introduce CLCP policy and GDCP policy. CLCP places core VNFs to the center of the physical network that has an average short hop count from any node in order to increase the utility of core VNFs. Unlike CLCP, GDCP spreads core VNFs on the physical network to avoid intensively using specific nodes and links to accommodate service chain requests. In both cases, with or without resource restrictions, GDCP is the best placement policy that accommodates many service chain requests with low deployment cost while using finite node resources efficiently.

In this thesis, we assume that the usage frequency among core VNFs are identical for simplicity. One of our future works is to consider the difference of usage frequency among core VNFs. For example, GDCP should treat the less-used core VNFs as periphery VNFs. Another of our future works is to evaluate the validity of parameter settings of CPBD/GDCP by using actual application services.

Acknowledgments

This thesis would not accomplish without a lot of great support from many people. First, I would like to express my deepest gratitude to Professor Masayuki Murata of Osaka University, for providing me with the opportunity to research with a talented team of researchers. He sometimes made me aware of my own mistakes, and at other times encouraged my research activities.

Furthermore, my heartfelt appreciation goes to Associate Professor Shin'ichi Arakawa of Osaka University. He took a lot of time for me, discussed with me again and again. I received a lot of support from him. Not to mention this thesis, my research activities for more than two years would not be the same without him.

Moreover, I offer my special thanks to Associate Professor Yuichi Ohsita and Assistant Professor Daichi Kominami of Osaka University. Their continued support and kindness have helped me to improve my research.

I would like to thank all the members of Advanced Network Architecture Research Laboratory.

Last, but not least, I thank my parents and my sister for their invaluable support and constant encouragement during my master studies.

References

[1] "Network Functions Virtualisation Introductory White Paper," ETSI, Oct. 2012.

[2] "Network Functions Virtualisation - White Paper #3," ETSI, Oct. 2014.

[3] K. Sanghyeok, P. Sungyoung, K. Youngjae, K. Siri, and L. Kwonyong, "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.

[4] N. Yeonghun, S. Sooeun, and C. Jong-Moon, "Clustered NFV service chaining optimization in mobile edge clouds," *IEEE Communications Letters*, vol. 21, no. 2, pp. 350–353, Oct. 2017.

[5] 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.

[6] C. Peter, L. András, W. Ling-Yun, and U. Brian, "Structure and dynamics of core/periphery networks," *Journal of Complex Networks*, vol. 1, no. 2, pp. 93–123, Oct. 2013.

[7] Miele V. and Ramos-Jiliberto R. and Vázquez D. P., "Core-periphery dynamics in a plant-pollinator network," *bioRxiv, 543637*, Jul. 2019.

[8] S. Quanying, L. Ping, L. Wei, and Z. Zuqing, "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.

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

[10] M. Alan, R. John, and B. C. Y, "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.

[11] 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.

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

[13] M. Alan and S. D. J, "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.

[14] H. Wilhelm, "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.

[15] A. Reznik, R. Arora, M. Cannon, L. Cominardi, W. Featherstone, R. Frazao, F. Giust, S. Kekki, A. Li, D. Sabella *et al.*, "Developing software for multi-access edge computing," *ETSI, White Paper*, vol. 20, no. 1, Sep. 2017.

[16] L. Vito and M. Massimo, "Efficient behavior of small-world networks," *Phys. Rev. E 87, 19801*, no. 19, Oct. 2001.

[17] 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.

[18] ——, "Fast algorithm for detecting community structure in networks," *Phys. Rev. E 69, 066133*, no. 6, Jun. 2004.

[19] Vincent 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.