Doctoral Thesis

Fault-Resilient Resource Allocation in Network Function Virtualization

Rui KANG

Graduate School of Informatics, Kyoto University

September 2023

Preface

Traditional network functions (NFs) are coupled with dedicated hardware, which is inflexible for scaling on demand and costly for updating. The idle resources on each dedicated hardware in a traditional network cannot be shared and increase the cost of deployment and maintenance, which is inconsistent with achieving the sustainable development goals (SDGs). Network function virtualization (NFV) introduced by the European telecommunications standards institute (ETSI) decouples the functions from specialized hardware. The functions are virtualized to virtual network functions (VNFs) and can form a service function chain (SFC) in a specific order to provide customized service. End users avoid the expense of replacing a multitude of dedicated machines in each room to update their systems. Computational resources provided by universal hardware can be shared among different NFs flexibly.

With the increase in the amount and complexity of NFs, manual deployment and management is becoming overwhelming. Automated deployment and management are imperative and can respond to user requirements in a timely manner. A well-designed resource allocation model cooperated with the NFV orchestration (NFVO) engine (e.g. Kubernetes) for VNFs can improve the network performance in terms of different objectives. Among them, ensuring fault-tolerance of the NFs and mitigating the impact of hardware and software failures and unavailabilities on the quality of service (QoS) are the issues to be explored in this thesis. This thesis studies five different allocation problems for different levels of known information about the failures with different practices of their implementations on an NFVO platform.

Firstly, this thesis proposes an optimization model to derive the VNF allocation of time slots in sequence aiming to maximize the continuous available time of SFCs in a network, which suppresses service interruptions created by

the node unavailability and the reallocation of VNFs. This work computes VNF allocation in a series of time slots based on a node availability schedule, which provides information on the availability of each node in each time slot. This work formulates the proposed model as an integer linear programming (ILP) problem with the goal of maximizing the minimum number of longest continuous available time slots in each SFC. This work proves that the decision version of the VNF allocation problem (VNFA) is NP-complete. As the size of ILP problem increases, the problem is difficult to solve in a practical time. This work develops a heuristic algorithm to solve the VNFA problem. Numerical results show that the proposed model improves the continuous available time of SFCs compared with existing models, which partially consider node unavailability or VNF reallocation. The proposed model together with a consideration of routing reduces the path length of requests. The introduced heuristic algorithm is faster than the ILP approach with a limited performance penalty.

Secondly, this thesis proposes a primary and backup VNF placement model to avoid service interruptions caused by node unavailability by using backup functions. The considered backup functions have a period of startup time for preparation before they can be used and the number of them is limited. The proposed model is formulated as an ILP problem to place the primary and backup VNFs based on the availability schedule at continuous time slots. This work aims to maximize the minimum number of continuously available time slots in all SFCs over the deterministic availability schedule. The proposed model considering the limited number of backup functions outperforms baseline models in terms of the minimum number of longest continuous available time slots in all SFCs. This work introduces an algorithm to estimate the number of key unavailabilities at each time slot, which can find the unavailable nodes which are the bottlenecks to increase the service continuous available time at each time slot.

Thirdly, this thesis proposes a robust optimization model to allocate VNFs in SFCs for time slots in sequence aiming to maximize the continuous available time of SFCs in a network with uncertain availability schedules by suppressing the interruptions caused by node unavailability marked in availability schedule and function reallocation. This work formulates the model as an ILP problem

over the given uncertainty set of the start time slot and period of unavailability on each node in the availability schedule. For solving the model in a practical time in a relative large size of network, this work develops a heuristic algorithm. The numerical results show that the proposed model outperforms the baseline models under different levels of robustness in terms of the worst-case minimum number of the longest continuous available time slot in each SFC. The heuristic algorithm reduces the computation time with limited performance loss compared with the ILP approach. In the discussion, this work introduces a constraint condition for the maintenance ability, which reduces the size of uncertainty set, and an extension for supporting more than one unavailability periods in the availability schedule on each node.

Fourthly, this thesis proposes an optimization model to derive a resilient virtual network function allocation in service function chains aiming to reduce the end-to-end (E2E) latency during the migrations from the primary functions to backup functions. The model considers k-fault tolerance assurance and the satisfactions of service requirements under different error patterns in this model. The allocation provided by the proposed model ensures that the processing ability satisfies the requirements even though there are k failed nodes in the network. Diversity splits a single VNF into a pool of replicas with different specifications. The diversity of both primary and backup functions is considered. Redundancy is used for recovering the failed functions. This work formulates the proposed model as a mixed integer linear programming problem to select suitable replicas from the pools of replicas and decide the allocations of these replicas for both primary and backup functions. The objective of the proposed model is to minimize the sum of the maximum E2E latencies among functions under all possible failure patterns which have k node failures. The numerical results show that the proposed model reduces the E2E latency between the pair of primary and backup VNFs while ensuring the resiliency of the functions compared with baseline models in the examined cases. Two approximate approaches are developed to reduce the computation time of solving the proposed model with a limited performance penalty. This work derives theorems to give the bounds of maximum resiliency in the proposed model.

Fifthly, this thesis proposes VNF allocation aiming to maximize the num-

ber of accepted requests with considering VNF diversity and ensuring the requirements of node fault tolerances in a dynamic scenario, where the requests have random requirements, arriving and releasing time. The model considers fault tolerance assurance and satisfies the service requirements under different error patterns. The allocation provided by the proposed model ensures the required amount of processing ability in the situation that there are several failed nodes. The node fault tolerance can be set variously for different requirements of services. The proposed model selects and instantiates suitable replicas from the pools of replicas, and then determines the locations of these replicas instances. This work develops a reinforcement learning approach for solving the proposed model including the design of the learning environment and the reward shaping. The numerical results show that the proposed model increases the number of accepted requests with ensuring the resiliency of the functions compared with baseline models in the examined cases, where the allocation of a request can be determined in tens of milliseconds.

Sixthly, this thesis demonstrates the implementations of resource allocation models in Kubernetes in two ways. Kubernetes is an open-source system for automating deployment, scaling, and management of containerized applications and used in various companies. Firstly, this work implements function scheduler plugins cooperating with multiple reliable function allocation models in Kubernetes. Demonstration validates that the plugins allocate functions by using the allocation results obtained by the model automatically and run service functions correctly. Secondly, this work designs and implements a custom resource and the corresponding controller in Kubernetes to manage the resource allocation model which consider the VNF diversity and redundancy jointly. The controller selects suitable replicas from a pool of replica templates to satisfy the required processing ability with the minimum required number of replicas and converts the backup functions to the primary functions when the primary functions cannot provide the required ability. Demonstration validates that the controller automatically manages the resources correctly, improves the resource utilization, and increases the number of acceptable requests.

Seventhly, this thesis demonstrates the implementations of SFC cooperated with SFC allocation models in two ways. Firstly, this work implements a network service header based service function chain application which can

be cooperated with the model. Demonstration validates that the application allocates functions by using the allocation from the model automatically and runs service function chains correctly. Secondly, this work implements an open virtual network based SFC-compatible network plugin for Kubernetes. This work implements two controllers for creating SFCs among existing functions and SFC deployments without existing functions which can be cooperated with allocation models. The plugin allocates the functions in chains according to the given models and connects each function in chains by setting suitable flow entries in Kubernetes.

This thesis is organized as follows. Chapter 1 introduces the background of fault-resilient resource allocation in NFV. Chapter 2 investigates the related works in literature. Chapter 3 presents the resource allocation model considering the deterministic available schedule. Chapter 4 presents the resource allocation model for primary and backup VNFs under the deterministic available schedule. Chapter 5 presents the robust resource allocation model considering the uncertain available schedule. Chapter 6 presents the fault-tolerant resource allocation model for static service requirements considering VNF diversity and redundancy jointly. Chapter 7 presents the fault-tolerant resource allocation model for dynamic service requirements by using the reinforcement learning approach. Chapter 8 presents the implementations of resource allocation models and SFCs in Kuberenetes. Finally, Chapter 9 concludes this thesis.

Acknowledgements

This thesis is the summary of my study in master's and doctoral courses at the Graduate School of Informatics, Kyoto University, Japan. I would like to extend my gratitude to a number of institutions and organizations and people for their help on this thesis and my future career.

First of all, I would like to thank the Japan Society for the Promotion of Science for providing financial support for my living and research. I would also like to thank the Japanese government and Kyoto University for their scholarship and tuition exemption, respectively. Thanks to the financial assistance provided by the above institutions and organizations, which prevents me from struggling in livings and makes it possible for me to not be limited to people for financial reasons. The financial freedom allows me to sink my knowledge so that I can make truly meaningful research. I can say that without the financial support they gave, I would not have been able to finish my thesis and continue my research career. I would like to give special thanks to our secretary Ms. Mariko Tatsumi and the staff of the Graduate School of Informatics, who gave me helps with procedures and listened to me when I was confused.

I would like to thank my supervisor Professor Eiji Oki. He led me into a new world of research and used his experience to help me. Without his help, there would be no basis for this thesis. I would like to thank my deputy supervisor Professor Takashi Sato. He gave me a lot of advice and helps on academic career and campus life. I would like to thank Professor Hiroshi Harada and Professor Takashi Sato for being part of my judging committee and providing their precious comments to improve my thesis. I would like to thank my co-authors, especially Ms. Mengfei Zhu and Dr. Fujun He. With everyone's help, a complete study can be achieved. Thanks for sharing the stress and pain with me. The honor also belongs to you. I would also like to thank Associate

Professor Takehiro Sato for his involvement at the beginning of my research experience. His careful comments are still fresh in my mind, but he also taught me an important lesson as I grew up and showed me how a person can grow and change. I would like to give special thanks to Professor Klaus-Tycho Foerster from TU Dortmund for not only his outstanding academic achievements but also his kindness. He not only opened new doors in my academic career. He also let me that there are still many good people in the world. In addition, I would like to thank the people I met while attending the conference for their discussions, encouragement, and understanding.

Finally, I must express my very profound gratitude to my parents and to my partners for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them. Thank you.

Contents

List of Figures

List of Tables

Notations

Notation	Description
$G(N, L)$	Virtual network.
N	Set of virtual nodes.
T	Set of time slots.
T_i	Set of time slots from 1 to $\|T\| - i + 1$.
R	Set of requests.
L	Set of virtual links.
S	Set of types of resources.
K_r	Length of request $r \in R$.
c_{nt}^s	Capacity of node $n \in N$ at time slot $t \in T$ for resource $s \in S$.
q_s^{rk}	Requirement of resource $s \in S$ for the $k \in K_r$th function of request $r \in R$.
e_t^n	An availability schedule; if node $n \in N$ is unavailable at time slot $t \in T$, $e_t^n = 1$, and otherwise 0.
d_r	Transmission resources demanded by request $r \in R$.
b_{ij}	Transmission resource between nodes $i \in N$ and $j \in N$.
l_{ij}	Length between nodes $i \in N$ and $j \in N$.
x_{tn}^{rk}	If the kth function of request r is assigned to node n at time slot t, $x_{tn}^{rk} = 1$, and otherwise 0.
α_{tn}^{rk}	Chapter 4: If the kth function of request r is prevented from the interruption caused by unavailability $e_t^{n'}$, $\alpha_{tn}^{rk} = 1$; otherwise, 0. Chapter 4: auxiliary variable for linearization.
β_r	SCAT of request $r \in R$.
λ	SSCAT.

Notation	Description
o_t^r	Chapters 3 and 5: If the allocations of at least one function in request r is changed between time slot $t-1$ and time slot t or any VNF of request r at time slot t or $t-1$ is allocated to an unavailable node, is set to 0, and otherwise 1. Chapter 4: If the kth function of request r is prevented from the interruption caused by unavailability $e_t^{n'}$, $\alpha_{tn}^{rk} = 1$; otherwise, 0.
z_i^{jr}	If allocations of request r from time slot i to time slot $i+j-1$ are consecutively unchanged, or j consecutive o_t^r are all 1 from $t=i$ to $t=i+j-1$, z_i^{jr} is set to 1, and otherwise 0.
y_j^r	If the allocations are consecutively unchanged during j time slots existing in T, y_j^r is set to 1, and otherwise 0.
g_r^k	Preparation time of the $k \in K_r$th function of request $r \in R$.
u_{tn}^{rk}	If the kth function of request $r$is backed up on node n at time slot t, $u_{tn}^{rk} = 1$; otherwise, 0.
s_n	Starting time slot of unavailability period.
f_n	Duration of unavailability period.
$\bar{s_n}, \hat{s_n}$	Definition of uncertainty set s_n
$\bar{f_n}, \hat{f_n}$	Definition of uncertainty set f_n
$\mathcal{P}_n$	Uncertainty set of beginning time slots on node n.
p_n	Selected set from $\mathcal{P}_n$
p_n^q	qth element in p_n
Γ_n^{S}	Degree of robustness of s_n. Size of $\mathcal{P}_n$.
Γ_n^{F}	Degree of robustness of f_n. Length of the duration of unavailability period on node n
F^c	Ordered set of VNFs belonging to SFC $c \in C$.
$A_f^{[\bullet]}$	Set of primary ($\bullet$ is $\mathcal{P}$) or backup ($\bullet$ is $\mathcal{B}$) replica VNFs of VNF $f \in F$.
U	Set of error patterns with k failures among nodes.
m_n	Capacity of node $n \in N$.
d_{ij}	Estimated latency of link $(i,j) \in L$.

Notation	Description
$P_{fa}^{[\bullet]}$	Processing ability of the ath primary ($\bullet$ is $\mathcal{P}$) or backup ($\bullet$ is $\mathcal{B}$) replica of VNF $f \in F$.
$S_{fa}^{[\bullet]}$	Resource requirement of the ath primary ($\bullet$ is $\mathcal{P}$) or backup ($\bullet$ is $\mathcal{B}$) replica of VNF $f \in F$.
γ_r^c	Set to 1 if request $r \in R$ requests SFC $c \in C$; 0 otherwise.
g_r^c	Required processing ability for SFC $c \in C$ of request $r \in R$.
ψ_{fc}	Set to 1 if function $f \in F$ belongs to SFC $c \in C$; 0 otherwise.
α_{ij}^w	Set to 1 if node w is the head of (i, j), i.e., $w = j$; set to -1 if node w is the tail of (i, j), i.e., $w = i$; 0 otherwise.
$x_{nfa}^{[\bullet]}$	Binary. Set to 1 if the $a \in [1, \lvert A_f^{[\bullet]} \rvert]$th primary ($\bullet$ is $\mathcal{P}$) or backup ($\bullet$ is $\mathcal{B}$) replica of VNF $f \in F$ is assigned to node $n \in N$, $x_{nfa}^{[\bullet]} = 1$; 0 otherwise.
$p_{ij}^{faa'}$	Binary. Set to 1 if the migration route from the $a \in [1, \lvert A_f^{\mathcal{P}} \rvert]$th primary replica of VNF f to the $a' \in [1, \lvert A_f^{\mathcal{B}} \rvert]$th backup replica of $f \in F$ includes link $(i, j) \in L$; 0 otherwise.
μ_u	Sum of the maximum E2E latencies from the primary replicas to backup replicas among all VNFs under error pattern $u \in U$.
R_t	Set of active requests at time slot $t \in T$.
$R_t^{\mathbb{N}}$	Set of newly arrival requests at time slot $t \in T$.
$R_t^{\mathbb{A}}$	Set of accepted active requests at time slot $t \in T$.
$R_t^{\mathbb{AN}}$	Set of accepted requests that arrive at time slot $t \in T$.
v_r	Request $r \in R$ arrives at time slot v_r.
δ_r	Time to live of request $r \in R$.
F_r	Set of VNFs requested by request $r \in R$.
P_{fa}	Processing ability of replica a in pool A_f.
S_{fa}	Resource requirement of replica a in pool A_f.
g_{rf}	Required processing ability of VNF $f \in F$ by request $r \in R$; 0, if request r does not contain VNF f.
k_r	Required resiliency level of request $r \in R$.
Δ_{rfa}	Maximum number of the instances of replica $a \in A_f$ of function F_r for request $r \in R$. Infinite if not appointed.

Notation	Description
x_{tn}^{rfa}	Integer. The number of replica instances instantiated from the replica $a \in A_f$ of VNF $f \in F_r$ for request $r \in R_t^{\mathbb{A}}$ and assigned to node $n \in N$ at time slot $t \in T$.
κ_{rft}^{k}	Binary. Set to 1 if VNF $f \in F$ ensures the required processing ability under any k fault nodes at time slot $t \in T$; 0 otherwise.
ι_{rft}	Integer. Resiliency level of VNF $f \in F$ for request $r \in R_t$ at time slot t.
ω_r	Integer. Resiliency level of request $r \in R$ during the active time slots.

Abbreviations

Abbreviation	Description
API	application programming interfaces
BP	probability of applying the backup strategy
CDR	custom defined resource
CF	cooling factor, between 0 and 1
CNI	container network interface
CPU	central processing unit
CR	complete randomization
DNN	deep neural network
DRL	deep reinforcement learning
DRPS	diversity and redundancy Pod set
E2E	end-to-end
ECP	external crossover probability
ETSI	European telecommunications standards institute
GA	genetic algorithm
ICMP	Internet control message protocol
ICP	internal crossover probability
ID	identifier
ILP	integer linear programming
INLP	integer nonlinear program
IP	Internet protocol
IPN	initial population number
IT	initial temperature
LP	linear programming
MAC	medium access control

Abbreviation	Description
MANO	management and orchestration
MC	Markov chain length
MDP	Markov decision process
MG	maximum generation
MILP	mixed ILP
MP	mutation probability
NF	network function
NFV	network function virtualization
NFVI	NFV infrastructure
NSH	network service header
OVN	open virtual network
PG	policy gradient
PH	probabilistic heuristic
PR	partial randomization
QoS	quality of services
RDM	random decision model
ReLU	rectified linear unit
RL	reinforcement learning
RT	minimum temperature limit
SA	simulated annealing
SAM	single-slot allocation model
SB	serialization-and-backtracking
SCAT	service continuous available time
SDG	sustainable development goals
SDN	software-defined network
SF	service function
SFC	service function chain
SP	service provider
SSCAT	shortest SCAT
TCP	transmission control protocol
TTL	time to live
UDP	user datagram protocol

Abbreviation	Description
ULPN	Upper limitation of population number
VLC	visible light communication
VM	virtual machine
VNF	virtual network function
VNFA	VNF allocation problem
VNFI	VNF instance

Chapter 1

Introduction

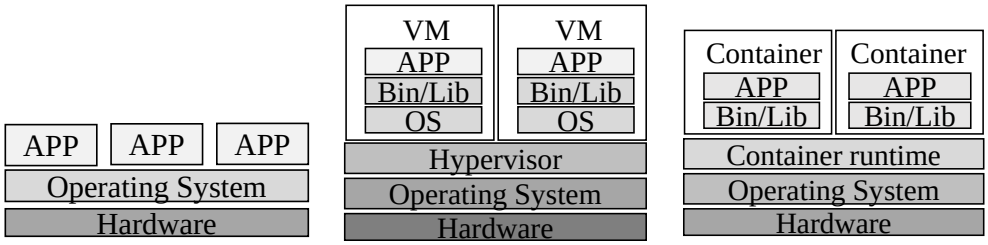
1.1 Network function virtualization

1.1.1 Definition

Network functions are provided by specialized hardware devices in traditional networks, which tend to make deploying new functions, services, and managing policies difficult and costly [1, 2]. In order to solve this problem, network function virtualization (NFV) has been introduced [3], which decouples the hardware from the functions. NFV enables virtual network functions (VNFs) to run on virtual machines (VMs) or containers, which provides more flexible services to users [4] and lowers the costs for Internet service providers. When a user requests a service, a service function chain (SFC) is generated to connect the user with the service provider (SP) servers that realize the desired functions. An SFC consists of several VNFs in a specific order that constitutes a service [5]. The flexible placement of VNFs is one of the advantages of NFV. In addition to the ability to rapidly add new network functions, this flexibility can improve network performance.

1.1.2 Function deployment

Modern Web services are implemented by the applications deployed in the servers. There are mainly three deployment types as shown in Figure 1.1. Traditionally, the function are directly deployed on physical servers, which en-



(a) Traditional deployment. (b) Virtualized deployment. (c) Container deployment.

Figure 1.1: Three function deployment types.

ables complete resource isolation at the hardware level and owns proprietary resources to ensure quality of service (QoS). At the same time, the traditional deployment method also brings some disadvantages. Network functions are deeply bound to the equipment vendors, which bring high costs. The traditional network functions are executed in private operating systems, whose configurations are not uniform and difficult to maintain. The network function processing ability cannot be increased flexibly with the increase of their workloads. It is difficult to scale their capacities. The update cycles for existing functions and development period for new functions takes long. To overcome the above disadvantages of the traditional deployment, virtualized deployment is introduced. Multiple virtual machines (VMs) are run on a single physical server. Each VM is a full machine running all the components. Applications run in VMs. Virtualized deployment separates network functions from dedicated hardware, saving potential operation expenses. Virtualized deployment allows network operators to build and deploy customized SFCs from sets of interconnected VNFs and supports automating the dynamic scaling of network functions. Along with the above advantages, virtualized deployment brings some challenges. The boot time of VM is long. There are additional overheads caused by Hypervisor and Guest OS. The VNF is implemented as a code monolith, whose development, testing, maintenance, deployment and troubleshooting must treat the VNF as a single element. The state of the art deployment evolves VMs to containers. Containers are similar to VMs, but lightweight. Functions are partitioned into smaller units which are called microservices. A group of smaller, interconnected microservices replaces a single function. Containers house microservices and managed by a platform unitedly.

1.2 Network function management and orchestration

1.2.1 Definition

NFV brings a series of new concepts including VNF, virtual links, physical network functions, NFV infrastructure (NFVI). For dealing with the relationships among them, a new management and orchestration framework is required, which is called network functions virtualization management and orchestration (NFV-MANO) architectural framework [6].

In terms of the MANO of VNFs, not only the traditional functionalities, e.g., fault management, configuration management, accounting management, performance management, and security management, but also some new functionalities for lifecycle management are required. The MANO of VNFs need to take responsibility for VNF initiation, scaling, updating, and terminating. The MANO monitors and allocates the virtulized resources for VNFs.

1.2.2 Platforms

There are two common function orchestration platforms for managing network functions and services.

OpenStack [7]:

OpenStack is an open source cloud computing management platform project, which provides scalable and elastic cloud computing services for private and public clouds. OpenStack covers all aspects of networking, virtualization, operating systems, servers, etc.. The orchestration is provided by Heat [8], which provides a collaborative deployment approach defined by templates to automate the deployment of cloud infrastructure software operating environments including computing, storage, and network resources.

Kubernetes [9]:

Kubernetes is an open-source system for automating deployment, scaling, and management of containerized applications and used in various companies. A

Pod is the smallest deployable unit of computing that we can create and manage in Kubernetes. The allocations of Pods are decided by the schedulers in Kubernetes, which are implemented by a set of plugin application programming interfaces (APIs) belonging to a pluggable architecture called scheduling framework.

1.3 Reliability issues caused by failures in NFV

Reliability is an aspect to evaluate the quality of service (QoS) in the network. VNF is expected to fail more often compared to the traditional hardware network middlebox which passes the sticky testing and validation process, while the VNF is a softwarized function that is likely buggy [10]. The services in the NFV environment are prone to the failures of not only the hardware of servers but also the software of VNFs. The failures will interrupt the services and influence reliability. It is necessary to enhance network reliability and improve the recovery ability of services. In traditional networks, it is common to add a group redundancy middleboxes for ensuring the recover ability of hardware middleboxes. The redundancy hardware keeps standby mode while there is no failure, which results in a considerable cost. In traditional networks, software and hardware are integrated in network equipment. It has been fully tested by the manufacturer. However, in NFV environments, generic hardware is adapted to different software to implement different VNFs, which introduces more complex errors. On the other hand, the compatibility of software and hardware needs to be tested and ensured by SPs, which increases the possibilities of errors compared with the specific hardware. In NFV environments, the types of failures increase and the causes become diverse compared with traditional networks. SPs need a dedicated mechanism to ensure high reliability with relatively low costs, such as resource consumption, overhead, and delay.

1.4 Problem statements

This thesis studies six specific problems about fault-resilience resource allocation in network virtualization.

1.4.1 Resource allocation under deterministic failures

The unavailability of a node, which can be caused by maintenance or failures, interrupts the SFCs that use the VNFs running on the node, so a VNF allocation model that prevents service interruptions is required. Conventional studies presented several allocation models to offset the effect of unavailability. Service providers (SPs) can use backup resources to protect network services [11], or remap the failed VNFs to offset the effect [12].

Regular maintenance and system update are often performed by SPs. SPs can mark specific nodes as being unavailable at a specific time for scheduled maintenance. By using the maintenance schedule, SPs can suppress service interruptions as well as know where and when service interruptions occur and control the positions and time of service interruptions.

The maintenance schedules identify unavailable nodes at each time slot in advance. This work refers to the information that presents the availability of each node at each time slot in future as the *availability schedule* hereafter. The duration of a time slot can be determined according to conditions of network operation, such as the interval of data sampling for fault detection. The number of uninterrupted time slots can be a metric that expresses the service quality of SFCs.

An availability schedule has the position and time of each node availability. Allocations based on the availability schedule can minimize service interruptions. However, no study has addressed an allocation model that uses availability schedules of nodes to prevent network services from being interrupted. This thesis studies this problem in Chapter 3.

1.4.2 Backup resource allocation under deterministic failures

Cold backup only marks each location of backup function at the beginning of the system configuration [13]. A backup function becomes active and starts to consume the computational resource when the corresponding primary function becomes unavailable. The advantage of cold backup is that it does not consume the computational resource of the backup function before it becomes active. The disadvantage is that it needs time to startup and synchronize the state

information for backup and generates an interruption. When a VM monitoring function which checks the status of every VM periodically detects VM failures, the latest snapshots of each VM are used to re-establish the failed VMs in the corresponding backup resources in physical machines [14].

Hot backup keeps an online backup function which synchronizes with the corresponding primary function. It can achieve faster recovery than cold backup, but requires the extra computational and memory resources in the system. Hot backup for service composition in a network is applied in [15]. According to the availability and current state of service composition before the services are interrupted, it restores the service composition dynamically. Hot backup can avoid service interruptions at the cost of extra resource consumption for backup functions.

This work considers hot backup. A backup function needs to be prepared for a period before it is implemented. This period is called *recovery time*. If backup functions are placed and are activated within suitable time slots for preparation instead of all the time slots from the beginning, the interruptions can be suppressed. The extra resource consumption for backup functions can be reduced since we do not need to backup functions from the beginning time slot.

The models introduced in Chapter 3 provided VNF allocation models to suppress the interruptions from the allocations on unavailable nodes and the reallocations. However, it is a passive way to avoid assigning to unavailable nodes and has a limitation decided by the availability schedules. If the model also places backup functions in addition to placing primary functions, the continuous available time slots of SFCs can be extended. This thesis studies this problem in Chapter 4.

1.4.3 Resource allocation under uncertain failures

An availability schedule has the positions and time of node availability. Allocations based on the availability schedule can minimize service interruptions. The works in Chapter 3 and Chapter 4 introduce optimization models to obtain the VNF allocation that maximizes the continuous available time of the SFCs by avoiding service interruptions caused by different VNF allocations

between adjacent time slots [16, 17] and VNF allocations to unavailable nodes under a given availability schedule.

However, in most cases, users cannot know an exact availability schedule whose positions and time of unavailabilities are all deterministic. The actual availability schedule may have some gaps with the maintenance schedule. The starting time of maintenance may be delayed due to the delay of preamble maintenance. The scheduled duration of maintenance may be longer than the real one since conservative estimates are used. The scheduled duration of maintenance may be shorter than the real one since the occurrence of unexpected failures. As a result, availability schedule may mark an availability as unavailability, and vice versa, which influences the VNF allocation. When the results of the maintenance schedule are not credible, we need a model to obtain the VNF allocation with uncertain availability schedules. This work incorporates the uncertainty in the model with deterministic availability schedules in order to make it robust against uncertain availability schedules. This thesis studies this problem in Chapter 5.

1.4.4 Fault-tolerant resource allocation considering function diversity and redundancy

Redundancy is a technique to improve reliability by allocating backup instances that replace unavailable ones, which is naturally resource-consuming. The existing researches on VNF protection tend to provide excessive backup instances for covering improbable scenarios [18]. The work in [10] overcame the above shortcoming by introducing a joint selective VNF diversity and tailored VNF redundancy mechanism to achieve resilient SFC. VNF diversity introduced in [19] uses a group of thin VNF instances with weak processing ability and low resource requirement to replace a single VNF instance, i.e., parallel processing to achieve the same VNF and processing capabilities. We call a thin VNF in the group a *replica*. Different replicas may have different properties and may use different images with different settings for deployments. Lower resource consumption is a feature of thin VNFs and achieved by modifying the functions. Even the same function can have different images depending on their internal parameter settings. VNF diversity can be used to completely prevent

service interruptions to occur. Even some replicas in the group fail, the remaining replicas can perform the function at a lower processing ability. VNF diversity consumes extra resources on load balancers and overheads.

However, the current researches on VNF allocation models consider the VNF diversity and redundancy separately regardless of the end-to-end (E2E) latency which is required to perform the migration from the failed primary functions to backup functions. In case of failure of a stateful function, the current state information of primary and the backup functions need to be synchronized to ensure the sustainability of the service and avoid interruptions. If the synchronization consumes too much time, it can lead to long service interruptions. Reducing the end-to-end communication delay between the primary and backup functions is a key part of reducing the synchronization time consumption. If the replicas in the groups are applied to the redundancy, the protection becomes more flexible than that without using them. If the VNF diversity and redundancy are not considered jointly with an objective to reduce the E2E latency during migration, the users' experience will be degraded due to untimely fault recovery since the backup replicas are separately allocated. A question arises: how can we determine the allocation of VNFs considering redundancy and diversity with E2E latency during migrations when node failures occur? This thesis studies this problem in Chapter 6.

1.4.5 Fault-tolerant resource allocation for dynamic user requirements

Resilience is a key performance indicator for SPs, which is the ability of the network to provide and maintain an acceptable availability level of service in the presence of failures and challenges to normal operation defined in [20]. Functions are decoupled from specialized hardware so that multiple network functions can run on the same commodity server [3]; services are deployed with a set of VNFs. The virtualization makes the services more prone to the failures of not only the physical hardware but also the software of VNFs. The failures interrupt the services and degrade the users' experience.

VNF reliability against failures is widely concerned; fault tolerance provides a metric to evaluate the ability to withstand failures. The work in [21]

introduced fault-tolerance level in the placement of virtual machines. If the placement is set to k-fault-tolerance, the minimum configurations survive at any k host server failures. Fault tolerance in software-based network functions was also considered in [22] to ensure the scalability and availability of network functions and function chains.

VNF instances (VNFIs) with the same function may have different properties against different requirements. The work in [19] introduced VNF diversity which uses a group of VNF instances to replace a single VNF instance. Each VNF instance is termed a *replica* in this paper. Replicas with different resource requirements can utilize the computation resources more efficiently. Even any replica in the group fails, the VNF can maintain a part of processing ability at the cost of performance degradation instead of totally crashing. The application of VNF diversity can increase the feasibility and reliability of function allocation. The work in Chapter 6 introduced an allocation model with considering VNF diversity in order to reduce the recovery time.

SPs provide network services to their users. A service can be a collection of unordered VNFs or a service function chain. A request of service contains several VNFs with each required processing ability. In the dynamic scenarios of arrivals of requested functions and releases of existing functions, a real-time VNF allocation is required to be considered with limited computation time. However, Chapter 6 and the past researches [10] only considered the static VNF diversity and fault tolerance without dynamically arriving requests. The acceptance of dynamic requests should be considered; a request can be accepted when each function in the request can be assigned with sufficient processing ability. As well, the past researches aimed to achieve a fixed fault tolerance, which may lead to no feasible solution due to limited capacity. A question arises: how to perform VNF allocation with considering VNF diversity and dynamic arrival requests, aiming to maximize the acceptance ratio of requests with satisfying the requirements of corresponding resiliency levels?

Some VNF placement optimization problems are proved to be NP-hard [23]. The optimal solutions of the problems are usually obtained with the mixed integer linear programming approach, which leads to intolerably long computation time for a practical larger-size problem. Heuristic algorithms have been introduced to solve the problems in a practical time as the problem size increases

[24–28]. Nonetheless, these heuristic algorithms for solving VNF allocation problems still have some challenges. For example, it requires each independent computation with each arrival request, which is hard to be realized for dynamic requests. Further, it is difficult to capture and handle the characteristics in dynamically arriving requests for these heuristic algorithms. The arriving time, leaving time, arriving orders, required VNFs, and processing abilities of requests are not known in advance, so the heuristic algorithms are difficult to reserve enough resources for the future replica instances.

Reinforcement learning (RL) trains an intelligent agent to learn the policy of choosing an action from the interaction between the agent and the environment based on the Markov decision process (MDP) and maximize the potential rewards. General RL is not efficient in terms of dealing with a large size of state space. Deep reinforcement learning (DRL) combines deep learning with reinforcement learning to overcome the shortage by using deep neural networks (DNNs). DRL has been used in solving the VNF placement problem [29–31], which showed better results compared with those obtained by the greedy and random approaches. Another question arises: how to apply RL to the problem that considers specific aspects, including the dynamic requests, VNF diversity, and k-fault-tolerance? This thesis studies these two problems in Chapter 7.

1.4.6 Implementations of SFCs and resource allocation models

Service functions are widely deployed and used in many networks. Service function chain (SFC) is an ordered set of service functions, which directs the service traffic through these functions in order [5]. In network virtualization, virtual network functions (VNFs) are deployed on physical machines to act as these service functions.

A VNF allocation model is designed to obtain a possible allocation strategy for VNFs in SFCs. Performances of allocation models in Chapters 3–7 need to be evaluated in the network with different parameters of network elements in addition to the mathematical results. It is not cost effective and convenient to perform the evaluation with real network devices. One possible solution is to simulate the whole process for testing the VNF allocation models by software

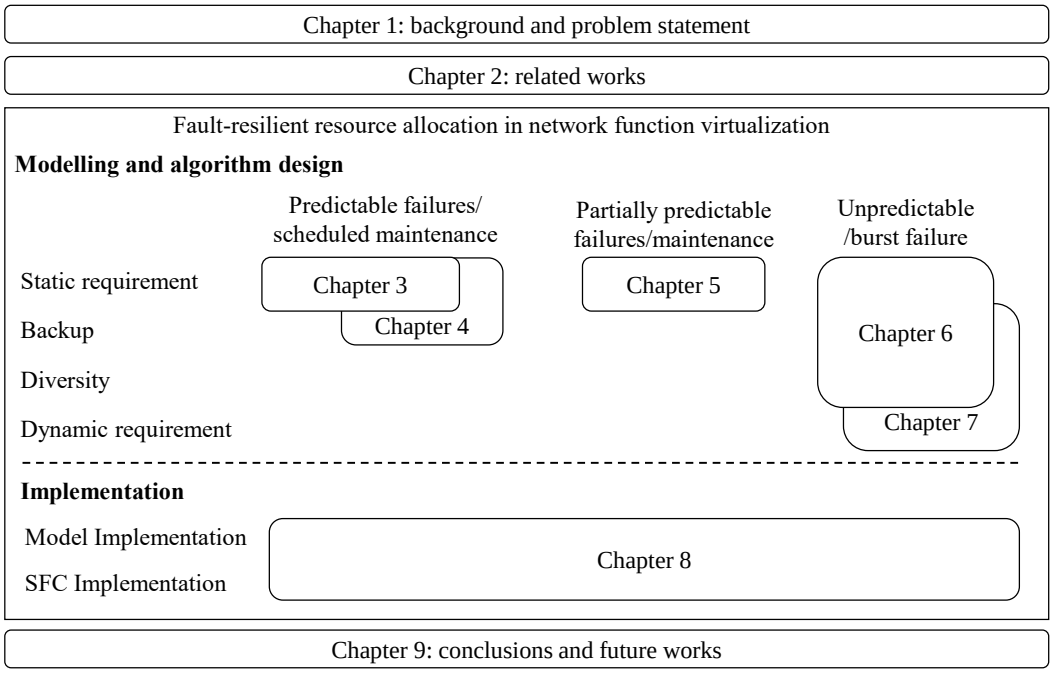


Figure 1.2: Chapter overview.

on a single computer. Computational capability required for such a software should not be so high so that the simulation can run at a low-end computer, such as a laptop. The result of the model is obtained by a mathematical programming approach or a heuristic algorithm. Both of them can run by writing a programming language such as Python. The functions in SFCs need to be allocated according to the computed result in a single application. Two questions arise: How to implement the SFCs in real network environments? How to allocate the resources in real networks according to the results provided by the allocation models? This thesis studies these two problems in Chapter 8.

1.5 Overview and contributions of this thesis

Figure 1.2 shows the chapter overview of this thesis. Chapter 1 introduces the background and the problem statements of this thesis. Chapter 2 surveys the related works in literature. Figure 1.3 shows the relationships among the Chapters 3-8 in terms of the research topics. This thesis discusses the resource

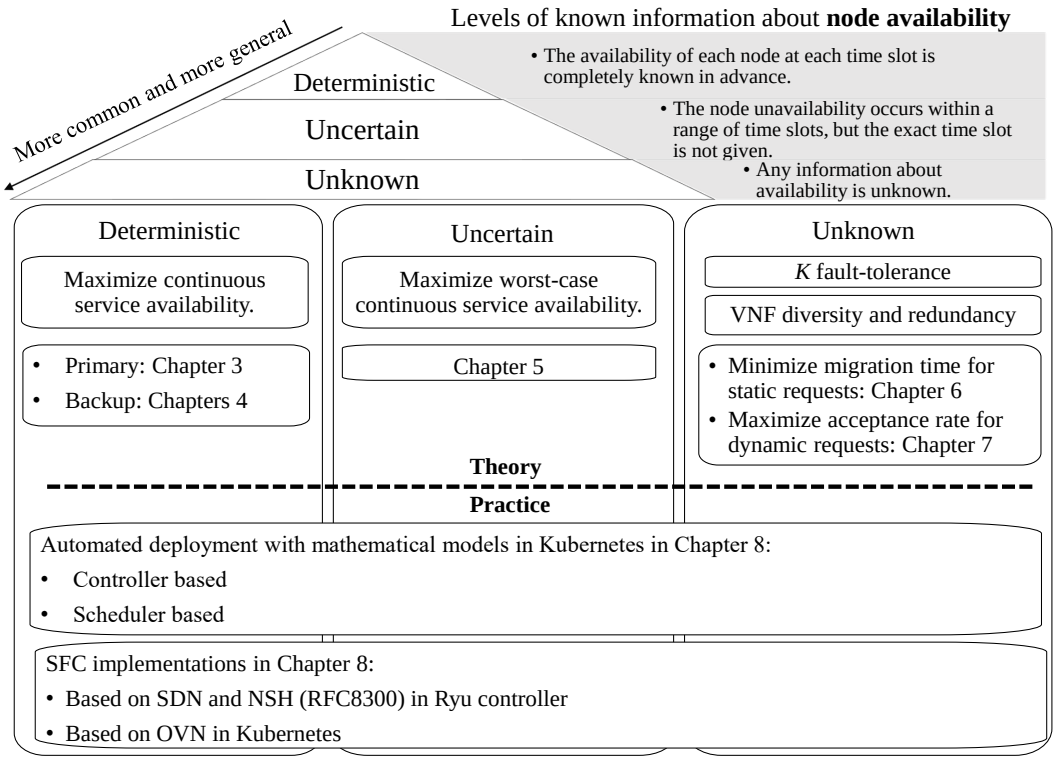


Figure 1.3: Relationships among research topics in Chapters 3-8.

allocation based on three different levels of the known information about the node availability. A deterministic node availability in Fig. 1.3 indicates the time and object of the node unavailable are completely known in advance, e.g., predictable failures and scheduled maintenance in Fig. 1.2. An uncertain node availability in Fig. 1.3 indicates the time and object of the node unavailable cannot be obtained in advance exactly and only the ranges of them are given, e.g., partially predictable failures and maintenance in Fig. 1.2. An unknown node availability in Fig. 1.3 indicates the time and object of the node unavailable are completely unknown in advance, e.g., the unpredictable and burst failures in Fig. 1.2. The commonness of the above three types increases gradually. For solving the resource allocation problems under different levels of node availability information, this thesis considers different aspects in modelings and algorithm designs as well as the implementations of the proposed models. The detail of each Chapter is described as follows.

Chapter 3 proposes an optimization model to obtain the VNF allocation that maximizes the continuous available time of SFCs in a network by avoiding service interruptions. This work assumes that service interruptions have two causes: different VNF allocations between adjacent time slots [17] and VNF allocations to unavailable VMs. This work assumes that network VMs follow the availability schedules precisely. Note that methods to create precise or efficient availability schedules are out of the scope of this paper. For each SFC, this work improves the quality of experience by providing a longer period of stable service. This work defines service continuous available time (SCAT) as the largest number of continuous uninterrupted available time slots for this SFC. For a network containing several SFCs, this work uses the worst-performing service to represent the performance of the network. This work takes the shortest SCAT (SSCAT) among all SFCs in this network as the evaluation metric. The larger SSCAT is, the longer is the available time of the network. This work formulates the model as an integer linear programming (ILP) problem. This work sets the objective function to maximize SSCAT in the network. This work proves the decision version of the VNF allocation problem in the proposed model is NP-complete. This work introduces a heuristic algorithm to solve the proposed model in practical time. Numerical results show that the proposed model provides larger SSCAT values than three existing models.

The position and time of each VM availability in availability schedules influence SSCAT values. Determining routes and VNF allocation simultaneously with the proposed model can identify paths with shorter lengths compared with those determined by processing routes and allocating VNFs separately. The developed heuristic algorithm reduces the computation time compared with the ILP approach with a limited penalty on objective value compared with the ILP-based approach.

Chapter 4 proposes an optimization model to place the primary and backup VNFs in SFCs on nodes based on the model in Chapter 3, which aims to maximize SSCAT in the network under a given availability schedule against service interruptions. Linear SFCs are considered in this paper. This work formulates the problem to maximize SSCAT over the given availability schedule as an ILP problem. This work analyzes the proposed model and derive the upper bound of SSCAT provided by the proposed model. This work provides a heuristic algorithm; the problem becomes more difficult to solve in a practical time as the size of the problem increases. Numerical results compare the proposed model by using the ILP approach with four baseline models. This work also compares the results obtained by the heuristic algorithm and the ILP approach. This work gives an algorithm to estimate the number of bottlenecks of unavailable nodes at different time slots. The proposed model using the backup strategy introduced in this work improves SSCAT compared with the model without using backup functions in examined cases.

Chapter 5 proposes a robust optimization model to obtain the VNF allocation that maximizes SSCAT in the network under an uncertain availability schedule based on the model in Chapter 3. This work considers the uncertain start time slot and period of unavailability on each node in availability schedules. The uncertainty set of availability schedule is given. The robustness of the solution can be controlled. The proposed model obtains the solution by traversing and comparing all possible choices in the uncertainty set limited by the given robustness. This work formulates the problem to maximize SSCAT over the given uncertainty set as an ILP problem, which provides different levels of robustness against uncertain availability schedule. The model handles the uncertainty set and provides an exact solution under the worst-case condition. Numerical results compare the proposed model with three baseline

models. In addition, this work provides a heuristic algorithm to speed up the computation process and introduce an extension for supporting parallelization acceleration. Compared with the existing researches [32–34], the proposed model provides an exact solution of the problem under the uncertainty set.

Chapter 6 proposes a VNF allocation model, which minimizes the sum of the maximum E2E latencies among functions under all possible failure patterns; each failure pattern includes k node failures. The diversity of VNFs is applied to both primary and backup functions. Each VNF has a pool of replicas with different processing abilities and requirements of capacities. The proposed model chooses suitable replicas of primary and backup VNFs from the corresponding pools and allocates them to suitable locations. The allocation provided by the proposed model ensures k-failure tolerance, which can provide enough processing ability to satisfy the requirements even though k failed nodes cannot hold any VNFs. Compared with the previous works, we enable the VNF diversity on the basis of VNF redundancy, which makes resource allocation flexible and full use of resource. This work considers the recovery time under k-resiliency, which can be adopted to different application environments and provide customized service fault tolerances.

Chapter 7 proposes a VNF allocation model to maximize the number of accepted requests which ensures the required resiliency levels. The proposed model chooses suitable replicas of VNFs from the corresponding pools and allocates them to suitable locations. This work uses MDP to formulate the dynamically arriving requests and capture the network variation. This work designs a training environment used for solving the proposed model in an RL approach with dynamically arriving requests. This work compares the proposed model with two baseline models. The results from the examined cases show that the proposed model outperforms two baseline models. Suitable extra input training data may increase the performance of the proposed model.

Chapter 8 consists of two parts on the practices of the introduced allocation models in the previous chapters in real network systems. The first part of Chapter 8 presents the implementations of the allocation models in Kubernetes in two ways: scheduler and controller. Scheduler is a general way for all allocation models. Controller is a method for a specific allocation model which requires unique resource types or control loops. Firstly, this work de-

signs and implements a scheduler to allocate VNFs according to calculation results from multiple reliable function allocation models based on the scheduler framework in Kubernetes. The scheduler provides an intermediate layer to convert the results from different calculators to the pairs of VNFs and nodes in the network. New allocation models can be added to the scheduler without restarting the scheduler. This work prepares two demonstrations of the scheduler, one for multiple allocation models, and the other one for a specific use case in a sensor network. Secondly, this work designs and implements a custom resource controller in Kubernetes based on the processing ability. The custom defined resource (CDR) jointly considers the diversity and redundancy of VNFs, which is called *diversity and redundancy Pod set (DRPS)*. This work uses exact and approximate methods to select suitable replicas from a pool of replica templates to satisfy the required processing ability with the minimum required number of replicas. This work performs demonstrations of the controller including the function allocation and the switching from primary replicas to backups, to show the improvement of server utilization by adopting DRPS.

The second part of Chapter 8 presents the implementations of VNFs in SFCs with allocation models in two network environments: SDN and Kubernetes (container network). Firstly, this work reports an implementation of the network service header (NSH) [121] based SFC application with the cooperation of the allocation model. This work implements an NSH-based SFC application on Ryu software-defined networking framework [35]. This work implements the functions of classifier, service function forwarder, and SFC proxy [5] on switches by the modification of flow tables which is conducted by the application. This work uses the application to simulate the VNF allocation and the traffic through these functions. The network devices are simulated in Mininet [36], which is connected to the application as a controller. It receives the registration message from allocated VNFs and instructs the actions of switches when they are necessary. As a result, the application allocates the VNFs to corresponding servers automatically and the path of each SFC is configured correctly. Secondly, this work reports an implementation of SFCs with the cooperation of the SFC allocation models in Kubernetes. This work modifies the open virtual network (OVN) [37] so that it can support the rout-

ing for SFC flow, e.g., classifier, service function forwarder, and load balancer for multiple ports. This work applies the customized OVN to Kubernetes as a container network interface (CNI) plugin based on Kube-OVN [38]. This work adds the definition of the SFC resource and its controller in Kubernetes. This work prepares a demonstration to show that the functions in SFCs are allocated to corresponding nodes automatically and the path of each SFC is configured correctly.

Finally, Chapter 9 concludes this thesis and discusses the future works to extend this work.

Chapter 2

Related works

2.1 VNF allocation considering service interruptions

VNF placement needs to be determined in order to provide services to users by using SFCs. Several works tackled the VNF placement problem for configuring SFCs [39–41]. Different studies use different objectives in the VNF placement problem: minimizing the latency of services [39], minimizing the total cost for VNF placement and link utilization [40], and minimizing the resource consumption while ensuring the requested quality of services [41]. The VNF placement models presented in the above works determine the placement regardless of time slots or node unavailability. There are two possible scenarios that the models face in tackling the VNF placement problem in a network whose nodes can be unavailable in some time slots. The first scenario applies the model to determine continuous VNF placement; the VNF allocation is kept from the first time slot to the last time slot. In this scenario, service interruptions can occur in some time slots due to node unavailability. The second scenario applies the models to determine VNF placement in each time slot independently. This scenario can avoid node unavailability in each time slot, but service interruptions can occur due to VNF reallocation between adjacent time slots. The VNF placement model proposed in this paper determines the VNF allocation in a given set of time slots, which suppresses the service interruptions due to either node unavailability or VNF reallocation.

Table 2.1: Comparison of conventional models on VNF allocation considering service interruptions.

Literature	Handled issue	Migration object	Objective	Motivation	Optimization problem	Solution
[42]	In order to guarantee the required reliability, an incremental approach was introduced to determine the number of required VNF backups with using resource sharing.	VNF	Minimize the network resource consumption.	Any failure of VNFs could break down an entire service chain, thus interrupting the service, the reliability must be enhanced.	ILP	Solved by optimization tool (CPLEX) and a bi-directional search-based greedy algorithm.
[43]	The allocation model of a virtual network function forwarding graph in a network where the traffic is changing over time.	VNF	Minimize the cost of reallocation, the migration distance and transmission distance are considered.	The effects of variations of networks over time have not been addressed.	ILP	Solved by an alternating direction method of multipliers-based algorithm.
[44]	Trade-off between the resources for VNF replica and number of migrations.	VM	Minimize the sum of migration, server and link costs with given weights.	The necessary migrations of VNFs cause service interruptions and reduce quality of service.	LP	Solved by optimization tool (Gurobi).
[45]	Trade-off between the reconfiguration of SFCs and the optimality of the resulting placement and (re)routing.	VNF	Minimize the new placement cost, and the reconfiguration cost with given weights.	Dynamic VNF placement with changing load.	INLP	Solved by optimization tool (MATLAB toolbox YALMIP and Gurobi).
This work	Improve the continuous available time for services with availability schedule.	VNF	Maximize the shortest service continuous available time among all services in the network.	Reallocation and unavailable nodes impact on the continuous availability of services and influence quality of experience.	ILP	Solved by optimization tool (CPLEX) and a genetic algorithm based algorithm.

ILP: integer linear programming, LP: linear programming, INLP: integer nonlinear program.

Node unavailability and the service interruptions caused by VNF and VM migrations impact the performance of network and services. The impact has been addressed previously in various contexts. The work in [42] concerns the low reliability of softwarized networks caused by the service interruptions. In [42], the authors introduced a model for VNF placement and service flow routing. To overcome the service interruptions and increase the network reliability, the model uses VNF backups with sharing the backup resources. The work in [43] introduced a model of the adaptive and dynamic VNF allocation problem considering also VNF migration in a network where the traffic is changing over time. The model presented in [44] addressed to minimize three different costs: migration, server, and link costs. The cost of migration is evaluated by the number of migrations. The model aims to reduce the number of migrations and improve the quality of service. The work in [45] introduced a model for dynamic VNF placement under changing traffic load. A static placement decreases the operation cost. Reconfiguration causes service interruption and the operation cost increases. The model presented in [45] minimizes the new placement cost, and the reconfiguration cost with given weights. We summarized the comparison between the conventional models and the proposed model on issues, migration objects, objectives, motivations, optimization problems, and solutions in Table 2.1.

Compared with conventional models which also care about the interruption of services, the objective of this work specializes in the continuous available time, which is aware of the quality of experience for users. The objectives of the conventional models are costs, which are abstract concepts for users. Moreover, this work does not need extra server resource including storage and computation resources compared with conventional models. This work considers a sequence of time slots so that the model only calculates the placement at the very beginning of service deployments instead of at each time slot.

In this work, the contribution is to achieve the maximum continuous available time of services with the consideration of availability schedules. The proposed model determines the VNF locations in a sequence of time slots. Compared with the conventional researches, this work considers the interruptions which influence the longest continuous available time of services. This work does not need extra storage resources, such as duplicated functions. It

is used for deploying disruption-sensitive functions in projects with limited number of VMs.

2.2 VNF allocation considering primary and backup resources jointly

Several studies considered the backups in SFC. The works in [46, 47] studied the placement methods for active VNFs and backup VNFs with flow and SFC parallelism. The models split a large flow into multiple parallel smaller sub-flows and any SFC is replicated into multiple sub-SFCs. The models aimed to increase service reliable probability while fewer backup resources are required. The work in [48] designed a redundancy mechanism to protect the service from interruptions by introducing a node-ranking algorithm. The mechanism reduces the consumption of backup resources with respect to a higher acceptance ratio of SFC requests. The work in [49] introduced a backup model that combines path backup and VNF backup in a joint way. The backup model aimed to reduce resource consumption for backups. The work in [50] introduced a method to allocate SFCs aiming to maximize the number of SFC requests that can be served while meeting their heterogeneous availability requirements, which includes an ILP model for one SFC and a heuristic algorithm for mapping multiple SFC requests. The work also introduced a backup pooling mechanism to further improve the efficiency of backup resource usage. The work in [51] presented a framework to provide availability of SFC requests with the objective of minimizing resource usage including an optimization problem of SFC mapping and backup estimation. The work in [52] introduced a coordinated protection mechanism that adopts both backup path protection in the network and VNF replicas at nodes to guarantee an SFC's availability aiming to reduce the SFC blocking and the cost of computational resources.

Compared with the existing studies, the objective of this work is different. The existing studies care about end-to-end reliability, which is evaluated by the metrics such as an available probability or a mean time between failures. This work cares about time sensitive services which are evaluated by the continuous servable time. In addition, the existing models only consider one possible error

pattern which may lead to service interruptions at one time. This work considers the error patterns in a sequence of time slots and the reallocations between different allocations in the adjacent time slots. Moreover, this work gives an initial allocation for VNFs in SFCs. It focuses on the optimal allocation at the beginning of function deployment instead of after a disturbance.

2.3 Robust optimization and applications in network

In the past researches, robust techniques were applied to deal with uncertainty in the problems. To deal with data uncertainty in linear programming, Soyster [53] introduced a linear optimization model to construct a solution that is feasible for all data that belong to a convex set, which is too conservative and gives up much of the optimality for the problem. To overcome the over-conservation, Ben-Tal et al. [54] introduced less conservative models by considering uncertain linear problems with ellipsoidal uncertainties. However, the above models are designed for convex uncertainty sets. In our model, the uncertainty set is not a convex set, which is discrete. To deal with robust discrete optimization problems, Bertsimas et al. [55] introduced an approach for robust linear optimization problem based on [53]; it offers an ability to control the degree of conservatism for each constraint.

Robust optimization techniques have been applied to different network design problems. The work in [55, 56] introduced a robust integer programming problem with the data uncertainty in network flow problems and solved the minimum cost flow problems. The work in [32] introduced a problem of backup network design for general link loads, where the uncertainty is the number of primary links that fail. The work in [57] applied the robust optimization to primary and backup allocation problem, where the number of failing PMs is given but which PM fails is uncertain. The work in [34] adopted the robust optimization technique on minimizing the required backup capacity with probabilistic protection against multiple PM failures, where the uncertainty of capacity was considered. The above researches consider the discrete uncertainty sets and provide an approximate solution. This work provides an exact

solution without gaps by taking advantage of the limited size of uncertainty set in the proposed model.

2.4 VM/VNF allocations considering fault tolerance

In conventional researches, the protection of VMs has been studied. The work in [21] presented a VM allocation model that establishes a redundant configuration against host server failures with fewer host machines by achieving *k-resiliency* for VMs. k-resiliency is defined as an availability property of a VM, which can protect the VM against at most k physical machine failures. When a VM is marked as k-resilient, as long as there are k host failures, k-resilient guarantees that it can be relocated to a non-failed host without affecting any other VMs. In other words, an allocation model that considers k-resilient provides k-fault tolerance assurance. The work in [58] modelled the k-resiliency with the objectives including achieving high availability and constraints such as resource feasibility. The work in [59] introduced an online redundant VM allocation model aiming to minimize the consumption of network resources when primary VM failures need be recovered by backup VMs under the k-fault tolerance constraint.

The resiliency protection for VMs is coarse-grained if SFCs are considered. There are several ordered VNFs in SFCs instead of independent VMs. VNF increases the flexibility of function deployments in the network. The protection for SFC is required to consider the inner connections between VNFs in each SFC. The works in [60] and [61] focused on the SFC allocation model to minimize the transmission delay for latency-sensitive services. Redundant VNFs and links are considered to prevent the decline of QoS caused by node and link failures. These works consider the routing of SFCs and the partly and fully ordered VNFs in SFCs. The conventional researches on the resilient VNF allocation model in the NFV environment [62, 63] focused on the availability requirement regardless of the resource optimization.

SFC reconfiguration is a method to realize resilient SFC provisioning in previous studies. The work in [64] introduced an ILP model for allocating SFCs

dynamically and adaptively, whose allocations can be readjusted. It aimed to make a trade-off between resource consumption and operation cost. The resilient SFC allocation with considering SFC reconfiguration can bring extra cost and may generate service interruptions [65]. Suitable initial allocations which consider resiliency in advance can reduce the cost on readjustments and improve the quality of user experience by reducing the potential service interruptions.

The resilient VNF placement problems considered in existing researches have different objectives. The work in [66] presented an approach to determine the allocations of VNFs in SFCs aiming to minimize the number of affected SFCs upon a node failure. The work in [67] minimizes the cost of used computing resources by sharing computing resources among multiple service chains. The work in [52] considered VNF replication to improve the service availability. It determines the number of replicas for each VNF and allocates them to suitable locations aiming to reduce the SFC blocking and the cost of computing resources.

The redundancy of VNF links and instances is a common way to guarantee the resiliency of service function chains. The work in [68] studied reliable service provisioning through redundant placement of instances of VNFs. The model maximizes the number of requests admitted, while meeting the specified reliability requirement of each admitted request. A randomized algorithm was introduced to solve the problem in large-scale cases. The work in [69] introduced a model to deploy both active and backup VNF instances while guaranteeing the required VNF service resilience. The objective is to minimize the total expected transmission delay of all flows.

VNF diversity uses a group of replicas to replace one replica, which is more vulnerable to failures. Different replicas can have different properties, e.g., requirements of resources and providable processing abilities. The work in [10] allocates the VNFs in SFCs while meeting the target SFC availability level with considering the VNF diversity as well as the VNF redundancy. The model aims to reduce the inherent cost from overheads and redundant resources. However, it did not apply the diversity on redundant resources.

Fat-tree is a classic topology for data center networks, which typically consists of trees with three levels of switches [70]. There are three levels of switches,

core, aggregation, and edge, from the top to the bottom in each fat-tree network topology. Physical servers are connected to the network through edge switches. The upper-level switches are more likely to become congested than the lower level switches since the upper switches transfer more data. The data transferring between different domains or data centers may have additional costs and increase the transmission delay. Therefore, it is a problem to reduce the network resource consumption of the upper-level links [59]. VNFs need to re-fetch the data from the databases on storage servers when it fails, which may consume both edge level and aggregation level, or even the core level network resources if the backup VNF and failed VNF are located in different domains. If a node failure is caused by software, the failed node can store a copy of the data so that the recovered VNF does not need to re-fetch the original data from the databases. For an SFC, the recovered VNF does not need to re-fetch the data from upstream VNFs and the data may avoid being recalculated by upstream VNFs. Distributing VNFs can provide operation diversity and reduce the possibility of concurrent failures. However, the E2E latency is increased if the VNFs are placed far from each other redundantly [71]; this influences the quality of services with applying the diversity and redundancy VNFs in the data center networks, if the allocation strategy has not been designed properly.

Compared with the existing studies, this work applies VNF diversity and redundancy jointly, which provides more flexible function allocation and higher resource utilization compared with considering diversity and redundancy separately. The diversity on both primary and backup resources can fully utilize the resources by dividing a VNF instance into several thin replicas, which is useful for resource-limited devices, e.g., edge servers. In terms of the objective functions, this work takes the E2E latency into consideration, which has also been concerned by other studies. Table 2.2 summarizes the comparison between the conventional models and the proposed model.

2.5 VNF allocation problem solved by RL

A VNF placement problem is proved to be an NP-hard problem, whose globally optimal solution is difficult to be found in polynomial time [72]. RL provides an approach to find the approximate solutions of the VNF allocation prob-

Table 2.2: Summary of related work on VM/VNF allocations considering fault tolerance.

Literature	Protection object	Diversity	Redundancy	Objective	Solution
[10]	VNF	✓	✓	Minimize required resources of VNFs	MILP
[21]	VM		✓	Minimize number of used virtual machines	Greedy
[58]	VM		✓	Find an allocation satisfies required reliability	Constraint programming
[59]	VM		✓	Minimize network consumption and data transfer delay	Heuristic
[66]	VNF		✓	Minimize maximum impact of a hosting PM failure	ILP
[67]	VNF		✓	Minimize total cost of using computation resources	ILP
[52]	VNF		✓	Reduce SFC blocking and cost of computing resources	Heuristic
[68]	VNF		✓	Maximize the number of requests admitted	ILP, randomized, and heuristic
[69]	VNF		✓	Minimize total transmission delay of all flows	Greedy
This work	VNF	✓	✓	Minimize total maximum E2E latency among functions of all error patterns and resource consumption	MILP, greedy, and probabilistic heuristic

ILP: integer linear programming, MILP: mixed ILP.

lems within a practical time. DRL has been used in solving VNF allocation problems, especially for dynamic cases. The work in [29] adapted an RL-based allocation approach on jointly minimizing the operation cost of service providers and maximizing the total throughput of dynamic requests. The work in [30] presented an RL-based online method to place the active and standby SFCs with handling the dynamic network state transitions in real-time. The work in [31] studied an RL-based dynamic SFC allocation method with using monitored resource information.

Compared with the existing studies, this work considers the number of accepted requests among a large amount of arriving requests while allocating them to the nodes with limited resources ensuring the fault-tolerance performance of the dynamic requests with random requirements. This work uses an RL-based approach to handle real-time requests and efficiently and automatically learns to allocate resources for arriving requests with different requirements on resiliency and processing abilities.

Chapter 3

Resource allocation model for deterministic availability schedules

This chapter proposes a VNF allocation to maximize the continuous available time of SFCs in a network against service interruptions considering deterministic availability schedules [73, 74].

The remainder of the chapter is organized as follows. Section 3.1 describes the application scenario and the selection of metrics in the proposed model. Section 3.2 describes the model. Section 3.3 introduces a heuristic algorithm that solves the proposed model in realistic time. Section 3.4 presents numerical results that show the performance of the proposed model in different cases. Section 3.5 discusses how to deal with the incoming requests at a new set of time slots and the applicability of the proposed model in the system. Section 3.6 summarizes the key points of this chapter.

3.1 Motivation

Before the services, SPs need to deploy the VNFs on VMs. During the running of services, any failure of VNFs may break down an entire service chain, interrupting the service [42]. Changing the resource allocation of VNFs, which causes the reallocations of VNFs, interrupts network services [43, 75]. VNFs

are service-chained, and migration of one VNF in a chain can interrupt the entire service chain [44]. The interruption of network services influences the quality of experience. If a sequence of continuous time slots is considered, the interruptions divide the sequence into several small sequences. Each sequence means a continuous available time of the service. This work takes the longest one among these sequences, which means the longest continuous available time of the service, as a main metric in this work, which is called SCAT. Improving SCAT is useful for disturbance sensitive functions, such as file transferring or video chatting, or the functions with a high cost of recovery. An allocation model which can increase SCAT is necessary.

There are several services in a network. Because of the limitation of VM capacity, the SCATs of all services cannot reach the highest value at the same time. This work considers the service with the worst SCAT and considers maximum SSCAT as the objective in the proposed model. The challenge is how to design a model to maximize SSCAT by taking advantage of the given availability schedule for the initial allocation of VNFs.

With the introduction of a certain availability schedule from maintenance schedule, users can get the locations of unavailable VMs in the network during a period of time [76] in advance. By taking advantage of the given availability schedule, the allocation of VNFs before service running can suppress the interruptions caused by unavailabilities and reallocations during the given period of time so that SSCAT can be increased.

3.2 Problem formulation

3.2.1 VNF allocation

Consider a virtual network, $G(N, L)$, which is made up of a set of virtual nodes, N, and a set of directed virtual links connecting these VMs, L. This work considers a set of different types of resources S for each node, such as CPU, memory, and storage. Node $n \in N$ at time slot $t \in T$ has c_{nt}^{s} units of available resources of resource $s \in S$ in total. Each virtual link $(i, j) \in L$ corresponds to a connection between two VMs with transmission resource b_{ij} and length l_{ij}.

R is the set of SFCs, which represent SPs' requirements. Each SFC contains several VNFs. According to the SPs' requirements, VNFs are allocated to VMs and connected to each other in some specified order. Let K_r be the set of ordered VNFs in chain $r \in R$, each of which is allocated with an index in the range of $[1, |K_r|]$. Instances of any VNF type can be deployed in a VM. The VNF instance of the $k \in K_r$th function of request $r \in R$ placed on a VM occupies q_s^{rk} units of resource $s \in S$. Several different functions from different requests can be assigned to the same node, each of which utilizes certain computing and transmission resources. The transmission resource demanded by request r is d_r.

This work divides continuous time into discrete time slots. A set of time slots is represented by $T = [1, |T|]$. The node availability at each time slot $t \in T$ is given by the availability schedule. This work uses binary parameter e_t^n to express elements in the availability schedule: if node $n \in N$ is unavailable at time slot t, $e_t^n = 1$, and otherwise 0.

The decision variables in the ILP problem are represented as follows. This work uses binary decision variable x_{tn}^{rk} to represent the allocation. x_{tn}^{rk} is set to 1 if the kth function of request r is assigned to node n at time slot t, and 0 otherwise.

$o_t^r, \forall r \in R, t \in T$, denotes a binary variable; if the allocation of at least one function in request r is changed between time slot $t-1$ and time slot t or any VNF of request r at time slot t or $t-1$ is allocated to an unavailable node, o_t^r is set to 0, and 1 otherwise. When $t = 1, o_t^r = 0$; otherwise, o_t^r can be calculated by:

$$o_t^r = \prod_{k \in K_r} \prod_{n \in N} ((x_{tn}^{rk} \odot x_{t-1,n}^{rk}) \wedge (1 - x_{tn}^{rk} e_t^n) \wedge (1 - x_{t-1,n}^{rk} e_{t-1}^n)), \forall r \in R, t \in T \backslash \{1\}. \tag{3.1}$$

$\odot$ means exclusive NOR operation between two binary variables whose operations are: $1 \odot 1 = 1, 0 \odot 0 = 1, 1 \odot 0 = 0, 0 \odot 1 = 0$. $\wedge$ means the multiplication of two binary variables whose operations are: $1 \wedge 1 = 1, 0 \wedge 0 = 0, 1 \wedge 0 = 0, 0 \wedge 1 = 0$.

In the definition, $x_{tn}^{rk} \odot x_{t-1,n}^{rk} = 1$ means that the kth function in request r is not allocated to node n at time slots t and $t-1$ or is allocated to node n at time slots t and $t-1$; 0 otherwise. $1 - x_{tn}^{rk} e_t^n = 0$ means that the kth function in request

r is allocated to an unavailable node n at time slot t; 1 otherwise. $1-x^{rk}_{t-1,n}e^n_{t-1} = 0$ means that the kth function in request r is allocated to an unavailable node n at time slot $t-1$; 1 otherwise. $(x^{rk}_{tn} \odot x^{rk}_{t-1,n}) \wedge (1-x^{rk}_{tn}e^n_t) \wedge (1-x^{rk}_{t-1,n}e^n_{t-1}) = 1$ means that all the three elements are true; otherwise 0. If it is true for all $n \in N$ or $\prod_{n\in N}((x^{rk}_{tn} \odot x^{rk}_{t-1,n}) \wedge (1-x^{rk}_{tn}e^n_t) \wedge (1-x^{rk}_{t-1,n}e^n_{t-1})) = 1$, the allocation of kth function in request r is not changed between time slots t and $t-1$ and the function is not allocated to an unavailable node at time slots t and $t-1$; 0 otherwise. If it is true for all $k \in K_r$, $o^r_t = 1$; otherwise 0. For example, in Fig. 3.1, $o^3_2 = 0$, because of the reallocation between time slots 1 and 2. $o^2_4 = o^1_4 = 0$, because of the functions in requests 1 and 2 are allocated to an unavailable node at time slots 4.

$z^{jr}_i, i \in T, j \in T_i, r \in R$, denotes a binary variable; if allocations of request r from i to $i+j-1$ are consecutively unchanged, or j consecutive o^r_t are all 1 from $t = i$ to $t = i+j-1$, z^{jr}_i is set to 1, and otherwise 0. Let $T_i = [1, |T|-i+1] \subseteq T$ be a set of time slots from 1 to $|T|-i+1$. z^{jr}_i is constrained as follows:

$$(\sum_{t=i}^{i+j-1} o^r_t) - j + 1 \leq z^{jr}_i, \forall i \in T, j \in T_i, r \in R, \tag{3.2}$$

$$z^{jr}_i \leq \frac{\sum_{t=i}^{i+j-1} o^r_t}{j}, \forall i \in T, j \in T_i, r \in R. \tag{3.3}$$

$y^r_j, j \in T, r \in R$, denotes a binary variable; if the allocations are consecutively unchanged during j time slots existing in T, y^r_j is set to 1, and otherwise 0. y^r_j is constrained as follows:

$$z^{jr}_i \leq y^r_j, \forall i \in T, j \in T_i, r \in R, \tag{3.4}$$

$$y^r_j \leq \sum_{t=1}^{|T|-j+1} z^{jr}_t, \forall j \in T, r \in R. \tag{3.5}$$

$\beta_r \in [1, |T|], r \in R$ is an integer variable that represents the SCAT of request r, i.e., the maximum number of continuous available time slots in request r. β_r is given by:

$$\beta_r = \max_{j\in T}\{jy^r_j\} + 1, \forall r \in R. \tag{3.6}$$

λ is an integer variable that represents SSCAT, i.e., the minimum number of longest continuous available time slots among all requests. λ can be expressed by:

$$\lambda \leq \beta_r, \forall r \in R. \tag{3.7}$$

The objective function of the proposed model is given by:

$$\max \lambda + \epsilon_1 \sum_{r \in R} \beta_r. \tag{3.8}$$

In this problem, the solution that maximizes the sum of continuous available time slots for all requests, $\sum_{r \in R} \beta_r$, is chosen when there are multiple solutions that maximize λ. Therefore, the small number, ϵ_1, is multiplied to the second term to prioritize the first term over the second term. ϵ_1 is given by $\frac{1}{|R| \cdot |T|}$.

This work uses an example shown in Fig. 3.1 to explain the meanings of service interruptions, SCAT, and SSCAT. In Fig. 3.1, there are five nodes in the network. There are three SFCs that contain three, two, and two functions, respectively. f_x^y represents the yth function of chain x. If the functions are allocated according to Fig. 1, service provided by chain 3 is interrupted at time slot 2 due to the different VNF allocations triggered by the unavailability of node 4 at time slot 2. VNF allocated to an unavailable node at time slots 4 and 5 interrupt chains 1 and 2. The SCAT of chains 1, 2, and 3 are three, three, and five, respectively. SSCAT is the smallest number among them, i.e., three.

This work also considers several constraints as below.

The node capacity constraint is given by:

$$\sum_{r \in R} \sum_{k \in K_r} x_{tn}^{rk} \cdot q_s^{rk} \leq c_{nt}^s, \forall n \in N, t \in T, s \in S. \tag{3.9}$$

Each node, because of the computational resource limitation, can carry only a limited number of functions. Equation (3.9) ensures that each node's computational resources must not be overused during allocation.

The assignment constraint is given by:

$$\sum_{k \in K_r} x_{tn}^{rk} \leq 1, \forall r \in R, n \in N, t \in T, \tag{3.10}$$

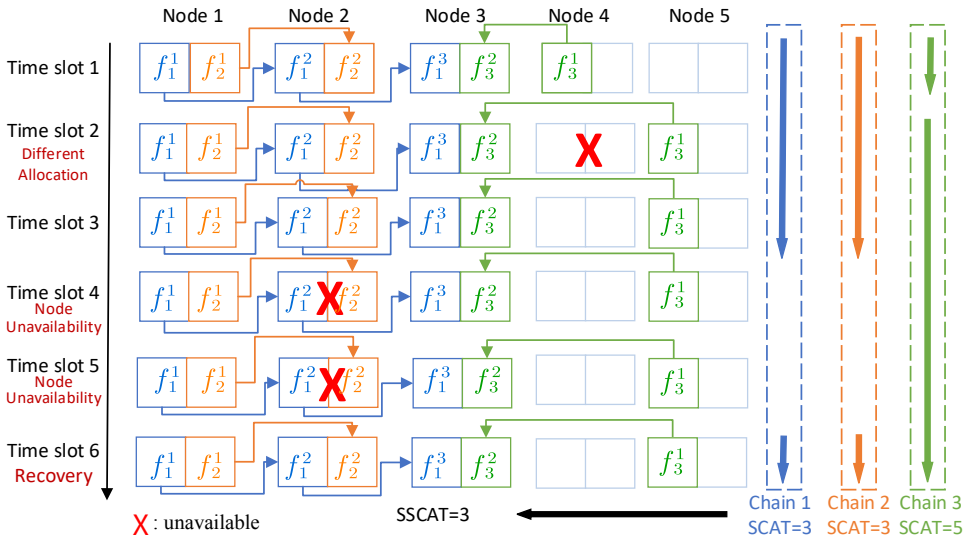


Figure 3.1: Example of SSCAT.

and

$$\sum_{n \in N} x_{tn}^{rk} = 1, \forall r \in R, k \in K_r, t \in T. \tag{3.11}$$

Equation (3.10) assumes that one service chain does not allocate multiple VNFs in this chain on one VM in case of the influence of the reallocation of VMs [44]. Equation (3.11) ensures that all functions are allocated in the network.

If it is necessary to avoid allocating functions to unavailable nodes, this work adds the following constraint:

$$x_{tn}^{rk} e_t^n = 0, \forall r \in R, t \in T, n \in N, k \in K_r. \tag{3.12}$$

According to the linearization process in Appendix A, (3.6) is linearized to (3.13)-(3.18), and (3.1) is linearized to (3.19)-(3.37) with some auxiliary variables as follows:

$$\beta_r - 1 \leq j y_j^r + \left(1 - \delta_j^r\right) \cdot B, \forall j \in T, r \in R, \tag{3.13}$$

$$\beta_r - 1 \geq j y_j^r - \left(1 - \delta_j^r\right) \cdot B, \forall j \in T, r \in R, \tag{3.14}$$

$$j y_j^r \geq (\delta_j^r - 1) \cdot B + j' y_{j'}^r, \forall j \in T, j' \in T \backslash \{j\}, r \in R, \tag{3.15}$$

$$\sum_{j \in T} \delta_j^r = 1, \forall r \in R, \tag{3.16}$$

$$\beta_r - 1 \geq j y_j^r, \forall j \in T, r \in R, \tag{3.17}$$

$$\delta_j^r \in \{0,1\}, \forall j \in T, r \in R, \tag{3.18}$$

$$\phi_{tn}^{rk} = 1 - x_{tn}^{rk} - x_{t-1,n}^{rk} + 2 \cdot h_{tn}^{rk}, \forall r \in R, t \in T \backslash \{1\}, k \in K_r, n \in N, \tag{3.19}$$

$$h_{tn}^{rk} \leq x_{tn}^{rk}, \forall r \in R, t \in T \backslash \{1\}, k \in K_r, n \in N, \tag{3.20}$$

$$h_{tn}^{rk} \leq x_{t-1,n}^{rk}, \forall r \in R, t \in T \backslash \{1\}, k \in K_r, n \in N, \tag{3.21}$$

$$h_{tn}^{rk} \geq x_{tn}^{rk} + x_{t-1,n}^{rk} - 1, \forall r \in R, t \in T \backslash \{1\}, k \in K_r, n \in N, \tag{3.22}$$

$$\theta_{tn}^{rk} \leq \phi_{tn}^{rk}, \forall r \in R, t \in T \backslash \{1\}, k \in K_r, n \in N, \tag{3.23}$$

$$\theta_{tn}^{rk} \leq 1 - x_{tn}^{rk} e_t^n, \forall r \in R, t \in T \backslash \{1\}, k \in K_r, n \in N, \tag{3.24}$$

$$\theta_{tn}^{rk} \geq \phi_{tn}^{rk} - x_{tn}^{rk} e_t^n, \forall r \in R, t \in T \backslash \{1\}, k \in K_r, n \in N, \tag{3.25}$$

$$\pi_{tn}^{rk} \leq \theta_{tn}^{rk}, \forall r \in R, t \in T \backslash \{1\}, k \in K_r, n \in N, \tag{3.26}$$

$$\pi_{tn}^{rk} \leq 1 - x_{t-1,n}^{rk} e_{t-1}^n, \forall r \in R, t \in T \backslash \{1\}, k \in K_r, n \in N, \tag{3.27}$$

$$\pi_{tn}^{rk} \geq \theta_{tn}^{rk} - x_{t-1,n}^{rk} e_{t-1}^n, \forall r \in R, t \in T \backslash \{1\}, k \in K_r, n \in N, \tag{3.28}$$

$$w_t^{rk} \leq \pi_{tn}^{rk}, \forall r \in R, t \in T \backslash \{1\}, k \in K_r, n \in N, \tag{3.29}$$

$$w_t^{rk} \geq \sum_{n \in N} \pi_{tn}^{rk} - |N| + 1, \forall r \in R, t \in T \backslash \{1\}, k \in K_r, \tag{3.30}$$

$$o_t^r \leq w_t^{rk}, \forall r \in R, t \in T \backslash \{1\}, k \in K_r, \tag{3.31}$$

$$o_t^r \geq \sum_{k \in K_r} w_t^{rk} - |K_r| + 1, \forall r \in R, t \in T \backslash \{1\}, \tag{3.32}$$

$$o_1^r = 0, \forall r \in R, \tag{3.33}$$

$$o_t^r \in \{0,1\}, \forall r \in R, t \in T, \tag{3.34}$$

$$y_j^r \in \{0,1\}, \forall r \in R, j \in T, \tag{3.35}$$

$$w_t^{rk} \in \{0,1\}, \forall r \in R, k \in K_r, t \in T \backslash \{1\}, \tag{3.36}$$

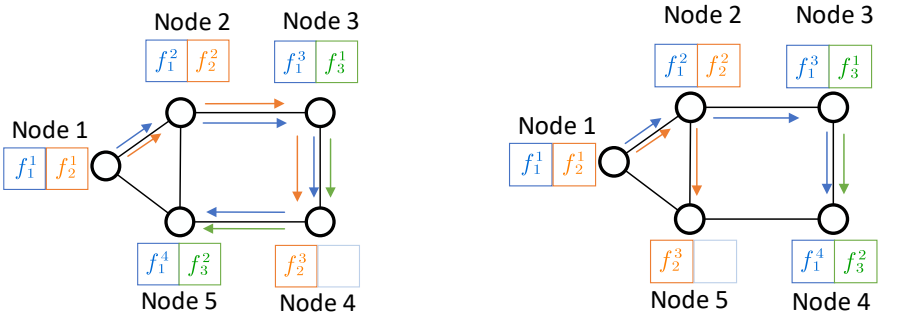
$$\phi_{tn}^{rk}, x_{tn}^{rk}, h_{tn}^{rk}, \theta_{tn}^{rk}, \pi_{tn}^{rk} \in \{0,1\}, \forall r \in R, k \in K_r, t \in T \backslash \{1\}, n \in N. \tag{3.37}$$

In (3.13)-(3.15), B is a given constant value that satisfies $B > j y_j^r, \forall j \in T, r \in R$; B can be taken to be $|T| + 1$.

If the users are more concerned about VNF allocation than the routes of the SFCs, this work introduces the following model:

$$\max \ \lambda + \epsilon_1 \sum_{r \in R} \beta_r \tag{3.38a}$$

$$\text{s.t.} \ \ (3.2) - (3.5), (3.7), (3.9) - (3.11), (3.13) - (3.37) \tag{3.38b}$$



(a) Path determination after function deployment

(b) Path determination during function deployment

Figure 3.2: Example of computing routes.

$$\lambda \in [1, |T|] \tag{3.38c}$$

$$\beta_r \in [1, |T|], \forall r \in R, i \in T, j \in T_i \tag{3.38d}$$

$$z_i^{jr} \in \{0,1\}, \forall r \in R, i \in T, j \in T_i. \tag{3.38e}$$

3.2.2 Route determination

The model described above determines VNF allocation. Sometimes the route of each SFC needs to be considered. There are two possible ways to determine the route of each SFC. In the first way, the route of each SFC is computed separately after VNF allocation is computed. In the second way, the VNF allocation and the routes of all SFCs are determined at the same time. Fig. 3.2 shows examples of VNF allocation and SFC routes; each SFC has shorter route length in Fig. 3.2(b) than in Fig. 3.2(a). By computing the VNF allocation and the routes of all SFCs at the same time, shorter SFC routes can be obtained with consideration of the demands placed on transmission resources.

This work presents the following constraints to compute the VNF allocation and the routes of all SFCs.

The flow constraint is given by:

$$\sum_{(i,j)\in L} a_{w,ij} p_{rt}^{k,ij} = \begin{cases} -1, \text{ if } x_{tw}^{rk} = 1 \\ 1, \text{ if } x_{tw}^{r,k+1} = 1 \\ 0, \text{ if } x_{tw}^{rk} = x_{tw}^{r,k+1} = 0. \end{cases} \tag{3.39}$$

For each request, there are three types of nodes: source node (nodes allocated the first function of a request), destination node (nodes allocated the last function of a request) and others. This work defines indicator $a_{w,ij}, w \in N, (i,j) \in L$, to represent the adjacency of nodes on directed graph G, where $a_{w,ij} = 1$ if node w is the tail of the directed link (i,j), i.e., $w = j$; $a_{w,ij} = -1$ if node w is the head of the directed link (i,j), i.e., $w = i$; $a_{w,ij} = 0$ otherwise. This work uses binary variable $p_{rt}^{k,ij}$ to express the route. If link (i,j) is a segment link between the kth node and $k+1$th node of request $r \in R$ at time slot $t \in T$, $p_{rt}^{k,ij} = 1$, and 0 otherwise. The model has the following constraints, $\forall k \in K_r \setminus \{|K_r|\}, r \in R, t \in T, w \in N$. According to (3.10), x_{tw}^{rk} and $x_{tw}^{r,k+1}$ cannot be 1 at the same time. (3.39) can be simplified to:

$$\sum_{(i,j)\in L} a_{w,ij} p_{rt}^{k,ij} = -x_{tw}^{rk} + x_{tw}^{r,k+1}, \forall k \in K_r \setminus \{|K_r|\}, r \in R, t \in T, w \in N. \tag{3.40}$$

The link capacity constraint is given by:

$$\sum_{r\in R} \sum_{k\in K_r} p_{rt}^{k,ij} d_r \le b_{ij}, \forall (i,j) \in L, t \in T. \tag{3.41}$$

Equation (3.41) ensures that each link's transmission resource is not overused.

This work takes the routes of all SFCs into consideration. The solution that minimizes the sum of lengths for all routes $\sum_{r\in R} \sum_{t\in T} \sum_{k\in K_r} \sum_{(i,j)\in L} l_{ij} p_{rt}^{k,ij}$ is chosen when there are multiple solutions that maximize $\lambda + \epsilon_1 \sum_{r\in R} \beta_r$. Therefore, the small number, ϵ_1, is multiplied to the second term of objective function to prioritize the first term over the second term; ϵ_2 is multiplied to the third term to prioritize the second term over the third term. To sum up, the model is:

$$\max \lambda + \epsilon_1 \sum_{r\in R} \beta_r - \epsilon_2 \sum_{r\in R} \sum_{t\in T} \sum_{k\in K_r} \sum_{(i,j)\in L} l_{ij} p_{rt}^{k,ij} \tag{3.42a}$$

$$\text{s.t. } (3.2) - (3.5), (3.7), (3.9) - (3.11), (3.13) - (3.37), (3.39) - (3.41)$$

$$\beta_r \in [1, |T|], \forall r \in R, \lambda \in [1, |T|] \tag{3.42b}$$

$$p_{rt}^{k,ij} \in \{0,1\}, \forall r \in R, t \in T, k \in K_r, (i,j) \in L \tag{3.42c}$$

$$z_i^{jr} \in \{0,1\}, \forall r \in R, i \in T, j \in T_i. \tag{3.42d}$$

3.2.3 NP-completeness

This work defines the decision version of the VNF allocation problem (VNFA) in the proposed model and prove that the VNFA problem is NP-complete [77, 78].

Definition 3.1 *Given a set of nodes N, a set of SFCs R and an availability schedule E, can an allocation of each SFC to make SSCAT be at least k be found?*

Theorem 3.1 *The VNFA problem is NP-complete.*

Proof: The VNFA problem is NP, as this work can verify whether SSCAT of the allocation is at least k in polynomial time $O(|T|\sum_{r\in R} K_r + |T||R|)$. $\sum_{r\in R} K_r$ is the sum of the lengths of all SFCs. It takes $O(|T|\sum_{r\in R} K_r)$ to check if the allocation of SFCs is feasible. It takes $O(|T||R|)$ to compute SCAT for each SFC. SSCAT is obtained by completing the process to compute SCAT.

This work presents that the partition problem, which is a known NP-complete problem [79], is polynomial time reducible to VNFA. The partition problem is defined as: whether a given multi-set S of positive integers can be partitioned into two subsets G_1 and G_2 such that the sum of the numbers in G_1 equals that in G_2.

First, an instance of VNFA from any instance of the partition problem is conducted, which consists of a set of positive integers $G = \{I_g : g \in [1, |G|]\}$. The set G is divided into two sets G_1 and G_2 so that $\sum_{I_g\in G_1} I_g = \sum_{I_g\in G_2} I_g$. An instance of VNFA is constructed with the following steps.

1. Consider a network with $\sum_{I_g\in G} I_g$ nodes. Each node is able to accommodate at most one function. G in the partition problem is the set of requests in VNFA.

2. I_g in the partition problem is the number of functions in request $g \in G$ in VNFA.

3. Consider the set of time slots T, where $|T| = 2k$.

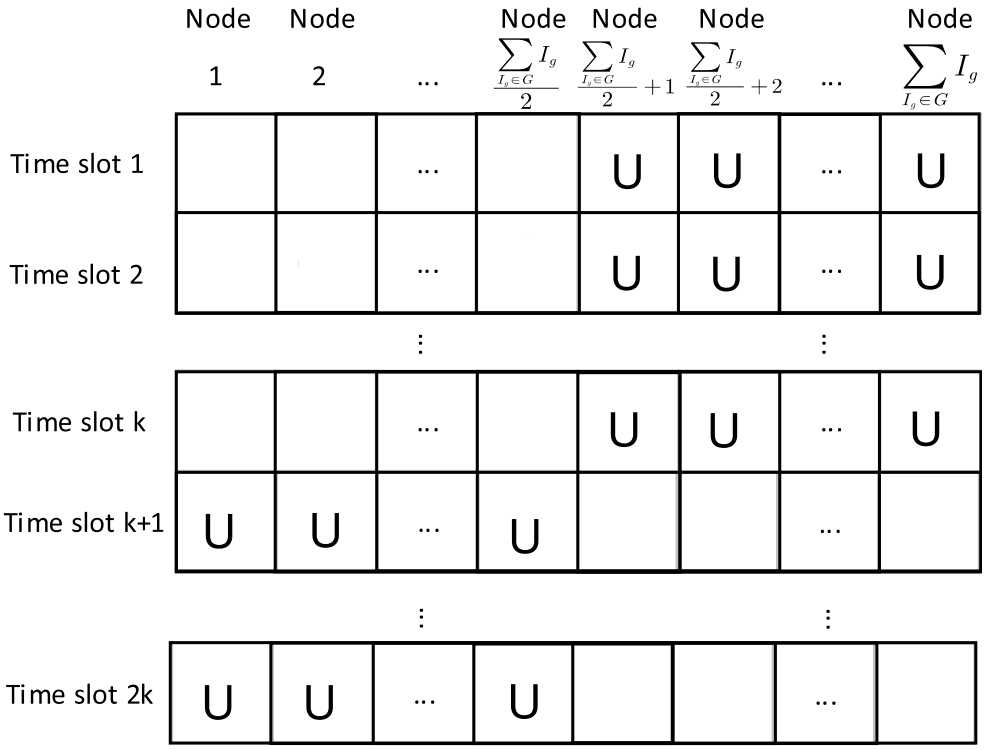


Figure 3.3: Construction demonstration. If a time slot is marked "U", it is unavailable, and otherwise available.

4. Consider the availability schedule, from time slot 1 to time slot k, node $i \in [1, \frac{1}{2}\sum_{I_g \in G} I_g]$ is available and node $i \in [\frac{1}{2}\sum_{I_g \in G} I_g + 1, \sum_{I_g \in G} I_g]$ is unavailable. From time slot $k+1$ to time slot $2k$, node $i \in [1, \frac{1}{2}\sum_{I_g \in G} I_g]$ is unavailable and node $i \in [\frac{1}{2}\sum_{I_g \in G} I_g + 1, \sum_{I_g \in G} I_g]$ is available.

Fig. 3.3 shows the construction. The presented steps have a polynomial complexity of $O(|G|)$, which transforms any instance of the partition problem into an instance of VNFA.

Consider that a partition problem instance is a Yes instance, which indicates that there exist two subsets of G_1 and G_2 with $\sum_{I_g \in G_1} I_g = \sum_{I_g \in G_2} I_g$. By using the above presented steps to define the corresponding VNFA instance from any partition problem instance, each function is assigned to an available node. The functions belonging to the requests corresponding to G_1 are allocated to the upper left part of the availability schedule from node 1 to node

$\frac{1}{2}\sum_{I_g \in G} I_g$. These functions are available during the first k time slots and unavailable during the next k time slots. The functions belonging to the requests corresponding to G_2 are allocated to the lower right part of the availability schedule from node $\frac{1}{2}\sum_{I_g \in G} I_g + 1$ to node $\sum_{I_g \in G} I_g$. These functions are unavailable at the first k time slots and available at the next k time slots. SSCAT of this network is k, which means that the VNFA instance is a Yes instance.

Conversely, if a VNFA instance is a Yes instance, then the corresponding partition problem instance is a Yes instance. When SSCAT of a VNFA instance is at least k, the smallest SCAT among the requests is k. To make SSCAT be k in the given network with the given availability schedule, the functions in the set of requests must be allocated to the available nodes at the first k time slots from node 1 to node $\frac{1}{2}\sum_{I_g \in G} I_g$, and the other functions must be allocated to the other nodes which are available at the next k time slots from node $\frac{1}{2}\sum_{I_g \in G} I_g + 1$ to node $\sum_{I_g \in G} I_g$. The functions allocated to node 1 to $\frac{1}{2}\sum_{I_g \in G} I_g$ belong to G_1 and the functions allocated to node $\frac{1}{2}\sum_{I_g \in G} I_g + 1$ to $\sum_{I_g \in G} I_g$ belong to G_2. So, if SSCAT of the network is at least k, i.e., the VNFA instance is a Yes instance, the corresponding partition problem instance is also a Yes instance, i.e., $\sum_{I_g \in G_1} I_g = \sum_{I_g \in G_2} I_g$.

This confirms that the partition problem, which is NP-complete, is polynomial time reducible to VNFA. Since VNFA belongs to NP, VNFA is NP-complete. ■

3.3 Heuristic algorithm

As the size of the ILP problem presented in Section 3.2 increases, the problem becomes more difficult to solve in practical time. A feasible solution may not be obtained within admissible computational time, which can be specified by SPs. This work introduces a heuristic algorithm for the proposed problem.

3.3.1 Selection of heuristic algorithm

As a heuristic algorithm, there are several candidates, which include a greedy algorithm, a simulated annealing (SA) algorithm [80], and a genetic algorithm (GA) [81].

A greedy algorithm is a simple heuristic algorithm which finds the best decision at each step aiming to find a near-optimal solution. Once the decision is made at each step, it is not changed at the later steps. The greedy algorithm traverses all the nodes for placing each function and find a suitable location, where the deployment does not violate any constraints. The greedy algorithm chooses the position for each function at each time slot, which has the maximum objective value at this step. Finally, the algorithm gets a final allocation for all functions at all time slots. However, the best decision in each step chosen in the greedy algorithm does not mean that the final result is the best solution. The worse choice which has a worse objective value in some steps may be chosen.

SA can avoid trapping in a local-optimal solution with a higher probability than the greedy algorithm. SA is a metaheuristic algorithm to obtain a near-optimal solution in a large search space for an optimization problem. It allows to accept a worse solution with a certain probability so that trapping into a local-optimal solution can be avoided. SA does not always choose the solution with a better objective value in each step so that a better final solution hidden behind sub-optimal solutions can be found.

GA is also a metaheuristic algorithm to obtain a near-optimal solution. Compared with SA, the solutions in GA are encoded in a special structure. GA generates new feasible solutions based on the structure with the help of existing solutions so that the new solutions can get close to the optimal solution. In the proposed model, each solution has four dimensions: requests, functions, time slots, and nodes. With GA, the solving procedure can be faster than SA because of the structure in solutions. The greedy algorithm can be used in generating a set of better initial solutions for GA compared with a set of random initial solution, which reduces the searching time for better solutions.

3.3.2 Framework

The framework of this heuristic algorithm is shown in Algorithm 3.1. A set of initial feasible solutions whose size is *IPN* is given by Algorithm 3.2 in line 2. In lines 3–5, if Algorithm 3.2 cannot provide a feasible solutions, the heuristic algorithm reports that no feasible solution is found. In lines 6–29,

the heuristic algorithm enters a loop. The loop has MG cycles. In each cycle, the heuristic algorithm explores new feasible solutions by performing internal crossover (see function *cross_in* in Algorithm 3.3), external crossover (see function *cross_out* in Algorithm 3.3), and mutation (see Algorithm 3.4) according to three probabilities ICP, ECP and MP, respectively. Newly generated solutions at lines 10, 15, and 20 are stored in the new feasible solution set S_{n}. They are added to the feasible solution set at a time in line 23. The heuristic algorithm calculates the fitness for each solution (see Algorithm 3.5). The genetic algorithm finds the solution with the highest fitness score and stores it. Finally, if the size of the feasible solution set exceeds $ULPN$, the heuristic algorithm chooses $ULPN$ feasible solutions as a new set of feasible solutions according to roulette gambler (see Algorithm 3.6).

Algorithm 3.1 Framework

Input: N, T, R, S, K_r, $c_{nt}^{s} \in C$, $e_{t}^{n} \in E$, $q_{s}^{rk} \in Q$, *IPN, MG, ECP, ICP, MP, ULPN*

Output: allocation for all functions

1: Define I as the feasible solution set
2: $I \leftarrow$ Generate set of initial feasible solutions by using function *init_chromos* in Algorithm 3.2
3: **if** $I = \emptyset$ **then**
4: **return** No feasible solution is found
5: **end if**
6: **for** $step = 1 \rightarrow MG$ **do**
7: Define I_{n} as the new feasible solution set
8: **for** each solution in I **do**
9: **if** a random number in $[0, 1] > 1 - ICP$ **then**
10: $I_{\mathrm{n}} \leftarrow$Generate a non-redundant and mutant solution by using function *cross_in* in Algorithm 3.3 whose inputs are the selected solution in I and random time slot t
11: **end if**
12: **end for**
13: **for** each solution in I except for the first one **do**
14: **if** a random number $[0, 1] > 1 - ECP$ **then**
15: $I_{\mathrm{n}} \leftarrow$Generate a non-redundant and mutant solution by using

function *cross_out* in Algorithm 3.3 whose inputs are the selected solution and its previous solution in I

16: **end if**
17: **end for**
18: **for** each solution in I **do**
19: **if** a random number $[0, 1] > 1 - MP$ **then**
20: $I_{\mathrm{n}} \leftarrow$Generate a non-redundant and mutant solution by using function *mutation* in Algorithm 3.4 whose input is the selected solution in I
21: **end if**
22: **end for**
23: Integrate I_{n} into I
24: Calculate the fitness score of the solutions in I by using function *calc_fin_ness* in Algorithm 3.5
25: Store the solution with the highest fitness score
26: **if** size of $I > ULPN$ **then**
27: Reduce the size of the set to $ULPN$ by using function *roulette_gambler* in Algorithm 3.6
28: **end if**
29: **end for**

3.3.3 Initial solution generation

The heuristic algorithm generates a set of initial feasible solutions by using Algorithm 3.2. Based on this set, more feasible solutions can be generated by Algorithms 3.3 and 3.4.

In the heuristic algorithm, each solution is a three-dimensional matrix. The first dimension represents time slots, the second one represents requests and the third one represents functions. The value of an element whose location is (t, r, k) is the allocation of the kth function of request r at time slot t, which belongs to N.

Algorithm 3.2 Initial solution

1: **function** INIT_CHROMOS(N, T, R, K_r, E, C, Q)
2: Set of initial solutions $I \leftarrow \emptyset$

3: Sort requests in R in a non-increasing order of K_r
4: **for** each time slot in T **do**
5: Sort nodes in N in a non-increasing order of time from time slot t to a time slot in which a node becomes unavailable
6: **for** $r = 1 \rightarrow |R|$ **do**
7: **for** $f = 1 \rightarrow K_r$ **do**
8: **for** $n = 1 \rightarrow |N|$ **do**
9: **if** used capacity of n is less than $c_{nt}^{s} - q_{s}^{rk}, \forall s \in S$ **then** AND any other functions in r were not allocated to n
10: Allocate the fth function in SFC r to n
11: Break
12: **else**
13: **if** $n = |N|$ **then**
14: **return** $s \leftarrow \emptyset$
15: **end if**
16: Continue
17: **end if**
18: **end for**
19: **end for**
20: **end for**
21: Store the allocation to I
22: **end for**
23: Duplicate a solution iteratively until the number of solutions in I becomes IPN
24: **return** I
25: **end function**

Algorithm 3.2 reorders set R in a non-increasing order of K_r (line 3). Then, it performs function allocation one by one (lines 4–22). At each time slot, the heuristic algorithm reorders set N according to the occurrence of unavailabilities from late to early (line 5). Then the genetic algorithm allocates the functions to nodes according to these new orders (lines 9–17). If the function cannot be assigned to a suitable node, no feasible solution is found in Algorithm 3.2. The judgment is performed in lines 13–15. Finally, the heuristic algorithm duplicates a solution iteratively until the number of solutions

becomes *IPN* (line 23).

3.3.4 New solution generation

There are three methods for generating new solutions in the heuristic algorithm.

The internal crossover, function *cross_in* in Algorithm 3.3, crosses adjacent time slots in the same solution. The aim of *cross_in* is to suppress the reallocations of VNFs between adjacent time slots.

The external crossover, function *cross_out* in Algorithm 3.3, crosses the same time slot between two solutions in the feasible solution set. A new solution is generated by modifying the VNF allocation in a randomly selected time slot of one solution based on that of another solution.

Algorithm 3.3 Crossover

1: **function** CROSS_IN($i \leftarrow$ the selected solution , $t \leftarrow$ the random time)
2: $i[t] \leftarrow i[t+1]$
3: **return** i
4: **end function**
5: **function** CROSS_OUT(i_1, i_2)
6: $location \leftarrow$ a random integer in $[1, |T|]$
7: $i_c \leftarrow i_1$
8: $i_c[location] \leftarrow i_2[location]$
9: **return** i_c
10: **end function**

The other function for generating new solutions is function *mutation* in Algorithm 3.4. *init_chromos* function in Algorithm 3.2 generates the initial solution set based on the length of each SFC. On the other hand, *mutation* function generates a new solution based on the SCAT of each SFC.

Algorithm 3.4 Mutation

1: **function** MUTATION($i \leftarrow$ the selected solution, E)
2: Calculate the SCAT for all SFCs in solution i
3: Sort requests in R in a non-decreasing order of SCAT
4: Sort nodes in N in a non-increasing order of time from the first time slot to a time slot where a node becomes unavailable

5: **for** $r \in R$ **do**
6: **for** $f = 1 \rightarrow K_r$ of request r **do**
7: **for** $n \in N$ **do**
8: **if** used capacity of n is less than $c_{nt}^{s} - q_{s}^{rk}, \forall s \in S$ **then** AND any other functions in r were not allocated to n
9: Allocate the fth function in SFC r to n
10: Break
11: **else**
12: Continue
13: **end if**
14: **end for**
15: **end for**
16: **end for**
17: **return** new solution
18: **end function**

3.3.5 Calculation of fitness

The algorithm computes the fitness score for each solution by using (3.8).

Algorithm 3.5 Fitness calculation

1: **function** CALC_FIN_NESS($i \leftarrow$ the selected solution, E)
2: Calculate the SCAT for all SFCs in solution i
3: **return** min($SCAT$)+sum($SCAT$)/($|T| \times |R|$)
4: **end function**

3.3.6 Choice of solutions

The heuristic algorithm uses roulette wheel selection to create a new feasible solution set by choosing $ULPN$ solutions from the feasible solution set.

In the *roulette_gambler* and *choice* functions in Algorithm 3.6, input $chroms$ is the set of solutions and fit_pros is the set of the fitness scores of the corresponding solutions in $chroms$.

Algorithm 3.6 Choice

1: **function** ROULETTE_GAMBLER($fit\_pros, chroms$)
2: $pick \leftarrow$ a random number in $[0, 1]$

```
    for j = 1 → |chroms| do
        pick ← pick − fit_pros[j]/sum(fit_pros)
        if pick ≤ 0 then
            return j
        end if
    end for
    return |chroms| − 1
end function
function CHOICE(chroms, fit_pros)
    choice_gens ← ∅
    for i = 1 → min(|chroms|, ULPN) do
        j ← ROULETTE_GAMBLER(fit_pros, chroms)
        append chroms[j] to choice_gens
    end for
    return choice_gens
end function
```

3.4 Evaluations

3.4.1 Comparison with other models

This work compares the proposed model with three other models. The first model, which is called the persistence allocation model, does not consider the service interruptions caused by node unavailability. The model maximizes SSCAT by suppressing the interruptions caused by VNF reallocation. It determines a node to which each VNF is allocated randomly and keeps this allocation from the first time slot to the last one. To some extent, such a model has no ability to avoid unavailable nodes. The second model, called the single-slot allocation model, considers the service interruptions caused by node unavailability at each time slot, regardless of VNF reallocation between all adjacent time slots. The model minimizes the number of VNFs allocated to unavailable nodes at each time slot. This model independently determines VNF allocation of each time slot and tries to avoid allocating VNFs to unavailable nodes according to the availability schedule. However, this model does not consider the relationship between VNF allocations at adjacent time

slots; different allocations at adjacent time slots cause service interruptions. For each $t \in T$, the optimization problem of single-slot allocation model is formulated as an ILP problem by:

$$\min \sum_{r \in R} \sum_{k \in K_r} \sum_{n \in N} x_{tn}^{rk} e_t^n \tag{3.43a}$$

$$\text{s.t. } (3.9) - (3.11). \tag{3.43b}$$

The third model, the double-slot allocation model, is improved variant of the single-slot allocation model. This model computes the VNF allocation at time slot t by considering that in the last time slot, $x_{t-1,n}^{rk}, \forall n \in N, r \in R, k \in K_r$. The solution that minimizes the differences between time slots t and $t-1$ is chosen when there are multiple solutions that minimize $\sum_{r \in R} \sum_{k \in K_r} \sum_{n \in N} x_{tn}^{rk} e_t^n$. For each $t \in T$, the optimization problem of double-slot allocation model is formulated by:

$$\min \sum_{r \in R} \sum_{k \in K_r} \sum_{n \in N} (x_{tn}^{rk} e_t^n + \epsilon_3 x_{tn}^{rk} \odot x_{t-1,n}^{rk}) \tag{3.44a}$$

$$\text{s.t. } (3.9) - (3.11). \tag{3.44b}$$

The small number, ϵ_3, is multiplied to the second term to prioritize the first term over the second term. ϵ_3 is given by $\frac{1}{|N||R| \max_{r \in R}\{K_r\}}$. This work compares the SSCAT values yielded by the above three models with those of the proposed model.

The ILP problems of the proposed model, the single-slot allocation model, and the double-slot allocation model are solved by the IBM(R) ILOG(R) CPLEX(R) Interactive Optimizer, version 12.7.1 [82], running on an Intel Core i7-7700 3.60 GHz 4-core CPU, 32 GB memory. The persistence allocation model is implemented by Python 3.7 and run on the same hardware.

Five cases are examined. The conditions of these cases are shown in Table 3.1. All cases focus on VNF allocation regardless of route computation. All resources requirements of functions are one. The capacities of all nodes in the network are two for all types of resources. For the other conditions shown in the Table 3.1, *Nodes* means the number of nodes; *Requests* means the number of functions in each request; *Functions* means the number of functions in each request; *Time slots* means the number of time slots; *Maximum Unavailabilities* means the maximum number of unavailable nodes at a time slot. The

Table 3.1: Evaluation conditions.

Case	Nodes	Capacity	Request	Functions	Time slots	Maximum unavailabilities
1	8	2	4	3, 2, 2, 4	6	2
2	8	2	4	3, 2, 2, 4	6	6
3	16	2	8	6, 3, 2, 2, 4, 4, 3, 2	6	2
4	16	2	8	6, 3, 2, 2, 4, 4, 3, 2	6	12
5	5	2	3	2, 2, 3	24	1

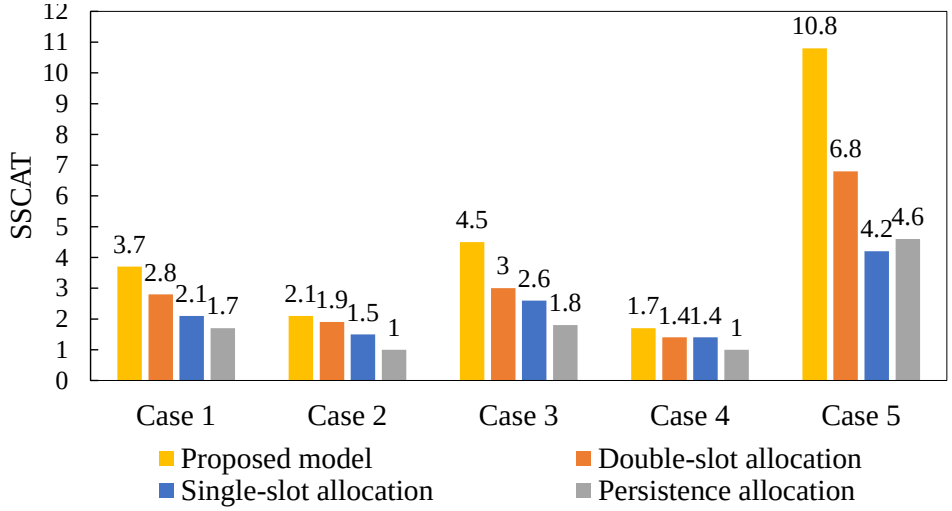


Figure 3.4: SSCAT values yielded by four models.

evaluation obtains the VNF allocation of the proposed model by solving the ILP problem in Section 3.2 without flow or link capacity constraints. Cases 1, 2, 3 and 4 randomly generate ten different availability schedules. Case 5 randomly generates five different availability schedules. Each case computes the VNF allocations in the same network. This yields the computed SSCAT values which are then averaged.

Fig. 3.4 shows the SSCAT values yielded by the four models. We can observe that the persistence allocation model performs better than single-slot allocation only in case 5. Persistence allocation is sensitive to failures. Single-slot allocation is sensitive to the changes in allocations between adjacent time slots. Since case 5 only has at most one failure at each time slot, the persistence allocation perform slightly better than the single-slot allocation because of the

Table 3.2: γ_{p}, γ_{s}, and γ_{d} for Table 3.1.

Case	γ_{p}	γ_{s}	γ_{d}
Case 1	2.18	1.76	1.32
Case 2	2.10	1.40	1.10
Case 3	2.50	1.73	1.50
Case 4	1.70	1.21	1.21
Case 5	2.34	2.57	1.59

fewer failures and the changing positions of the failures in different time slots. Table 3.2 shows γ_{p}, γ_{s}, and γ_{d}, which denote the ratios of SSCAT obtained by the proposed model to that of the persistent, single-slot, and double-slot allocations, respectively. The proposed model improves SSCAT compared with the other models in all cases examined. From the comparison of the results between cases 1 and 3 or cases 2 and 4, the proposed model yields larger SSCAT values when the number of unavailable nodes in a time slot is small.

3.4.2 Demonstration using different availability schedules

The distribution of unavailabilities in the availability schedules impacts on the SSCAT values obtained by the proposed model. To demonstrate this impact, this work creates five different availability schedules and calculates the SSCAT values with these availability schedules under the same condition. The availability schedules used in this demonstration are shown in Fig. 8. From type 1 to 5, the positions of unavailabilities become more random and scattered. In this demonstration, an eight-node network is considered, where the capacity of each node is set to two. The evaluation considers four requests with lengths of three, two, two, and four. Six time slots are considered in this demonstration.

Table 3.4 shows the SSCAT values for the five availability schedules shown in Table 3.3. The regular and compact error patterns, such as type 1, provide higher SSCAT values; random and sparse error patterns, such as type 5, provide lower SSCAT values.

Table 3.3: Five availability schedules. If "U" is marked in a time slot, it is unavailable, and otherwise available.

(a) Availability schedule 1

Time slot \ Node	1	2	3	4	5	6	7	8
1							U	U
2							U	U
3							U	U
4							U	U
5							U	U
6							U	U

(b) Availability schedule 2

Time slot \ Node	1	2	3	4	5	6	7	8
1							U	U
2							U	U
3							U	U
4	U	U						
5	U	U						
6	U	U						

(c) Availability schedule 3

Time slot \ Node	1	2	3	4	5	6	7	8
1							U	U
2							U	U
3	U	U						
4	U	U						
5							U	U
6							U	U

(d) Availability schedule 4

Time slot \ Node	1	2	3	4	5	6	7	8
1	U	U						
2		U	U					
3			U	U				
4				U	U			
5					U	U		
6						U	U	

(e) Availability schedule 5

Time slot \ Node	1	2	3	4	5	6	7	8
1		U			U			
2	U		U					
3	U							U
4						U	U	
5			U		U			
6				U			U	

3.4.3 Effect of determining routes and VNF allocation simultaneously

This work evaluates the effect of determining the SFC routes at the same time as the VNF allocation in the proposed model. Two computation methods are examined in this evaluation. The computation method that computes the VNF allocation and routing by solving the ILP problem in 3.2.2 is called method 1. On the other hand, the computation method that computes allocation by the model in Section 3.2.1 and then determines the route between each adjacent VNF pair based on the shortest path is called method 2.

The evaluation randomly generates five different availability schedules and compute the allocations with these different availability schedules by using

Table 3.4: Results of the demonstrations for different availability schedules in Fig. 3.5.

Type	SSCAT	SCAT of Chain 1	SCAT of Chain 2	SCAT of Chain 3	SCAT of Chain 4
1	6	6	6	6	6
2	3	6	3	3	6
3	3	3	6	6	3
4	3	4	4	4	3
5	2	3	3	4	2

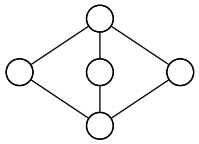


Figure 3.5: Network in case 5.

methods 1 and 2. This work compares the two methods in terms of the lengths of obtained paths and the computation time. If a function in a request is allocated to an unavailable node, the path length of request is calculated under the assumption that the SFC of this request is active. SFC routes are determined by computing the shortest paths between each adjacent VNF pair. This work uses case 5 shown in Table 3.1 in this evaluation. The network in case 5 is shown in Fig. 3.5. The transmission resource of each link is set to one, and the transmission resource demand of each request is set to 0.1. The length of each link is set to 1 unit. Fig. 3.6 shows the sum of path lengths of all three requests with methods 1 and 2. The path length of a request is the sum of the lengths of the links in its route. Note that the SSCAT and SCAT values computed by the two methods are the same. Method 1 offers, on average, 14.71% shorter path length than compared with method 2. Fig. 3.7 shows that the computation time of method 1 is, on average, 4.53 times greater than that of method 2. The proposed model, which determines routing, can provide smaller path lengths than the model that selects the shortest path in all cases examined at the cost of an increase in computation time.

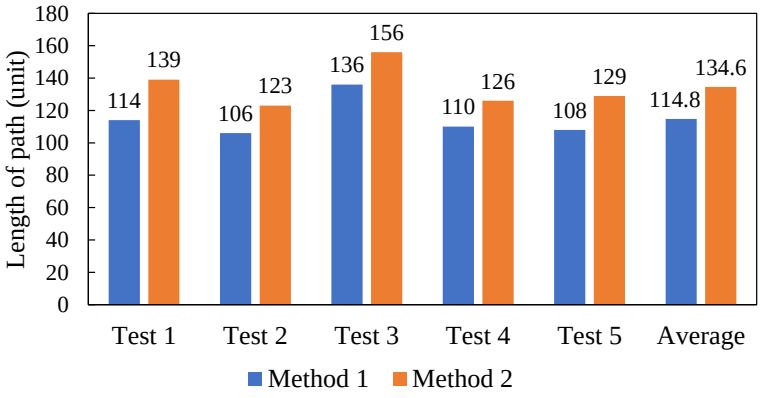


Figure 3.6: Comparison of path lengths in case 5.

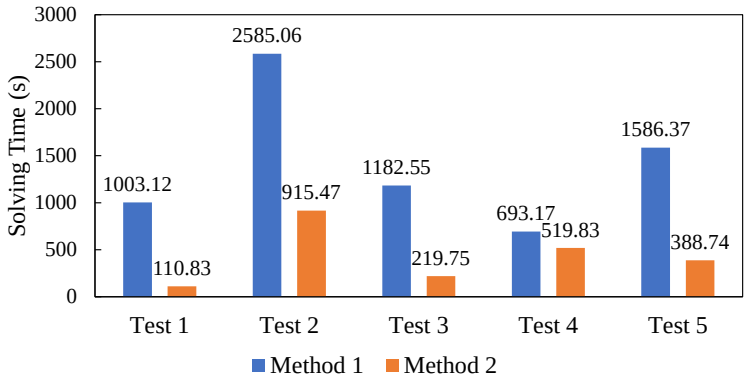


Figure 3.7: Comparison of solving time in case 5.

3.4.4 Effect of heuristic algorithm

This work evaluates the performance of the heuristic algorithm in terms of the objective value in (3.8), SSCAT, and the computation time. This evaluation uses cases 1, 3, and 5 in Table 3.1 in this evaluation. Each case randomly generates ten different availability schedules for cases 1 and 3 and five different availability schedules for case 5. The evaluation computes the allocations with these different availability schedules by using the heuristic algorithm and the ILP approach. Table 3.5 shows the parameter setting used in this evaluation. The heuristic algorithm is implemented by C++ 15, compiled by Microsoft Visual C++ 2017 v15.9.16, using Intel Core i7-7700 3.60 GHz 4-core CPU, 32 GB memory.

Table 3.5: Parameter setting.

Parameter	Setting
MG	100
IPN	12
$ULPN$	100
ICP	0.8
ECP	0.6
MP	0.8

Table 3.6 compares the results of each test obtained by using the heuristic algorithm with those obtained by the ILP approach. In most randomly generated availability schedules, the heuristic algorithm obtains the same SSCAT values as the ILP approach. In some test results, the heuristic algorithm yields a smaller sum of the SCAT values of each request, $\sum_{r \in R} \beta_r$. The heuristic algorithm reduces the calculation time. However, it does not perform well in a small number of tests, such as test 6 in case 3 and test 2 in case 5. As the size of network gets larger or the number of unavailabilities gets larger, the difference becomes large.

Table 3.7 shows the average computation time for the ILP approach and the heuristic algorithm for each case. The heuristic algorithm reduces the computation time by 97.71% compared with the ILP approach on average; the larger the problem size is, the more the computation time of heuristic algorithm is reduced below that of the ILP approach. Fig. 3.8 shows the differences in objective and SSCAT values between the heuristic algorithm and the ILP approach. The difference in objective value between the results calculated by the ILP approach and the heuristic algorithm is 1.59% on average. Fig. 3.9 shows how the value of the objective in (3.8) changes with successive generations for test 2 case 1. Heuristic algorithm performance increases with each generation, but more generations take longer time. The trade-off between the solution time and result quality when setting the parameters of the heuristic algorithm need to be considered.

Table 3.6: Test result.

Case	Test	ILP				Heuristic algorithm				Obj. value difference	SSCAT difference	$\sum_{r\in R}\beta_r$ difference
		SSCAT (λ)	$\sum_{r\in R}\beta_r$	Obj. value	Time (s)	SSCAT (λ)	$\sum_{r\in R}\beta_r$	Obj. value	Time (s)			
Case 1	1	3	20	3.83	22.66	3	18	3.75	0.76	2.09%	0%	10%
	2	4	19	4.79	16.77	4	18	4.75	0.76	0.84%	0%	5.3%
	3	3	18	3.75	21.22	3	15	3.63	0.74	3.33%	0%	16.7%
	4	4	20	4.83	14.57	4	18	4.75	0.76	1.66%	0%	10%
	5	4	21	4.88	12.23	4	21	4.88	0.74	0.00%	0%	0%
	6	5	23	5.96	2.48	5	23	5.96	0.80	0.00%	0%	0%
	7	5	23	5.96	3.05	5	23	5.96	0.88	0.00%	0%	0%
	8	3	15	3.63	17.31	3	12	3.5	0.81	3.45%	0%	20%
	9	3	21	3.88	9.06	3	21	3.88	0.85	0.00%	0%	0%
	10	3	19	3.79	34.04	3	18	3.75	0.87	1.06%	0%	5.3%
Case 3	1	6	48	7	7.64	6	48	7	1.37	0.00%	0%	0%
	2	4	44	4.92	1034.54	4	44	4.92	1.56	0.00%	0%	0%
	3	5	44	5.92	613.94	5	44	5.92	1.63	0.00%	0%	0%
	4	5	47	5.98	195.73	5	46	5.96	1.81	0.35%	0%	2.1%
	5	4	44	4.92	619.26	4	44	4.92	1.62	0.00%	0%	0%
	6	5	44	5.92	787.32	4	42	4.88	1.02	17.61%	20%	-
	7	3	41	3.85	5947.47	3	38	3.79	1.90	1.62%	0%	7.3%
	8	3	42	3.88	9292.16	3	42	3.88	1.63	0.00%	0%	0%
	9	5	46	5.96	592.74	5	46	5.96	1.03	0.00%	0%	0%
	10	5	46	5.96	299.59	5	44	5.92	1.05	0.71%	0%	4.3%
Case 5	1	11	42	11.58	194.86	11	42	11.58	1.83	0.00%	0%	0%
	2	14	48	14.67	219.73	13	47	13.65	1.23	6.95%	7.14%	-
	3	10	36	10.50	78.50	10	35	10.49	1.20	0.01%	0%	2.78%
	4	18	58	18.81	224.30	18	56	18.78	1.34	0.16%	0%	3.45%
	5	11	38	11.53	68.60	11	38	11.53	1.28	0.00%	0%	0%

Table 3.7: Comparison of average computation time for case 5.

Case	ILP (s)	Heuristic Algorithm (s)	Reduction (%)
Case 1	15.36	0.8	94.79%
Case 3	1939.04	1.46	99.92%
Case 5	156.2	1.38	99.12%

3.5 Discussions

3.5.1 Dealing with new requests at new time slots

If allocations at new time slots for new requests are needed, the proposed model is applied with the following small changes. Assume that a new set of

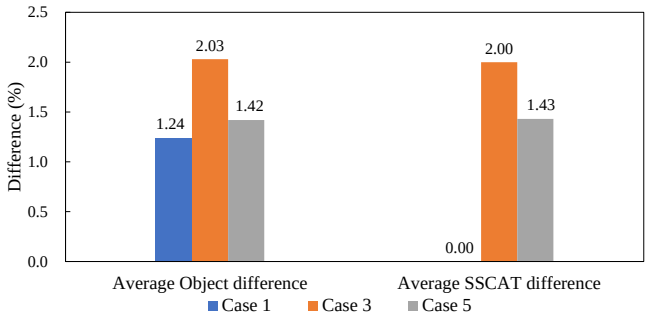


Figure 3.8: Comparison of results for case 5.

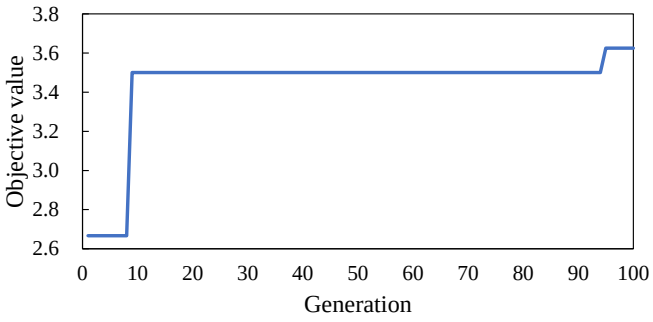


Figure 3.9: Changes in objective value with each generation.

time slots T^* and requests R^* at time slot $t_n \in T$ is given, the allocations at time slot t_n are known. In order not to affect the SSCAT values of R^*, this work considers a set of virtual nodes $N^*, |N^*| = \max_{r \in R^*} K_r$ whose capacities are unlimited. These nodes are available from time slot t_n to time slot $|T|$ and unavailable in T^*.

For a new set of time slots $T^{**} = \{[t_n, |T|], T^*\}$ and a new set of requests $R^{**} = \{R, R^*\}$, this work allocates all of the functions in R^* to N^* at time slot 1 in T^{**} and gets the values of $x_{1,n}^{rk}, \forall n \in N^*, r \in R^*, k \in K_r$. The allocations for R^* in T^* can be acquired by using the proposed model with the known allocations at the first time slot in T^*.

3.5.2 Applicability of proposed model in real systems

The proposed model is applied for the initial allocation of VNFs in SFCs before the running of services. The proposed model is suitable for deterministic

availability schedule obtained in advance. The requests in the proposed model are static and given in advance. The users of the proposed model emphasize the optimization of the allocations. Failures frequently appear or the maintenances are frequently performed.

In real systems, SPs need to store the information of availability schedules, VNFs, services, and network devices in the database. The calculation of the VNF allocation by using the proposed model is performed in a computation node in the network with the necessary information from the database. The computation node can be an independent node whose role is only the allocation calculation or a node in N with a special function for allocation calculation. Then the model calculates the allocation of VNFs. The allocation result is sent to each corresponding node. The nodes run the corresponding functions or the containers in the nodes by pulling the corresponding images from repositories.

3.6 Summary

This chapter proposed a VNF allocation model for improving the continuous available time of service function chains assuming the existence of known availability schedules. This work formulated the proposed model as an ILP problem that maximizes the minimum number of the longest continuous available time slots in each SFC. This work proved that the decision version of the VNF allocation problem (VNFA) in the proposed model is NP-complete. Numerical results showed that the proposed model improves the continuous available time of SFCs, compared with the persistent allocation model, the single-slot allocation model, and the double-slot allocation model. This work observed that the proposed model reduces the path lengths of SFCs as it computes VNF allocation and SFC routes at the same time. This work developed a heuristic algorithm to yield practical solution time. In the cases examined, the developed heuristic algorithm reduces the average computation time with some penalty in performance compared with the ILP approach.

Chapter 4

Primary and backup resource allocation model for deterministic availability schedules

This chapter proposes a primary and backup VNFs allocation to maximize the continuous available time of SFCs in a network against service interruptions [83, 84].

In this chapter, a kind of hot backup is considered. A backup function needs to be prepared for a period before it is implemented. The period is called *recovery time*. If backup functions are placed and are activated within suitable time slots for preparation instead of all the time slots from the beginning, the interruptions can be suppressed. The extra resource consumption for backup functions can be reduced since the users do not need to backup functions from the beginning time slot.

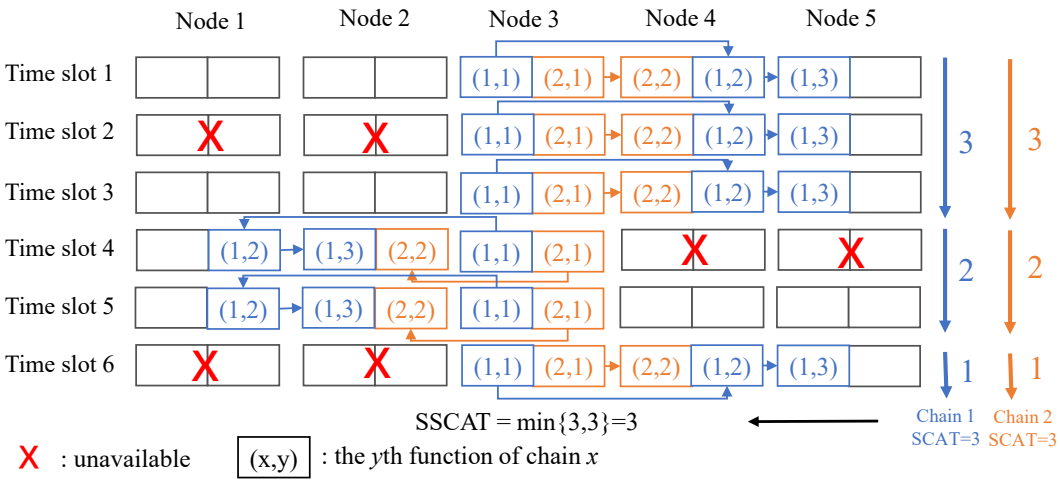
Figure 4.1 shows the examples for calculating SSCAT and SCATs and the differences between the allocations with and without considering backups under a given availability schedule. The example places two function chains 1 and 2 with two and three VNFs to a set of five nodes among six time slots. Nodes 1 and 2 are not available at time slots 2 and 6, and nodes 4 and 5 are not available at time slot 4. (x, y) denotes the yth function of chain x.

If the example places the functions without considering backup as shown in Fig. 4.1(a), chains 1 and 2 are interrupted between time slots 3 and 4 caused by the sudden replacements of functions (1, 2) and (1, 3), and (2, 2), respectively, and interrupted between time slots 5 and 6 caused by the sudden replacements of functions (1, 2) and (1, 3), and (2, 2), respectively. SCAT of chain 1 is the largest continuous available time, i.e., the largest one among three, two, and one, i.e., three. SCAT of chain 2 is the largest continuous available time, i.e., the largest one among three, two, and one, i.e., three. SSCAT is the smallest value among SCATs of all SFCs, i.e., three. The example assumes that the recovery times of all functions are one. Functions (1,2), (1,3) and (2,2) at time slots 3 and 5 are backed up to prevent the interruptions caused by function replacement at time slots 4 and 6, as shown in Fig. 4.1(b). As a result, the backup can increase SCATs of all the chains from three to six and SSCAT from three to six.

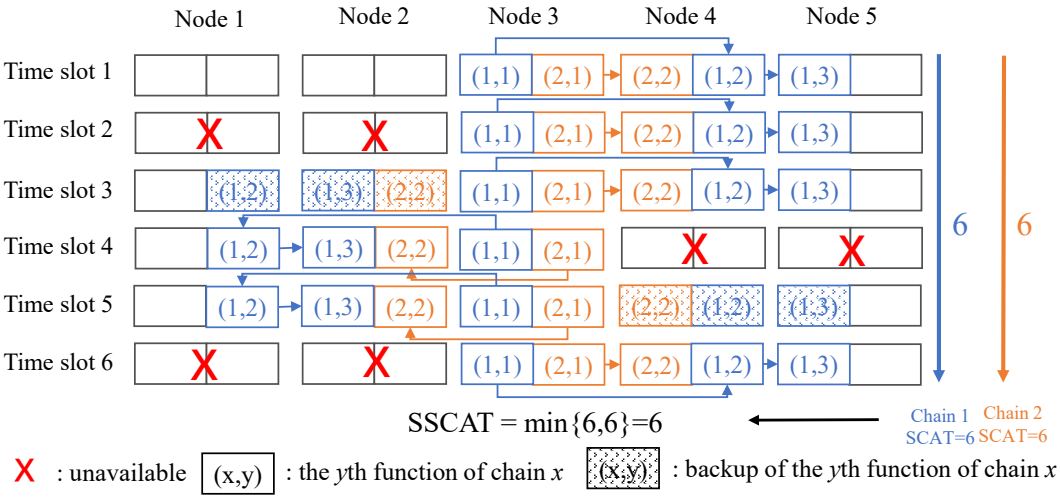
The remainder of the chapter is organized as follows. Section 4.1 introduces the motivation and several applicabilities of the proposed model. Section 4.2 presents the proposed model and gives the upper bound of SSACT provided by the proposed model. Section 4.3 gives a heuristic algorithm for solving the considered problem. Section 4.5 discusses some extensions of the proposed model in different scenarios. Section 4.4 compares the performance of the proposed model with those of baseline models in different cases. Section 4.6 summarizes the chapter.

4.1 Motivation and applicability

SPs perform maintenance and system updates regularly. Specific VMs waiting for maintenance and updating are marked as being unavailable at a specific time by SPs. SPs can reduce the number of service interruptions by knowing where and when service interruptions occur and controlling the positions and time of service interruptions referring to the maintenance schedule known forehead. In terms of the unscheduled and sudden failures, users can rely on protection mechanisms [85–87] in the unavailable periods of nodes. This work does not concern the details of the mechanisms of providing availability schedules. This work assumes that the availability schedules are estimated and



(a) Placement without backup.



(b) Placement with backup. The recovery time of each VNF is one time slot.

Figure 4.1: Examples of service continuous available time of a chain by comparing the allocations of all VNFs in a chain in different time slots with and without backups.

provided by administrators. If a node is marked as available, it is considered to be available. The availability is ensured by the other protection systems and techniques, e.g., duplication.

If the exact period or the starting time that a physical machine becomes unavailable is not considered, the availability and reliability are often evaluated

by other metrics including probability, mean time to repair, and mean time between failures in existing studies [61, 88]. The above evaluation is widely used since the exact time of the unavailability is not usually obtainable. Once the exact period of unavailability from the maintenance schedules is obtained, the available period of a service which is evaluated by time, i.e., SSCAT in this work, is more intuitionistic and applicable compared with the mean times or probability. This work is based on the motivation that the available and unavailable time of a physical node is given and the normal running time of services is the metric to measure. The schedule is assumed to be obtained in advance.

The models introduced in Chapter 3 provided VNF allocation models to suppress the interruptions from the allocations on unavailable nodes and the reallocations. However, it is a passive way to avoid assigning to unavailable nodes and has a limitation decided by the availability schedules. The proposed model actively uses backups to avoid the interruptions caused by the reallocation for improving the SSCAT compared with the model without considering backup VNFs.

4.2 Model design

4.2.1 Problem formulation

In addition to the notions in Section 3.2, this work introduces a new parameter. $g_r^k \in T, r \in R, k \in K_r$ is a given parameter which represents the recovery time of the kth function of request r. This work assumes that $0 < g_r^k < |T|$. If the $k \in K_r$th function of request $r \in R$ is backed up on node $n \in N$ at time slot $t \in T$, $u_{tn}^{rk} = 1$; and otherwise 0. The backed up functions also consume computing resources.

If n can be continuously occupied for backup from time slot $t - g_r^k$ to time slot $t - 1$, that is, $u_{t-g_r^k,n}^{rk} = u_{t-g_r^k+1,n}^{rk} = \cdots = u_{t-1,n}^{rk} = 1$, and this function is allocated to n at time slot t, that is, $x_{tn}^{rk} = 1$, this function can be prevented from the interruption caused by unavailability $e_t^{n'}$. If the $k \in K_r$th function of request $r \in R$ is prevented from the interruption caused by unavailability $e_t^{n'}$,

$\alpha_{tn}^{rk} = 1$; and otherwise 0. $\forall r \in R, k \in K_r, n \in N$, α_{tn}^{rk} is expressed by:

$$\alpha_{tn}^{rk} = \begin{cases} ((\vee_{n' \in N \setminus \{n\}} e_t^{n'} x_{t-1,n'}^{rk}) \wedge ((\prod_{t' \in [t-g_r^k, t-1]} u_{t'n}^{rk}) \wedge x_{tn}^{rk})) \vee ((e_t^n \\ x_{t-1,n}^{rk}) \wedge (\vee_{n' \in N \setminus \{n\}} (\prod_{t' \in [t-g_r^k, t-1]} u_{t'n'}^{rk}) \wedge x_{tn'}^{rk}), t \in T \setminus \left[1, g_r^k\right] \\ 0, t \in \left[2, g_r^k\right], \end{cases} \tag{4.1}$$

in which $\vee_{n' \in N \setminus \{n\}} e_t^{n'} \cdot x_{t-1,n'}^{rk} = 1$ means that there is an unavailability on node n' at time t, i.e. $e_t^{n'}$, interrupts the allocation of the kth function of request r on node n' between time slot t and $t-1$; and 0 otherwise. To ensure the backup of this interrupted function work, the maintenance of this backup needs to be kept from $t - g_r^k$ to $t - 1$, i.e., $\prod_{t' \in [t-g_r^k, t-1]} u_{t'n}^{rk} = 1$. Finally, the backup is activated, i.e., $x_{tn}^{rk} = 1$. The function cannot be backed up on an unavailable node, which is expressed by:

$$e_t^n \cdot u_{tn}^{rk} = 0, \forall r \in R, k \in K_r, n \in N, t \in T. \tag{4.2}$$

o_t^r is redefined in this chapter. If the allocation of request r is changed between time slot $t - 1$ and time slot t or any VNF of request r at time slot t or $t - 1$ is allocated to an unavailable node and the function is not backed up, o_t^r is set to 0, and otherwise 1. When $t = 1, o_t^r = 0$; otherwise, o_t^r can be expressed by:

$$o_t^r = \prod_{k \in K_r} \prod_{n \in N} ((x_{tn}^{rk} \odot x_{t-1,n}^{rk} \vee \alpha_{tn}^{rk}) \wedge (1 - x_{tn}^{rk} e_t^n) \wedge (1 - x_{t-1,n}^{rk} e_{t-1}^n)), \\ \forall r \in R, t \in T \setminus \{1\}. \tag{4.3}$$

If the computational resources of backup functions are limited to a constant value R_{B}, u_{tn}^{rk} is constrained by:

$$\sum_{r \in R} \sum_{k \in K_r} \sum_{t \in T} \sum_{n \in N} u_{tn}^{rk} \leq R_{\mathrm{B}}. \tag{4.4}$$

R_{B} has a range of 0 to $\sum_{r \in R} \sum_{k \in K_r} g_r^k$.

The node capacity constraint is given by:

$$\sum_{r \in R} \sum_{k \in K_r} (x_{tn}^{rk} + u_{tn}^{rk}) \cdot q_s^{rk} \leq c_{nt}^s, \forall n \in N, t \in T, s \in S. \tag{4.5}$$

Each node, because of the computational resource limitation, can carry only a limited number of functions. Equation (4.5) ensures that each node's computational resources must not exceed its capacity during allocation.

The assignment constraint is given by:

$$\sum_{k \in K_r} (u_{tn}^{rk} + x_{tn}^{rk}) \leq 1, \forall r \in R, n \in N, t \in T, \tag{4.6}$$

$$\sum_{n \in N} x_{tn}^{rk} = 1, \ \forall r \in R, k \in K_r, t \in T. \tag{4.7}$$

$$\sum_{n \in N} u_{tn}^{rk} \leq 1, \ \forall r \in R, k \in K_r, t \in T. \tag{4.8}$$

Equation (4.6) assumes that one service chain does not allocate multiple VNFs in this chain on one VM considering the influence of the reallocation of VMs [44] and that the primary VNF and the backup VNF of the same function cannot be allocated to the same nodes at one time slot. Equation (4.7) ensures that all VNFs are placed once in the network. If any requested SFCs cannot be accepted, there is no feasible solution. To discuss the case that too many SFC are requested to accept, this work relaxes the constraint in (4.7) and give an extension of the proposed model to maximize the number of accepted SFCs as the objective in Section 4.5.3. Equation (4.8) assumes that each VNF has at most one backup at one time slot. Equations (4.7) and (4.8) impose that the proposed model only uses at most two replicas, one for primary VNF and one for backup VNF. If the users of the proposed model want to prevent service interruptions from the failures of functions and load balancing by using VNF replicas in an SFC, a possible extension is introduced in Section 4.5.2 to support multiple replicas for primary and backup VNFs.

According to the linearization process in Appendix A, (4.1) is linearized to (4.9)-(4.27) and (4.3) is linearized to (5.5a)-(5.5d), (3.26)-(5.5w), and (4.28)-(4.33) with some auxiliary variables as follows:

$$\gamma_{tn}^{rk} \leq \sum_{n' \in N \setminus \{n\}} e_t^{n'} x_{t-1,n'}^{rk}, \forall r \in R, k \in K_r, n \in N, t \in T \setminus \left[1, g_r^k\right], \tag{4.9}$$

$$\gamma_{tn}^{rk} \geq \frac{1}{|N| - 1} \sum_{n' \in N \setminus \{n\}} e_t^{n'} x_{t-1,n'}^{rk}, \forall r \in R, k \in K_r, n \in N, t \in T \setminus \left[1, g_r^k\right], \tag{4.10}$$

$$\eta_{tn}^{rk} \leq u_{t'n}^{rk}, \forall r \in R, k \in K_r, n \in N, t \in T \setminus \left[1, g_r^k\right], t' \in T \setminus \left[t - g_r^k, t-1\right], \tag{4.11}$$

$$\eta_{tn}^{rk} \geq \sum_{t' \in T \setminus \left[t-g_r^k, t-1\right]} u_{t'n}^{rk} - (|T| - g_r^k) + 1, \forall r \in R, k \in K_r, n \in N, \\ t \in T \setminus \left[1, g_r^k\right], \tag{4.12}$$

$$\mu_{tn}^{rk} \leq \eta_{tn}^{rk}, \forall r \in R, k \in K_r, t \in T \setminus \left[1, g_r^k\right], n \in N, \tag{4.13}$$

$$\mu_{tn}^{rk} \leq x_{tn}^{rk}, \forall r \in R, k \in K_r, t \in T \setminus \left[1, g_r^k\right], n \in N, \tag{4.14}$$

$$\mu_{tn}^{rk} \geq \eta_{tn}^{rk} + x_{tn}^{rk} - 1, \forall r \in R, k \in K_r, t \in T \setminus \left[1, g_r^k\right], n \in N, \tag{4.15}$$

$$\tau_{tn}^{rk} \leq \sum_{n' \in N \setminus \{n\}} \mu_{tn'}^{rk}, \forall r \in R, k \in K_r, t \in T \setminus \left[1, g_r^k\right], n \in N, \tag{4.16}$$

$$\tau_{tn}^{rk} \geq \frac{1}{|N| - 1} \sum_{n' \in N \setminus \{n\}} \mu_{tn'}^{rk}, \forall r \in R, k \in K_r, n \in N, t \in T \setminus \left[1, g_r^k\right], \tag{4.17}$$

$$\xi_{tn}^{rk} \leq e_t^n x_{t-1,n}^{rk}, \forall r \in R, k \in K_r, t \in T \setminus \left[1, g_r^k\right], n \in N, \tag{4.18}$$

$$\xi_{tn}^{rk} \leq \tau_{tn}^{rk}, \forall r \in R, k \in K_r, t \in T \setminus \left[1, g_r^k\right], n \in N, \tag{4.19}$$

$$\xi_{tn}^{rk} \geq \tau_{tn}^{rk} + e_t^n x_{t-1,n}^{rk} - 1, \forall r \in R, k \in K_r, t \in T \setminus \left[1, g_r^k\right], n \in N, \tag{4.20}$$

$$\psi_{tn}^{rk} \leq \mu_{tn}^{rk}, \forall r \in R, k \in K_r, t \in T \setminus \left[1, g_r^k\right], n \in N, \tag{4.21}$$

$$\psi_{tn}^{rk} \leq \gamma_{tn}^{rk}, \forall r \in R, k \in K_r, t \in T \setminus \left[1, g_r^k\right], n \in N, \tag{4.22}$$

$$\psi_{tn}^{rk} \geq \mu_{tn}^{rk} + \gamma_{tn}^{rk} - 1, \forall r \in R, k \in K_r, t \in T \setminus \left[1, g_r^k\right], n \in N, \tag{4.23}$$

$$\alpha_{tn}^{rk} \leq \psi_{tn}^{rk} + \xi_{tn}^{rk}, \forall r \in R, k \in K_r, t \in T \setminus \left[1, g_r^k\right], n \in N, \tag{4.24}$$

$$\alpha_{tn}^{rk} \geq \frac{1}{2}(\psi_{tn}^{rk} + \xi_{tn}^{rk}), \forall r \in R, k \in K_r, t \in T \setminus \left[1, g_r^k\right], n \in N, \tag{4.25}$$

$$\alpha_{tn}^{rk} = 0, \forall r \in R, k \in K_r, t \in \left[2, g_r^k\right], n \in N, \tag{4.26}$$

$$\gamma_{tn}^{rk}, \eta_{tn}^{rk}, \mu_{tn}^{rk}, \psi_{tn}^{rk}, \xi_{tn}^{rk}, \tau_{tn}^{rk} \in \{0,1\}, \forall r \in R, k \in K_r, t \in T \setminus \left[1, g_r^k\right], n \in N, \tag{4.27}$$

$$\rho_{tn}^{rk} \leq \phi_{tn}^{rk} + \alpha_{tn}^{rk}, \forall r \in R, k \in K_r, t \in T \setminus \left[1, g_r^k\right], n \in N, \tag{4.28}$$

$$\rho_{tn}^{rk} \geq 1/2(\phi_{tn}^{rk} + \alpha_{tn}^{rk}), \forall r \in R, k \in K_r, t \in T \setminus \left[1, g_r^k\right], n \in N, \tag{4.29}$$

$$\theta_{tn}^{rk} \leq \rho_{tn}^{rk}, \forall r \in R, t \in T \setminus \{1\}, k \in K_r, n \in N, \tag{4.30}$$

$$\theta_{tn}^{rk} \leq 1 - x_{tn}^{rk} e_t^n, \forall r \in R, t \in T \setminus \{1\}, k \in K_r, n \in N, \tag{4.31}$$

$$\theta_{tn}^{rk} \geq \rho_{tn}^{rk} - x_{t,n}^{rk} e_t^n, \forall r \in R, t \in T \setminus \{1\}, k \in K_r, n \in N, \tag{4.32}$$

$$\rho_{tn}^{rk} \in \{0,1\}, \forall r \in R, k \in K_r, n \in N, t \in T \setminus \{1\}. \tag{4.33}$$

In summary, this work gives the following model:

$$\max \ \lambda + \epsilon_1 \sum_{r \in R} \beta_r \tag{4.34}$$

$$\begin{aligned} \text{s.t.} \quad & (3.2) - (3.5), (3.7), (3.9) - (3.11), (3.13) - (3.22), \\ & (3.26) - (3.37), (4.4) - (4.33), \end{aligned}$$

$$\beta_r, \lambda \in [1, |T|], \forall r \in R, i \in T, j \in T_i, \tag{4.35}$$

$$z_i^{jr} \in \{0,1\}, \forall r \in R, i \in T, j \in T_i. \tag{4.36}$$

When there are multiple solutions that maximize $\lambda + \epsilon_1 \sum_{r \in R} \beta_r$, the solution with a small number of backup functions may be preferable. For this purpose, the following objective function can be used:

$$\max \ \lambda + \epsilon_1 \sum_{r \in R} \beta_r - \epsilon_2 \sum_{r \in R} \sum_{k \in K_r} \sum_{t \in T} \sum_{n \in N} u_{tn}^{rk}. \tag{4.37}$$

The solution that minimizes the number of backup functions $\sum_{r \in R} \sum_{k \in K_r} \sum_{t \in T} \sum_{n \in N} u_{tn}^{rk}$ is chosen when there are multiple solutions that maximize $\lambda + \epsilon_1 \sum_{r \in R} \beta_r$. Therefore, a sufficiently small positive number, ϵ_2, is multiplied to the third term to prioritize the second term over the third term. ϵ_2 is given by $\frac{1}{\sum_{r \in R} \sum_{k \in K_r} g_r^k}$.

There may be several requests sharing the same SFC with different requests of package processing abilities. This work gives an extension of the proposed model to separate the requests from the SFCs in Section 4.5.1.

4.2.2 Analysis of proposed model

This subsection derives the upper bound of SSCAT provided by the proposed model.

Lemma 4.1 *The m_CALCULATION function in Algorithm 4.1 gives the minimum number of required nodes on which all functions in* $\boldsymbol{R}$ *are placed at time slot* $t \in T$, m_t.

Proof: This work proves this lemma by contradiction. If the result provided by *m*_CALCULATION is not the minimum number of required nodes at time slot $t \in T$, m_t, there is a set of $m'_t < m_t$ nodes that can accommodate all functions limited by the constraints. To get m'_t, all the functions

Algorithm 4.1 Calculate the number of nodes on which functions are placed at time slot t

Input: Set of nodes N, a time slot $t \in T$, set of SFCs R, set of ordered VNFs in SFC $r \in R$ K_r, set of the capacities of all nodes at time slot t C_t

Output: Set of numbers of key unavailabilities at each time slot S

function m_CALCULATION(N, t, R, K_r, C_t)
    Sort requests in R in non-increasing order of $|K_r|, r \in R$
    Sort node in N in non-increasing order of $c_t^n \in C_t$
    for $r \in R$ **do**
        for $k \in K_r$ **do**
            for $n \in N$ **do**
                if used capacity of n is less than c_t^n and any other functions in request r were not placed on node n **then**
                    Place the kth function in SFC r on n
                    Break
                else
                    Continue
                end if
            end for
        end for
    end for
    return the number of nodes on which functions are placed.
end function

placed on a node that belongs to a set of m_t nodes obtained by function m_CALCULATION to other nodes that have been assigned functions need to be moved. However, the reallocation causes a situation that at least one node accommodates more functions than it can accommodate, or two functions from the same SFC are placed on the same node according to the placement procedure at lines 35-40 of Algorithm 4.8. There is no valid m'_t such that it is less than m_t. ■

Let E_t be a set of unavailable nodes at time slot $t \in T$. Let $\Xi_t^{t'}, t \in T\backslash\{|T|\}, t' \in T\backslash\{1\}, t' > t$, denote a binary variable, which is set to 1 if $|N| - |E_t \cup E_{t'}| \geq m_{t'}$; 0 otherwise.

Lemma 4.2 *The upper bound of SSCAT provided by the proposed model without considering backup functions is* $\Delta \equiv \max_{t\in T\backslash\{|T|\},t'\in T\backslash\{1\},t'>t:\Xi_t^{t+1}=\Xi_t^{t+2}=\cdots=\Xi_t^{t'}=1}\{t'-t\}$.

Proof: m_t is the minimum number of required nodes at time slot $t \in T$ by using Lemma 1. If $|N| - E_t < m_t$, there are so many unavailable nodes that no enough space to place all functions in R at time slot t exists. If $|N| - |E_t \cup E_{t'}| < m_{t'}$, some functions migrate to avoid being placed in an unavailable node or there is no enough space for function placement. The longest consecutive duration for which all functions in R can be placed is the largest difference between t and t', where $\Xi_t^{t''} = 1, \forall t'' \in [t+1, t']$. ■

Let g denote the smallest $g_r^k, \forall r \in R, k \in K_r$. Let E' denote a set of $(n,t), n \in N, t \in T$, where each (n,t) has $e_t^n = 1, e_t^n \in E$, and there is any node $n' \in N\backslash\{n\}$ that satisfies the following conditions: $t > g$, $e_t^{n'} = e_{t-1}^{n'} = \cdots = e_{t-g}^{n'} = 0$, and $c_t^{n'}, c_{t-1}^{n'}, \cdots, c_{t-g}^{n'} \geq 1$. This work chooses $\theta \in [1, |E'|]$ different elements and store each possible combination as a set to set E'_θ. Let E'' denote a set of sets, $E'' = \{\emptyset \cup E'_1 \cup E'_2 \cup \cdots \cup E'_{|E'|}\}$. Let $E'''_a, a \in [1, |E''|]$, denote a set of $e_t^n, t \in T, n \in N$, in which e_t^n is set to 0 if (n,t) is in E''_a and other e_t^n has the same value with the corresponding $e_t^n \in E$. Lemma 1 obtains m_t by using E. This work calculates $\Delta_a, a \in [1, |E''|]$, with $m_t, \forall t \in T$, and E'''_a, by using Lemma 2.

Theorem 4.1 *The upper bound of SSCAT provided by the proposed model with considering backup functions is* $\max_{a\in[1,|E''|]} \Delta_a$.

Proof: This proof assumes that the recovery time is the smallest one among all recovery times g in this proof. This proof assumes that R_{B} is $\sum_{r \in R} \sum_{k \in K_r} g_r^k$. This proof relaxes the constraint on the required capacities of backup functions. E' stores the unavailable locations (n, t) that can be avoided by using backup functions limited by recovery time g. By calculating different combinations of elements in E', all the possible backup plans and store them into E'' can be known. The proof forms new availability schedule $E_a''' \in E''$ by assuming that the unavailable locations in E_a''' are recovered. The proof regards each modified availability schedule $E_a''', a \in [1, |E''|]$, as an availability schedule in the VNF placement model. The proof calculates Δ_a by using $E_a''', a \in [1, |E''|]$. The largest Δ_a is the upper bound of SSCAT in the proposed model. ■

4.2.3 Network-aware model

Section 4.2.1 does not consider the routing among VNFs in an SFC and the recovery path between primary and backup VNFs. Some paths cannot be chosen because of the limitation of the characteristic of links, such as the link capacity. The consideration of networks while deciding the VNF allocations can avoid problems such as the turn-back traffic and the long-distance traffic, which lead to inefficient resource consumption and increased latencies [89]. This subsection gives an extension on how to handle the routing problems by considering the networks.

This work uses d_r to express the required transmission resources of request $r \in R$. For link $(i, j) \in L$, this work uses b_{ij} and l_{ij} to express the available transmission resources and the end-to-end latency from node i to node j. The end-to-end latency contains propagation delay, packetization delay, and queueing delay. It represents the required time for data forwarding between two nodes. This work assumes that the end-to-end latency can be estimated [90], which is related to the network congestion and physical distance. For the estimation of end-to-end latency, some network measurement tools, e.g., ping, netperf, and TCPing for TCP transmission, are used. The flow constraint to

compute the routes of all SFCs is given by:

$$\sum_{(i,j)\in L} a_{w,ij} p_{rt}^{k,ij} = \begin{cases} -1, \text{ if } x_{tw}^{rk} = 1 \\ 1, \text{ if } x_{tw}^{r,k+1} = 1 \\ 0, \text{ if } x_{tw}^{rk} = x_{tw}^{r,k+1} = 0. \end{cases} \tag{4.38}$$

For each request, there are three types of nodes: source node (to which the first function of a request is allocated), destination node (to which the last function of a request is allocated) and others. This work defines indicator $a_{w,ij}, w \in N, (i,j) \in L$, to represent the adjacency of nodes on directed graph G, where $a_{w,ij} = 1$ if node w is the tail of the directed link (i,j), i.e., $w = j$; $a_{w,ij} = -1$ if node w is the head of the directed link (i,j), i.e., $w = i$; $a_{w,ij} = 0$ otherwise. This work uses binary variable $p_{rt}^{k,ij}$ to express the routes in an SFC. If link (i,j) is a segment link between the kth node and the $(k+1)$th node of request $r \in R$ at time slot $t \in T$, $p_{rt}^{k,ij} = 1$; otherwise, 0. According to (4.6), x_{tw}^{rk} and $x_{tw}^{r,k+1}$ cannot be 1 at the same time. Thus (4.38) can be simplified to:

$$\sum_{(i,j)\in L} a_{w,ij} p_{rt}^{k,ij} = -x_{tw}^{rk} + x_{tw}^{r,k+1}, \forall k \in K_r \setminus \{|K_r|\}, r \in R, \\ t \in T, w \in N. \tag{4.39}$$

The flow constraint to compute the routes for primary and backup synchronizations is given by:

$$\sum_{(i,j)\in L} a_{w,ij} q_{rt}^{k,ij} = \begin{cases} -1, \text{ if } x_{tw}^{rk} = \sum_{w'\in N} u_{tw'}^{rk} = 1 \\ 1, \text{ if } u_{tw}^{rk} = \sum_{w'\in N} x_{tw'}^{rk} = 1 \\ 0, \text{ otherwise.} \end{cases} \tag{4.40}$$

This work uses binary variable $q_{rt}^{k,ij}$ to express the routes between the primary VNF and the backup VNF of a function if there exists any backup. If link (i,j) is a segment link between the loactions of the primary node and the backup node of the kth VNF in request $r \in R$ at time slot $t \in T$, $q_{rt}^{k,ij} = 1$, and otherwise 0. According to (4.6), x_{tw}^{rk} and u_{tw}^{rk} cannot be 1 at the same time. Thus (4.40) can be simplified to:

$$\sum_{(i,j)\in L} a_{w,ij} q_{rt}^{k,ij} = -x_{tw}^{rk} \sum_{w'\in N} u_{tw'}^{rk} + u_{tw}^{rk} \sum_{w'\in N} x_{tw'}^{rk}, \forall k \in K_r, r \in R, t \in T, w \in N, \tag{4.41}$$

which is linearized to (4.42)-(4.49) with some auxiliary variables as follows:

$$\sum_{(i,j)\in L} a_{w,ij} q_{rt}^{k,ij} = -\zeta_{tw}^{rk} + \iota_{tw}^{rk}, \forall k \in K_r, r \in R, t \in T, w \in N, \tag{4.42}$$

$$\zeta_{tw}^{rk} \leq x_{tw}^{rk}, \forall k \in K_r, r \in R, t \in T, w \in N, \tag{4.43}$$

$$\zeta_{tw}^{rk} \leq \sum\nolimits_{w' \in N} u_{tw'}^{rk}, \forall k \in K_r, r \in R, t \in T, w \in N, \tag{4.44}$$

$$\zeta_{tw}^{rk} \geq x_{tw}^{rk} + \sum\nolimits_{w' \in N} u_{tw'}^{rk} - 1, \forall k \in K_r, r \in R, t \in T, w \in N, \tag{4.45}$$

$$\iota_{tw}^{rk} \leq u_{tw}^{rk}, \forall k \in K_r, r \in R, t \in T, w \in N, \tag{4.46}$$

$$\iota_{tw}^{rk} \leq \sum\nolimits_{w' \in N} x_{tw'}^{rk}, \forall k \in K_r, r \in R, t \in T, w \in N, \tag{4.47}$$

$$\iota_{tw}^{rk} \geq u_{tw}^{rk} + \sum\nolimits_{w' \in N} x_{tw'}^{rk} - 1, \forall k \in K_r, r \in R, t \in T, w \in N, \tag{4.48}$$

$$\zeta_{tw}^{rk}, \iota_{tw}^{rk} \in \{0,1\}, \forall k \in K_r, r \in R, t \in T, w \in N. \tag{4.49}$$

The link capacity constraint is given by:

$$\sum_{r \in R} \sum_{k \in K_r} (p_{rt}^{k,ij} + q_{rt}^{k,ij}) d_r \leq b_{ij}, \forall (i,j) \in L, t \in T. \tag{4.50}$$

Equation (4.50) ensures that each link's transmission resource is not overused.

This work takes the connections between nodes into consideration. The network-aware model is given by:

$$\max\ \lambda + \epsilon_1 \sum_{r \in R} \beta_r - \epsilon_3 \sum_{r \in R} \sum_{t \in T} \sum_{k \in K_r} \sum_{(i,j) \in L} l_{ij} p_{rt}^{k,ij} \tag{4.51a}$$

$$\text{s.t.}\ \ (3.2) - (3.5), (3.7), (3.9) - (3.11), (3.13) - (3.22), (3.26) - (3.37), (4.4) - (4.33), (4.39), (4.42) - (4.50) \tag{4.51b}$$

$$z_i^{jr} \in \{0,1\}, \forall r \in R, i \in T, j \in T_i. \tag{4.51c}$$

The solution that minimizes the sum of end-to-end latencies for all routes, $\sum_{r \in R} \sum_{t \in T} \sum_{k \in K_r} \sum_{(i,j) \in L} l_{ij} p_{rt}^{k,ij}$, is chosen when there are multiple solutions that maximize $\lambda + \epsilon_1 \sum_{r \in R} \beta_r$. Therefore, a sufficiently small positive number, ϵ_1, is multiplied to the second term of objective function to prioritize the first term over the second term; a sufficiently small positive number, ϵ_3, is multiplied to the third term to prioritize the second term over the third term. ϵ_3 is given by $\frac{1}{|R| \cdot |Kr| \cdot |T| \cdot |L|}$.

The model in Section 4.2.1 is a subset of the model in this subsection. Routing is given and the minimum of end-to-end latencies is considered in this subsection. If the SPs are only responsible for function allocations and not for routing and forwarding between different VNFs in the SFCs, the model in Section 4.2.1 is more suitable and faster than that in this subsection since the model in Section 4.2.1 has less decision variables and items in the objective function.

4.3 Heuristic algorithm

As the size of the ILP problem presented in Section 4.2.1 increases, the problem becomes more difficult to solve in practical time. A feasible solution may not be obtained within admissible computational time, which can be specified by SPs. Genetic algorithms have been used for VNF resource allocation problems in previous works, e.g., [91] and [92]. This work introduces a genetic-algorithm-based heuristic algorithm in this section based on that introduced in Chapter 3.

4.3.1 Overall structure

The running parameters for the heuristic algorithm are the same with those in 3.3, which are set before the running of the algorithm by experience. The procedure of the heuristic algorithm is shown in Fig. 4.2. At first, the algorithm generates a group of initial feasible solutions whose size is *IPN* by using the approach in Section 4.3.2. For each solution in the group, the algorithm uses three mutation approaches, *internal crossover*, *external crossover*, and *mutation*, to generate new solutions based on the selected solution. Each approach has a probability which decides if the approach is used for generating new solutions. The three approaches are introduced in Section 4.3.3. Newly generated solutions are added to the group of solutions. The algorithm calculates the fitness score for each solution by using the method in Section 4.3.5 and updates the best solutions. If the number of solutions exceeds the limitation *ULPN*, *ULPN* solutions are selected from the group for the next process by using the roulette gambler approach. The unselected solutions are deleted. The probability of a solution being selected is related to its fitness score; the

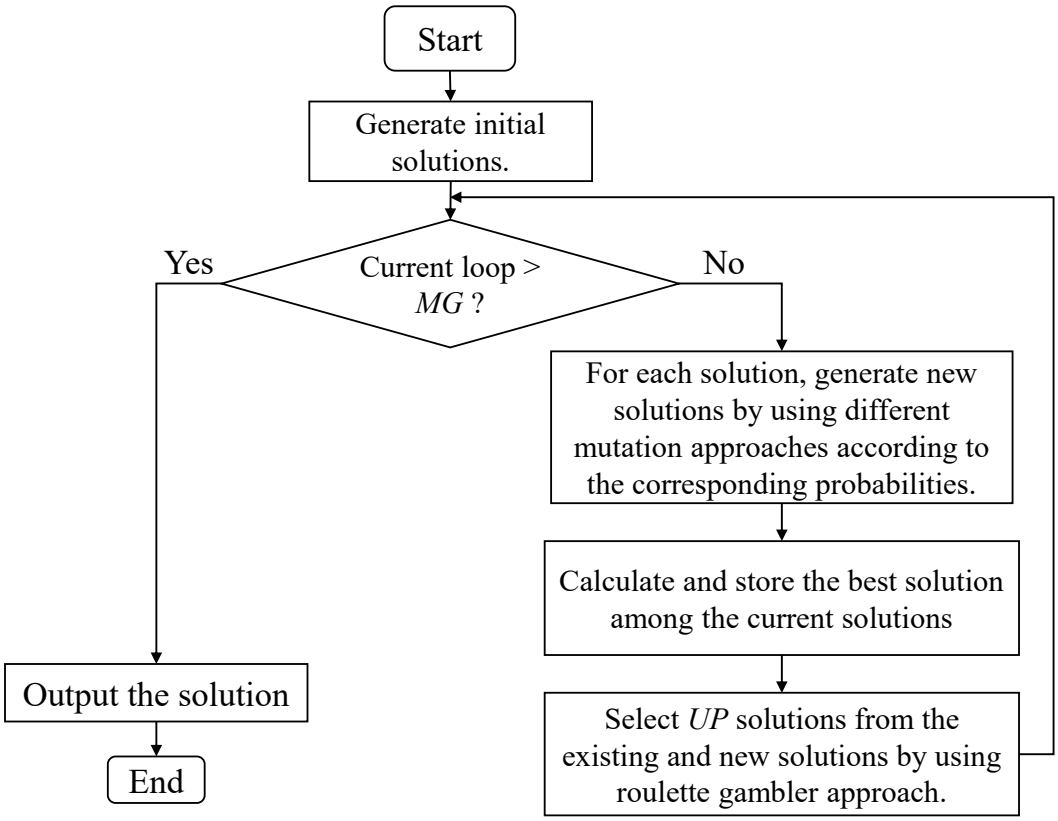


Figure 4.2: Procedure of heuristic algorithm.

higher the fitness score is, the higher the probability of the solution being selected is. The above steps are repeated MG times, and then the algorithm outputs the best solution explored in the procedures. The main function of the heuristic algorithm is shown in Algorithm 4.2. The algorithm calculates the allocations of primary and backup VNFs in given requests.

In line 1, Algorithm 4.3 introduced in Section 4.3.2 gives a set of initial feasible solutions for primary function allocation whose size is the initial population (IPN). In lines 2-4, if Algorithm 4.3 cannot provide a feasible solution, the heuristic algorithm reports that no feasible solution is found. In lines 5–28, the heuristic algorithm enters a loop with maximum generation (MG) cycles. In each cycle, the heuristic algorithm generates new feasible solutions by performing internal crossover (see function CROSS IN in Algorithm 4.5 of Section 4.3.3), external crossover (see function CROSS OUT in Algorithm 4.5

of Section 4.3.3), and mutation (see Algorithm 4.4 of Section 4.3.3) according to three probabilities: internal crossover probability (*ICP*); external crossover probability (*ECP*); and mutation probability (*MP*), respectively. Newly generated solutions in lines 9, 14, and 19 are stored in the new feasible solution set. They are added to the feasible solution set at a time in line 22. In line 23, the heuristic algorithm calculates the fitness for each solution and the backup function allocations for each function, and then returns the fitness score and the backup allocations (see Algorithm 4.7 in Section 4.3.5). In line 24, the genetic algorithm finds the solution with the highest fitness score and stores the primary and backup function allocation. In lines 25–27, if the size of the feasible solution set exceeds the upper limitation of population (*ULPN*), the heuristic algorithm chooses *UP* feasible solutions as a new set of feasible solutions according to roulette gambler (see Algorithm 3.6 in Section 4.3.4).

Algorithm 4.2 Main function

Input: N, T, R, K_r, $c_{nt}^s \in C$, *IPN, MG, ECP, ICP, MP, ULPN*

Output: primary and backup function allocations

1: $I \leftarrow$ Generate set of initial feasible solutions by using function *init_chromos* in Algorithm 4.3
2: **if** $I = \emptyset$ **then**
3: **return** No feasible solution
4: **end if**
5: **for** $step = 1 \rightarrow MG$ **do**
6: Define I_{n} as the new feasible solution set
7: **for** each solution in I **do**
8: **if** a random number in $[0, 1] > 1 - ICN$ **then**
9: $I_{\mathrm{n}} \leftarrow$Generate a non-redundant and mutant solution by using function *cross_in* in Algorithm 4.5 whose inputs are the selected solution in I and random time slot t
10: **end if**
11: **end for**
12: **for** each solution in I except for the first one **do**
13: **if** a random number $[0, 1] > 1 - ECP$ **then**
14: $I_{\mathrm{n}} \leftarrow$Generate a non-redundant and mutant solution by using function *cross_out* in Algorithm 4.5 whose inputs are the selected solution and its previous solution in I
15: **end if**
16: **end for**
17: **for** each solution in I **do**
18: **if** a random number $[0, 1] > 1 - MP$ **then**

19: $I_n \leftarrow$Generate a non-redundant and mutant solution by using function *mutation* in Algorithm 4.4 whose input is the selected solution in I
20: **end if**
21: **end for**
22: Integrate I_n into I
23: Calculate the fitness score of the solutions in I by using function *calc_fin_ness* in Algorithm 4.7
24: Store the solution with the highest fitness score
25: **if** size of $I > ULPN$ **then**
26: Reduce the size of the set to $ULPN$ by using function *roulette_gambler* in Algorithm 4.6
27: **end if**
28: **end for**

4.3.2 Initial solution generation

The heuristic algorithm generates a group of initial feasible solutions for primary VNF allocations by using the function described in this subsection. Based on this group, more feasible solutions can be generated by the mutation approaches in Section 4.3.3.

In the heuristic algorithm, each solution is a three-dimensional matrix. The first dimension represents time slots, the second one represents requests, and the third one represents functions. The value of an element whose location is (t, r, k) is the allocation of the kth function of request r at time slot t, which belongs to N.

This algorithm reorders the set of chains in non-increasing order of the number of VNFs in this chain. The algorithm tries to allocate each function to each node and ensure that the capacity of the node does not exceed. If all the functions of all chains among all time slots can be allocated, an initial solution is generated and stored. Since one initial solution is not enough for providing variations of the solutions, the algorithm duplicates the generated initial solution to a given parameter, initial population (IPN). A relatively large population can speed up the convergence of the algorithm and provide more variations of the solutions.

Algorithm 4.3 Initial solution

1: **function** INIT_CHROMOS(E)
2: Set of initial solutions $s \leftarrow \phi$

3: Sort requests in R in non-increasing order of K_r
4: **for** each time slot in T **do**
5: Sort nodes in N in non-increasing order of time from time slot t to a time slot in which a node becomes unavailable. If the above values are the same for some n, sort them in non-increasing order of time from time slot t to a time slot in which a node becomes unavailable secondly. If the above values are the same for some n, sort them in non-increasing order of time from time slot t to the last time slot in which a node becomes unavailable.
6: **for** $r \in R$ **do**
7: **for** $f = 1 \rightarrow K_r$ **do**
8: **for** $n \in N$ **do**
9: **if** used capacity of n is less than c_n^t AND any other functions in r were not allocated to n **then**
10: Allocate the fth function in SFC r to n
11: Break
12: **else**
13: Continue
14: **end if**
15: **end for**
16: **end for**
17: **end for**
18: Store the allocation to s
19: **end for**
20: Duplicate a solution iteratively until the number of solutions in s becomes *IPN*
21: **return** s
22: **end function**

Algorithm 4.3 reorders set R according to the corresponding K_r from long to short first (line 3). Then, it performs function allocation one by one (lines 4-19). At each time slot, the heuristic algorithm reorders set N according to the occurrence of unavailabilities from late to early (line 5). Then the heuristic algorithm allocates the functions to physical nodes according to these new orders (lines 9-13). Finally, the heuristic algorithm duplicates a solution iteratively until the number of solutions becomes initial population (IPN) (line

20).

4.3.3 Mutation

The heuristic algorithm provides three approaches for generating new solutions based on the current solutions so that better solutions may appear. One of the approaches for generating new solutions for primary VNF allocations is *mutation* based on an existing solution. The other two approaches are *internal crossover* and *external crossover*, which generate the solutions based on the length of each SFC. On the other hand, *mutation* generates a new solution based on the SCAT of each SFC.

The algorithm calculates the SCAT for all SFCs in the input solution. The algorithm sorts chains and nodes in weighted randomized order: the larger the SCAT is, the higher the order of the corresponding chain is; the larger the time from the first time slot to a time slot where a node becomes unavailable is, the higher the order of the corresponding node is. The algorithm generates a new solution with the above orders and the same method as the generation of the initial solution.

Algorithm 4.4 Mutation

1: **function** MUTATION($s \leftarrow$ the selected solution, E)
2: Calculate the SCAT for all SFCs in solution s
3: Sort requests in R in a weighted randomized order. The larger the SCAT, the higher the order of the corresponding r.
4: Sort nodes in N in a weighted randomized order. The larger the time from the first time slot to a time slot where a node becomes unavailable, the higher the order of the corresponding n.
5: **for** $r \in R$ **do**
6: **for** $f = 1 \rightarrow K_r$ of request r **do**
7: **for** $n \in N$ **do**
8: **if** used capacity of n is less than c_n^t AND any other functions in r were not allocated to n **then**
9: Allocate the fth function in SFC r to n
10: Break
11: **else**

```
12:                 Continue
13:             end if
14:         end for
15:     end for
16: end for
17: return new solution
18: end function
```

The internal crossover, function *cross_in* in Algorithm 4.5, crosses adjacent time slots in the same solution. The aim of *cross_in* is to suppress the reallocations of VNFs between adjacent time slots.

The external crossover, function *cross_out* in Algorithm 4.5, crosses the same time slot between two solutions in the feasible solution set. A new solution is generated by modifying the VNF allocation in a randomly selected time slot of one solution based on that of another solution.

Algorithm 4.5 Crossover

```
1: function CROSS_IN(s ← the selected solution , t ← the random time)
2:     s[t] ← s[t + 1]
3:     return s
4: end function
5: function CROSS_OUT(s_1, s_2)
6:     location ← a random integer in [1, |T|]
7:     s_c ← s_1
8:     s_c[location] ← s_2[location]
9:     return s_c
10: end function
```

4.3.4 Choice of solutions

The heuristic algorithm uses roulette wheel selection to create a new feasible solution set by choosing UP solutions from the feasible solution set.

In the *roulette_gambler* and *choice* functions in Algorithm 3.6, input $chroms$ is the set of solutions and fit_pros is the set of the fitness scores of the corresponding solutions in $chroms$.

Algorithm 4.6 Choice

```
1: function ROULETTE_GAMBLER(fit_pros, chroms)
```

```
    pick ← a random number in [0, 1]
    for j = 1 → |chroms| do
        pick ← pick − fit_pros[j]/sum(fit_pros)
        if pick ≤ 0 then
            return j
        end if
    end for
    return |chroms| − 1
end function
function CHOICE(chroms, fit_pros)
    choice_gens ← ϕ
    for i = 1 → min(|chroms|, UP) do
        j ← ROULETTE_GAMBLER(fit_pros, chroms)
        append chroms[j] to choice_gens
    end for
    return choice_gens
end function
```

4.3.5 Calculation of fitness

The algorithm computes the fitness score for each solution by using (3.8) and the possible backup function allocations. The algorithm finds the noncontinuous allocation of each chain. The algorithm calculates possible backup function allocations to extend the SCAT of each chain. There is a probability of applying the backup strategy (BP). Finally, the algorithm returns the calculated fitness score defined in (4.1), (3.7)-(3.6), and (3.8), and the corresponding backup function allocations.

Algorithm 4.7 Fitness calculation

```
function CALC_FIN_NESS(s ← the selected solution, E)
    Calculate the available resources of each node in solution s at each time slot
    for r ∈ R do
        for t ∈ T\{1} do
            if Allocation of functions in request r at time slot t is different from that at
    time slot t − 1 then
                for f = 1 → K_r of request r do
```

7: **if** the allocations of the kth function in request r at time slot t and $t-1$ are different AND t is larger than g_r^k **then**
8: $n \leftarrow$ the allocation of the kth function in request r at time slot t
9: **for** $t2 \leftarrow t - g_r^k 1, ..., t-1$ **do**
10: **if** n has no enough resources at time slot $t2$ OR any other function of request r is not allocated to n at time slot $t2$ **then**
11: Request r is not continuous between time slots $t-1$ and t
12: Break
13: **end if**
14: $diff \leftarrow (t, r, k, n)$
15: **end for**
16: **end if**
17: **end for**
18: **if** Continuity of request r between time slots $t-1$ and t has not been decided AND *diff* is not empty AND there are enough resources for allocations stored in *diff* AND a random number $[0, 1]$ >1- BP **then**
19: Request r is continuous between time slots $t-1$ and t
20: Merge set *diff* to backup function allocations and update the available resources of each node
21: **else**
22: Request r is not continuous between time slots $t-1$ and t
23: **end if**
24: **else if** Allocate any function in request r on an unavailable node at time slot t or $t-1$ **then**
25: Request r is not continuous between time slots $t-1$ and t
26: **else**
27: Request r is continuous between time slots $t-1$ and t
28: **end if**
29: **end for**
30: **end for**
31: Calculate the SCAT for all SFCs in solution s
32: **return** min($SCAT$)+sum($SCAT$)/($|T| \times |R|$), backup allocations
33: **end function**

4.4 Evaluations

This work compares the proposed model with the allocation model in Section 3.2.1, three baseline models are introduced in Section 3.4: a persistence allocation model, a single-slot allocation model, and a double-slot allocation model. This work compares the SSCATs provided by the above five models.

Table 4.1: Evaluation conditions.

Case	Nodes	Cap.	Req.	Func.	Time slots	Max. unavailabilities
1	8	2	4	3, 2, 2, 4	6	2
2	8	2	4	3, 2, 2, 4	6	6
3	16	2	8	6, 3, 2, 2, 4, 4, 3, 2	6	2

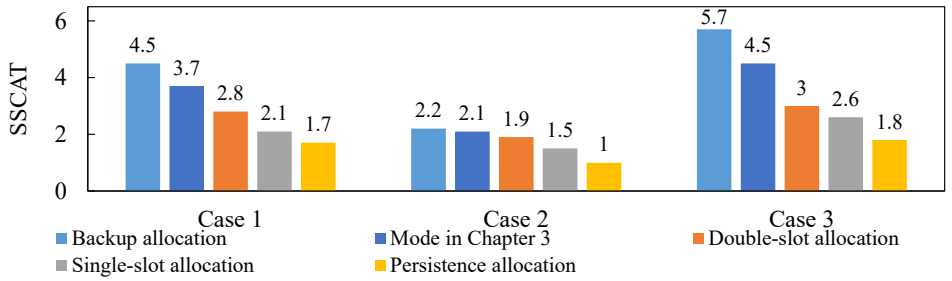
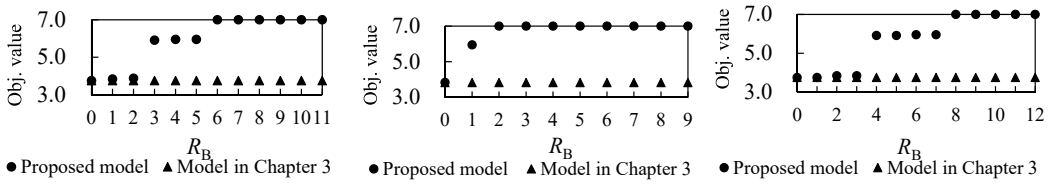


Figure 4.3: SSCAT values yielded by four models.

The persistence allocation model is implemented by Python 3.7, using Intel Core i7-7700 3.60 GHz 4-core CPU, 32 GB memory. The ILP problems of the other models are solved by the IBM(R) ILOG(R) CPLEX(R) Interactive Optimizer with version 12.7.1 [82] and run on the same hardware.

There are two tests for the comparison. In the first test, this work sets $R_{\mathrm{B}} = \sum_{r \in R} \sum_{k \in K_r} g_r^k$ in the proposed model and compares the result with other models. Three cases are examined in the first test. The conditions of these cases are shown in Table 4.1. In this table, *Nodes* means the number of nodes; *Capacity* means the capacity of each node at each time slot; *Requests* means the number of functions in each request; *Functions* means the number of functions in each request; *Time slots* means the number of time slots; *Maximum Unavailabilities* means the maximum number of unavailable nodes at a time slot. In all cases, this work obtains the VNF allocation of the proposed model by solving the ILP problem in Section 4.2. In cases 1, 2 and 3, this work randomly generates ten different availability schedules. In each case, this work computes the VNF allocations in the same network. This yields the computed SSCAT values which are then averaged. Fig. 4.3 shows the SSCAT values yielded by the four models.



(a) Case 1. (b) Case 2. (c) Case 3.

Figure 4.4: Dependency of objective value on R_{B}.

In the second test, this work compares the results with different values of R_{B}. There are three cases in the second test. In case 1, four SFCs are needed to be allocated to an eight-node network in six time slots. The lengths of these SFCs are four, three, two, and two, respectively. The unavailabilities are: $e_2^1, e_2^2, e_4^7, e_4^8, e_6^1, e_6^2$ are 1, and other e_t^n are 0. The recovery time is one. In case 2, the number of SFCs is reduced from four to three compared with case 1. The lengths of these SFCs are four, three, and two, respectively. In case 3, the length of the third SFC in case 1 is increased from two to three.

Figure 4.4 shows the dependency of objective value on R_{B}. With increase of R_B, there are three obvious increments of objective values in Figs. 4.4 (a), (b), and (c). There is a set of unavailabilities in availability schedule that are bottlenecks to reach higher SSCAT, which causes an obvious increment if R_B increases. This work calls these unavailabilities *key unavailabilities*. This work develops Algorithm 4.8 to estimate the number of key unavailabilities in each time slot. With this estimate, SPs can find the unavailable nodes which are the bottlenecks to increase SSCAT at each time slot. These unavailabilities are required to be eliminated priorly so that SPs can increase the service continuous available time with the least cost on failure recovery.

4.4.1 Impact of different types of availability schedules

The distributions of unavailabilities in the availability schedules impact the SSCAT values obtained by the proposed model. In addition, the effect of backup functions may be different in different types of availability schedules.

To demonstrate this impact, the evaluation randomly generates five different types of availability schedules with different positions of the unavailable

Algorithm 4.8 Calculate the number of key unavailabilities

Input: Set of nodes N, set of time slots T, set of SFCs R, set of ordered VNFs in SFC $r \in R$ K_r, set of the capacities of all nodes at all time slots C, availability schedule E

Output: Set of numbers of key unavailabilities at each time slot S

Set $U \leftarrow \emptyset$
Define u_t for storing the number of key unavailabilities at time slot $t \in T$ and set $u_t \leftarrow 0$.
for $t = 1 \rightarrow |T|$ **do**
    $m \leftarrow m\_\text{CALCULATION}(N, T, R, K_r, C_t)$.
    if $m > |N|$ **then**
        return No solutions.
    end if
    Check the nearest time slot $t' < t$, which has a different set of unavailabilities from those in t.
    if t' exists **then**
        Define N' as a set of $n \in N$ where $e_n^t = 1$ or $e_n^{t'} = 1$.
        if $|N| - |N'| < m$ **then**
            $u_t \leftarrow m - |N| + |N'|$. Add u_t to U.
        end if
    else
        if $|N| - \sum_{n \in N} e_n^t < m$ **then**
            $u_t \leftarrow m - |N| + \sum_{n \in N} e_n^t$. Add u_t to U.
        end if
    end if
end for
A_k is a set of all combinations of $k \in [1, |U|]$ elements in U, A_k^a is the ath element of A_k, or $A_k^a \in A_k, a \in [1, |A_k|]$.
$S \leftarrow \emptyset, s \leftarrow 0$
for $k = 1 \rightarrow |U|$ **do**
    for $a = 1 \rightarrow |A_k|$ **do**
        $v \leftarrow$ the objective value computed by assuming that $u_t \in A_k^a$ unavailable nodes become available at time slot t. u_t unavailable nodes are chosen in the following sets of unavailable nodes at time slot t in order they are found: new unavailable nodes at t; unavailable nodes chosen before t. In each set, any nodes are chosen.
        if $v >= s$ **then**
            if $v = s$ and $|A_k^a| > |S|$ **then**
                Continue
            end if
            $S \leftarrow A_k^a, s \leftarrow v$
        end if
    end for
end for

Table 4.2: Comparison between proposed model and model in Chapter 3 with different availability schedules.

Model	Type	SSCAT	Sum of SCATs
Chapter 3	1	6.0	24.0
	2	3.0	18.9
	3	3.0	18.0
	4	2.9	15.8
	5	2.8	15.4
This work	1	6.0	24.0
	2	3.0	19.3
	3	3.0	18.3
	4	3.0	18.0
	5	3.0	18.1

nodes in each time slot and calculate the SSCAT values with these availability schedules under the same condition. The evaluation considers an eight-node network, where the capacity of each node is set to two. The evaluation considers four requests with lengths of three, two, two, and four, respectively. The evaluation considers six time slots in this demonstration. In each time slot, there are two node unavailabilities. From types 1 to 5, the positions of unavailabilities become more random and scattered. The unavailabilities in the five types of schedules appear on two, three, four, five, and six nodes, respectively. The evaluation randomly generates ten availability schedules for each type and calculate the SSCATs and the sums of SCATs under all availability schedules. The average values of ten schedules belonging to the same type are shown in Table 4.2.

Table 4.2 shows the SSCAT values for the five types of availability schedules given by the proposed model and the model in Chapter 3. The proposed model provides a lower improvement on SSCATs if the positions of unavailabilities are more compact, such as type 1, and provides a higher improvement on SSCATs if the positions of unavailabilities are more sparse, such as type 5.

4.4.2 Analysis of network-aware allocation model

This subsection evaluates the performance of the proposed network-aware model in Section 4.2.3 compared with the model in Section 4.2.1 which does not consider routing and the model in Chapter 3, which considers routing without backup VNFs,

This evaluation uses four types of networks in this analysis: single data center of fat tree topology, multiple data centers dispersed geographically of fat tree topology [89], BCube topology [93], and visible light communication (VLC) networking topology [94]. The networks are shown in Fig. 4.5. □ represents the network devices, e.g., switches and routers. ◯ represents the nodes to which the VNFs can be allocated. Both two types of elements are considered as nodes in the model proposed in section III.C. To distinguish different nodes, the evaluation sets different capacities for them since VNFs cannot be allocated to switches and routers. The capacities of □ and ◯ are set to 0 and 2, respectively. The allocations among six time slots are considered. In Figs. 4.5(a) and (b), the available transmission resources of links between layers 0 and 1, 1 and 2, 2 and 3, and 3 and 4 are set to 16, 8, 4, and 2 units, respectively. The end-to-end latencies of links between layers 0 and 1, 1 and 2, 2 and 3, and 3 and 4 are set to 8, 4, 2, and 1 units, respectively. In Fig. 4.5(c), the available transmission resources of links in layer 0 and those between layers 0 and 1 are set to 8 and 4 units, respectively. The end-to-end latencies of links in layer 0 and those between layers 0 and 1 are set to 1 and 4 units, respectively. In Fig. 4.5(d), the available transmission resources of links between the router and access points, and those between access points and terminals are set to 8 and 4 units, respectively. The end-to-end latencies of links between the router and access points, and those between access points and terminals are set to 1 and 8 units, respectively. In Figs. 4.5(a) and (d), there are three requests with lengths of three, two, and two, respectively. The availability schedule used in these graphs is shown in Fig. 4.6(a). In Figs. 4.5(b) and (c), this evaluation considers four requests with lengths of four, three, two, and two. The availability schedules used in these graphs are shown in Figs. 4.6(b) and (c), respectively.

Observed from Table 4.3, the network-aware proposed model reduces the

Table 4.3: Comparison among proposed model with and without network-aware allocation and model in Chapter 3.

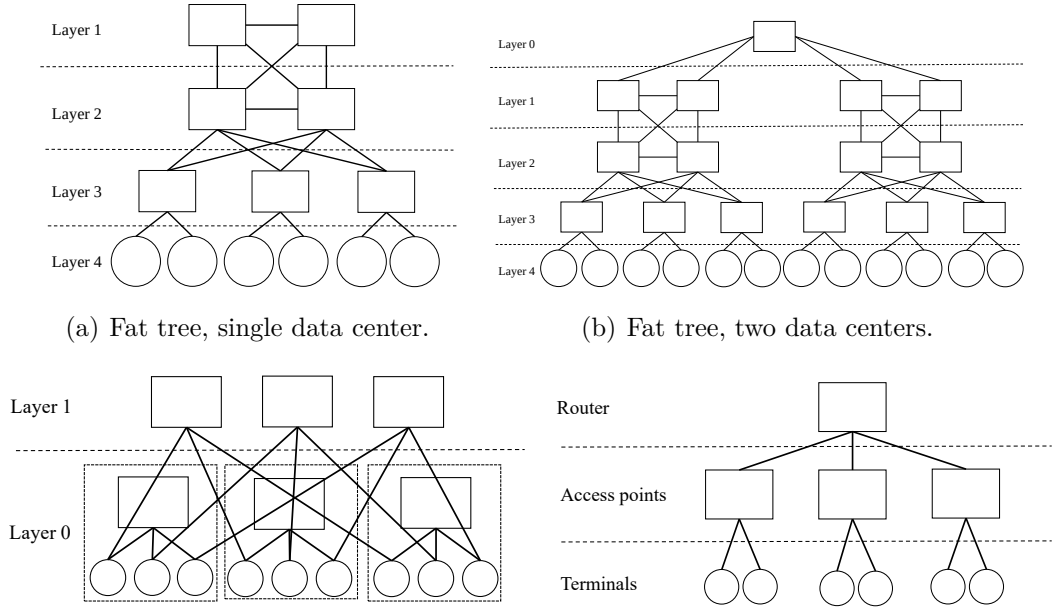
Model	Type†	SSCAT	Sum of SCATs	Sum of latencies
Chapter 3	S	3	9	84
	D	4	16	244
	B	3	17	198
	V	3	9	402
Network-aware proposed model	S	4	15	96
	D	5	23	176
	B	5	22	216
	V	4	15	402
Non-network-aware proposed model	S	4	15	156
	D	5	23	1640
	B	5	22	580
	V	4	15	410

†Type S: fat tree, single database network. Type D: fat tree, double database network. Type B: BCube network. Type V: VLC network.

latencies of services with keeping the same values of SCATs of services compared with the non-network-aware proposed model. Compared with the model in Chapter 3, the network-aware proposed model improves the value of SSCAT but the latencies of services also increase.

4.4.3 Analysis of heuristic algorithm

This subsection evaluates the performance of the proposed model with the heuristic algorithm introduced in Section 4.3. This evaluation compare the performances between the ILP approach and the heuristic algorithm in a 16-node network as the first test. The evaluation compares the performances among the proposed model with the heuristic algorithm and the five baseline



(a) Fat tree, single data center.

(b) Fat tree, two data centers.

(c) BCube.

(d) VLC.

Note: □: switch or router. ◯: node to which the VNFs can be allocated.

Figure 4.5: Two graphs for tests of network-aware allocation model.

models in a 100-node network during 365 time slots based on the Microsoft Azure maintenance records in [95].

The heuristic algorithm is designed to reduce the computation time within an acceptable performance degradation. The evaluation evaluates the performance of the proposed model with the heuristic algorithm in terms of the objective value calculated by (4.34), SSCAT, the sum of SCATs, and the computation time. The evaluation considers a 16-node network, where the capacity of each node is set to five. The evaluation considers seven requests with lengths of three, two, two, three, two, two, and four, respectively. The recovery time of each function is set to one time slot. The evaluation considers six time slots in this analysis. In each time slot, there are five node unavailabilities. The evaluation randomly generates ten different availability schedules with different positions of the unavailable nodes in each time slot and calculate the values with these availability schedules under the same condition. The evaluation compares the results obtained by the proposed model with the heuristic

Time slot \ Node	1	2	3	4	5	6
1	U			U		
2		U				
3						U
4					U	
5		U		U		
6			U			

(a) Availability schedules for the tests using single data center and VLC networks.

Time slot \ Node	1	2	3	4	5	6	7	8	9	10	11	12
1	U			U			U					
2		U						U				
3						U						U
4					U						U	
5		U						U		U		
6			U						U			

(b) Availability schedules for the tests using double data center network.

Time slot \ Node	1	2	3	4	5	6	7	8	9
1	U			U			U		
2		U						U	
3						U			
4					U				
5		U						U	
6			U						U

(c) Availability schedules for the tests using BCube network.

Figure 4.6: Availability schedules for tests of network-aware allocation model.

algorithm, the ILP approach of the proposed model, and the ILP approach of the model in Chapter 3.

The heuristic algorithm is implemented by C++, compiled by Microsoft Visual Studio Community 2019 with version 16.8.3, using Intel Core i7-7700 3.60 GHz 4-core CPU, 32 GB memory.

Table 4.4 shows the parameter settings used in this evaluation. Table 4.5 shows the results in ten tests and Table 4.6 shows the average differences between the ILP approach and the heuristic algorithm in terms of objective val-

Table 4.4: Parameter setting in heuristic algorithm.

Parameter	Setting
MG	100
IPN	12
$ULPN$	60
ICP	0.6
ECP	0.4
MP	0.3
BP	0.9

ues and computation times among ten tests. The heuristic algorithm reduces 66.75% computation time with a 1.57% performance degradation in terms of the objective value on average in the examined test cases. In test 10, the allocation result obtained by the heuristic algorithm has a lower objective value than that obtained by the ILP approach, but the objective value of the allocation obtained by the heuristic algorithm is still larger than that of Chapter 3.

The evaluation needs to decide the allocations on a 100-node network during 365 time slots in the second test. The evaluation considers 40 requests. The lengths of three requests are seven. The lengths of 10 requests are five. The lengths of 20 requests are four. The lengths of five requests are three. The lengths of two requests are two. The recovery time of each function is set to one time slot. The capacity of each node is two. The heuristic algorithms of the proposed model, the model in Chapter 3 and the persistent placement model are implemented by C++, compiled by Microsoft Visual Studio Community 2019 with version 16.8.3, using Intel Core i7-7700 3.60 GHz 4-core CPU, 32 GB memory. The other baseline models are solved by the IBM(R) ILOG(R) CPLEX(R) Interactive Optimizer with version 12.10.0 and run on the same hardware.

Table 4.7 shows the comparison results. The proposed model provides the maximum objective value and has the longest computation time among the

Table 4.5: Comparisons among ILP approach of Chapter 3 and ILP approach and heuristic algorithm of proposed model.

Test	ILP approach of Chapter 3				ILP approach of proposed model				Heuristic algorithm of proposed model			
	SSCAT	$\sum_{r \in R} \beta_r$	Obj. value	Time [s]	SSCAT	$\sum_{r \in R} \beta_r$	Obj. value	Time [s]	SSCAT	$\sum_{r \in R} \beta_r$	Obj. value	Time [s]
1	5	39	5.93	8.93	6	42	7.00	3.80	6	42	7.00	1.35
2	5	35	5.83	6.28	6	42	7.00	3.34	6	42	7.00	1.31
3	5	38	5.90	8.17	6	42	7.00	3.43	6	42	7.00	1.22
4	5	39	5.93	11.95	6	42	7.00	3.72	6	42	7.00	1.13
5	5	39	5.93	9.20	6	42	7.00	3.40	6	42	7.00	1.32
6	3	27	3.64	9.84	6	42	7.00	5.62	6	42	7.00	1.76
7	5	38	5.90	13.06	6	42	7.00	3.69	6	42	7.00	1.54
8	5	38	5.90	10.13	6	42	7.00	4.33	6	42	7.00	1.17
9	5	39	5.93	9.80	6	42	7.00	3.37	6	42	7.00	1.19
10	4	34	4.81	26.71	6	42	7.00	5.03	5	37	5.88	1.19

Table 4.6: Comparison of average computation times and objective values obtained by ILP approach and heuristic algorithm.

Method	Computation times [s]	Objective values
ILP approach	3.97	7.00
Heuristic algorithm	1.32	6.89
Difference (%)	66.75	1.57

Table 4.7: Comparison of average computation times and objective values obtained by proposed model with heuristic algorithm and four baseline models.

Method	Computation times [s]	Objective values
Proposed model	22248.00	14.05
Model in Chapter 3	22076.40	13.06
Double-slot placement	2298.609	9.034
Single-slot placement	517.6713	2.010
Persistent placement	1.345946	6.032

five examined models in this case. The single-slot placement outperforms the persistent placement since the unavailabilities in the given availability schedule are loose. The effect of the unavailable nodes on the persistent placement is weaker than that of the reallocations between different time slots on the single-slot placement. If both effects by using the double-slot placement are considered, the double-slot placement outperforms both of them in terms of the objective value with the cost of longer computation time. The proposed model outperforms the model in Chapter 3 with less cost on computation time. It is valuable to take backup measures by using the proposed model in the examined case compared with the model in Chapter 3.

4.5 Discussions

4.5.1 Separate SFCs from requests

This work assumes that one request corresponds to one SFC in Section 4.2. There may be several requests sharing the same SFC with different requests of package processing abilities. This work gives an extension of the proposed model to separate the requests from the SFCs. This work redefines the requests, SFCs, VNFs, and recovery time to replace the definitions in Section 4.2.1. The definitions of decision variables x_{tn}^{rk} and u_{tn}^{rk} are changed as follows.

R represents the set of requests from the users. C represents the set of SFCs waiting for provisions. Each SFC is an ordered set of VNFs. F is the set of functions. $F^c \subseteq F$ is the ordered set of VNFs used in SFC $c \in C$. Binary given parameter ψ_{fc} is set to 1 if function $f \in F$ is used in SFC $c \in C$; 0 otherwise. Binary given parameter γ_{cr}, $r \in R$, $c \in C$, is set to 1 if request r requests SFC c; 0 otherwise. Given parameter q_s^f represents the amount of resource $s \in S$ which function $f \in F$ requires. g_f is a given parameter which represents the recovery time of function $f \in F$.

This work uses binary decision variable x_{tn}^f to represent the allocation of primary VNF; x_{tn}^f is set to 1 if function $f \in F$ is assigned to node $n \in N$ at time slot $t \in T$; 0 otherwise. This work uses binary decision variable u_{tn}^f to represent the allocation of backup VNF; u_{tn}^f is set to 1 if function $f \in F$ is assigned to node $n \in N$ at time slot $t \in T$; 0 otherwise.

α_{tn}^{rk} in Section 4.2.1 is redefined to α_{tn}^f, which denotes a binary variable which is set to 1 if function $f \in F$ is prevented from the interruption happening on node $n \in N$ at time slot $t \in T$; 0 otherwise, $\forall f \in F, n \in N$, α_{tn}^f is expressed by:

$$\alpha_{tn}^{rk} = \begin{cases} \{(\vee_{n' \in N \setminus \{n\}} e_t^{n'} x_{t-1,n'}^f) \wedge ((\prod_{t' \in [t-g_f,\ t-1]} \\ u_{t'n}^f) \wedge x_{tn}^f)\} \vee \{(e_t^n x_{t-1,n}^f) \wedge (\vee_{n' \in N \setminus \{n\}} (\\ \prod_{t' \in [t-g_f,\ t-1]} u_{t'n'}^f) \wedge x_{tn'}^f)\}, t \in T \setminus [1,\ g_f] \\ 0, t \in [2,\ g_f], \end{cases} \tag{4.52}$$

According to the above redefinitions, this work gives a new version of the

related parameters and constraints as follows:

$$o_t^r = \prod_{n\in N}\Bigg\{\Bigg(\prod_{c\in C}\prod_{f\in F^c}\psi_{fc}\gamma_{cr}(x_{tn}^f \odot x_{t-1,n}^f)\Bigg)\wedge\Bigg(1-(\prod_{c\in C}\prod_{f\in F^c}$$
$$\psi_{fc}\gamma_{cr}x_{tn}^f)\wedge e_t^n\Bigg)\wedge\Bigg(1-(\prod_{c\in C}\prod_{f\in F^c}\psi_{fc}\gamma_{cr}x_{t-1,n}^f)\wedge e_{t-1}^n\Bigg)\Bigg\},$$

$$\forall r\in R, t\in T\backslash\{1\}, \tag{4.53a}$$

$$\sum_{f\in F}(x_{tn}^f + u_{tn}^f)q_s^f \le c_t^n, \forall n\in N, t\in T, \tag{4.53b}$$

$$\sum_{f\in F^c}(x_{tn}^f + u_{tn}^f) \le 1, \forall c\in C, n\in N, t\in T, \tag{4.53c}$$

$$\sum_{n\in N} x_{tn}^f = 1, \forall f\in F, t\in T, \tag{4.53d}$$

$$\sum_{n\in N} u_{tn}^f \le 1, \forall f\in F, t\in T. \tag{4.53e}$$

Equation (4.53a) replaces x_{tn}^{rk} in (3.1) with $\prod_{c\in C}\prod_{f\in F^c}\psi_{fc}\ \gamma_{cr}x_{tn}^f$. Equation (4.53b) replaces (3.9) and ensures that each node's computational resources must not exceed its capacity during allocation. Equation (4.53c) replaces (3.10) and assumes that one service chain does not allocate multiple VNFs in this chain on one VM to avoid the influence of the reallocation of VMs [44]. Equation (4.53d) replaces (3.11) and ensures that all functions are allocated in the network. Equation (4.53e) replaces (4.8) and assumes that each VNF has at most one backup at one time slot.

4.5.2 Replicas for primary and backup VNFs

VNF replicas in an SFC are used to prevent service interruptions from the failures of functions and load balancing. In Section 4.2, the proposed model only uses at most two replicas, one for primary VNF and one for backup VNF, which are limited by (3.11) and (4.8). A possible extension for the proposed model is to support multiple replicas for primary and backup VNFs to provide distributed backups and load balancing.

Let ρ_t^{rk} be an non-negative integer given parameter, which represents that ρ_t^{rk} active replicas are necessary at time slot $t\in T$ for the $k\in K_r$th function

in SFC $r \in R$. Equations (3.11) and (4.8) are replaced by:

$$\sum_{n \in N} x_{tn}^{rk} = \rho_t^{rk} \forall r \in R, k \in K_r, t \in T. \tag{4.54}$$

4.5.3 Considering numbers of accepted SFCs

The models in Section 4.2 do not consider maximizing the acceptance ratio. There is no feasible solution if any requested SFCs are not accepted. Sometimes the system may receive too many SFCs and not all of them can be accepted and allocated. This work can consider maximizing the number of accepted SFCs as the objective of the proposed model. This work gives the following model:

$$\max \sum_{r \in R} \prod_{t \in T} \prod_{k \in K_r} \sum_{n \in N} x_{tn}^{rk} \tag{4.55a}$$

s.t. $(4.2) - (3.7), (3.2) - (3.5), (4.4), (3.9) - (3.10), (4.8), (4.9) - (4.33),$

$$\lambda \geq \lambda_{\min}, \tag{4.55b}$$

$$\sum_{n \in N} x_{tn}^{rk} \leq 1, \forall r \in R, k \in K_r, t \in T, \tag{4.55c}$$

$$\beta_r, \lambda \in [1, |T|], \forall r \in R, i \in T, j \in T_i, \tag{4.55d}$$

$$z_i^{jr} \in \{0,1\}, \forall r \in R, i \in T, j \in T_i. \tag{4.55e}$$

Equation (3.11) is replaced by (4.55c), which ensures that there is at most one primary function for each VNF. $\sum_{r \in R} \prod_{t \in T} \prod_{k \in K_r} \sum_{n \in N} x_{tn}^{rk}$ represents the number of accepted SFCs. $\lambda_{\min}$ is a given parameter, which represents the specified minimum SSCAT that the SFCs should satisfy.

4.6 Summary

This chapter proposed a primary and backup VNF placement model for improving the continuous available time of service function chains by avoiding the interruptions caused by unavailable nodes and function reallocations. This work formulated the proposed model as an ILP problem that maximizes the minimum number of the longest continuous available time slots in each SFC by considering deterministic availability schedules. This work extended the

proposed model to a network-aware one with considering a routing problem. Evaluation results showed that the proposed model improves the continuous available time of SFCs, compared with the baseline models in the examined cases. The network-aware proposed model reduces the latencies of services with keeping the same values of SCATs of services compared with the non-network-aware proposed model. This work introduced an algorithm that estimates the number of key unavailabilities at each time slot, which indicates the number of unavailable nodes that are required to be eliminated priorly. The number of key unavailabilities helps SPs to increase the service continuous available time with the least cost of failure recovery. This work analyzed the impact of different types of availability schedules on the proposed model. In the examined test cases, the proposed model provides a lower improvement on SSCATs if the positions of unavailabilities are more compact, and provides a higher improvement on SSCATs if the positions of unavailabilities are more sparse. This work developed and analyzed a heuristic algorithm to speed up the computation for the case that the problem size increases. The heuristic algorithm reduces 66.75% computation time with a cost of 1.57% performance degradation in terms of the objective value on average in the examined test cases. This work provided the discussion on dealing with multiple replicas of a function and their backups.

Chapter 5

Robust resource allocation model for uncertain availability schedules

This chapter proposes a VNF allocation to maximize the continuous available time of SFCs against service interruptions with considering uncertain availability schedules [96].

The remainder of the chapter is organized as follows. Section 5.1 describes the application scenario and the selection of metrics in the proposed model. Section 5.2 describes the proposed model. Section 5.3 introduces a heuristic algorithm for large size networks. Section 5.4 presents numerical results that show the performance of the proposed model in different cases. Section 5.5 discusses the influence of limited maintenance ability on the proposed model, and an extension of the proposed model for supporting multiple unavailable time periods on each node. Section 5.6 gives several directions to extend the proposed model. Section 5.7 summarizes the key points of this chapter.

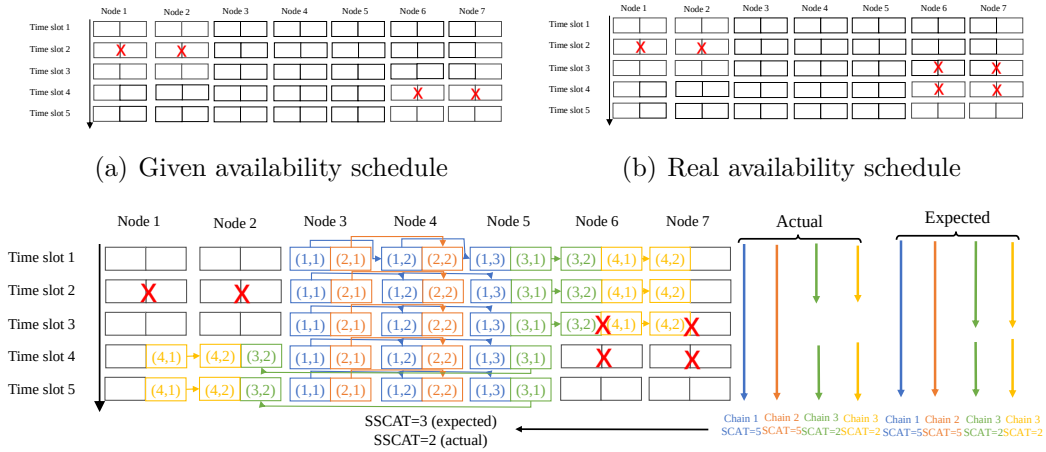
5.1 Motivation and applicability

5.1.1 Motivation

An availability schedule from a maintenance schedule [76] has the locations of unavailable VMs in the network during a period of time. The availability in the work is not a metric. It is a state or event of a node, i.e., the node is able or unable to allocate a VNF. The model is based on a motivation that the starting and stop times of a machine is given and the normal running time of services is what this work values. The model is not designed for the environment where the running time and the total availability evaluated by the probability is the objective are not cared about. If the operation time cannot be estimated, the proposed model cannot be applied, either. The allocation of VNFs before service running can suppress the interruptions caused by unavailabilities and reallocations with a given availability schedule during the given period of time so that SSCAT can be increased.

The scheduled maintenance may not be sometimes followed and is not usually exact on the time and locations. A source of availability schedules is predictive maintenance. A machine should be maintained after it continuously works [97]. The duration of a maintenance activity may vary according to some schedule-dependent factors, e.g., the starting time [98] and workload [99]. It is a common case that the duration of unavailability is longer than the duration in the plan. The unavailability can start a little earlier or later than the time given by availability schedules; for example, the constraint of starting time of maintenance activity is only a deadline in [98]. This work can rely on recovery mechanisms [85] in the unavailable periods of nodes. This work does not concern the details of the mechanisms. The model fully believes the estimated availability schedules provided by administrators. If a node is marked as available, this work considers that it is available. The availability is ensured by the other protection systems. However, the protection system cannot be fully believed in practice, which causes the uncertainty of the availability schedules.

This work cares about the exact running time of services instead of the mean time or probability of the normal running. As a condition of applying



(a) Given availability schedule

(b) Real availability schedule

(c) Expected and real SSCATs from an allocation based on given availability schedule

Figure 5.1: Demonstration of the impact of uncertain available schedule.

this model, the administrator or system must provide the time information of service starting and stopping.

A demonstration of the impact of availability schedule is shown in Fig. 5.1. An availability schedule including four unavailable periods is given in Fig. 5.1(a). Based on this availability schedule, an allocation of four SFCs is given in Fig. 5.1(c). The expected SSCAT is three. However, the given availability schedule is not exact. The unavailable periods on nodes 6 and 7 are longer than the estimation as shown in Fig. 5.1(b) so that the actual SSCAT of the calculated allocation decreases from three to two. It is necessary to design a robust model against the influences from these gaps. The challenge is how to design a robust model to maximize SSCAT against the uncertain availability schedules.

5.1.2 Applicability

The proposed model is designed for the initial allocation of VNFs in SFCs. SPs store the relative information including the availability schedules, VNFs, services, and network devices in the database. A computation element in the network, which is an independent node or a node with VNFs, calculates the allocation with these information. The allocation result is sent to each corresponding node. The nodes run the corresponding functions or the containers

in the nodes by pulling the corresponding images from repositories. The proposed model is designed to determine the locations of disturbance sensitive functions in the network, where the available schedule is uncertain and the number of available VMs for functions is limited.

In the real deployment of VNFs, the container or VM of the VNF must be placed to a node which is determined by an orchestration engine such as Kubernetes [9]. The proposed model works as a scoring mechanism in scheduler [100] in the engine. The placement is decided by the score provided by the model. Based on the information provided by administers including availability schedules and the information collected by the system including the nodes, the model calculates the suitable locations for each VNF and gives the highest score for the location.

The SPs usually have several replicas of a VNF in deployment for reliability and redundancy. For a VNF, the nodes can be classified into three states according to the state of the replica on the node: the replica is active, the image of the VNF is pulled but not active, and the image does not exist on the node. Without loss of generality, this work assumes that only one node where the replica is active for each VNF at the same time. The works give a direction of the assumption in Section 5.6. For stateless applications, the interruptions come from the download of VNF images and data redirection from the failed replicas to other replicas. For stateful applications, an additional interruption comes from the status data synchronization. The interruptions periods are estimated by the administrators and turned to the number of time slots used in the availability schedules.

5.2 Problem formulation

5.2.1 Formulation with deterministic unavailability periods

The models with given deterministic availability schedules were introduced in Chapters 3 and 4. They considered each unavailability marked in the availability schedule independently even on the same node.

This work assumes that the unavailabilities on a node can be continuous

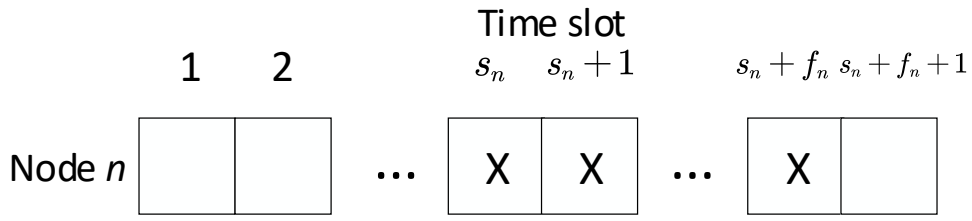


Figure 5.2: Unavailability period. "×" means unavailability.

from one time slot to another time slot, i.e., form a period that can be one or more time slots, which is called *unavailability period*. This work assumes that there is at most one unavailability period on each node in the given T. This work gives a future discussion on multiple unavailability periods on each node in Section 5.5.2.

This work uses a binary variable e_t^n to express elements in the availability schedule; if node $n \in N$ is unavailable at time slot t, $e_t^n = 1$, and otherwise 0. For node $n \in N$, the period starts at time slot $s_n \in T$ and lasts for another $f_n \in [0, |T| - s_n]$ continuous time slots, i.e., ends at time slot $s_n + f_n \in T$, as shown in Fig. 5.2. If there is no unavailability marked in availability schedule at node n, f_n is set to 0 and s_n is set to a positive integer value B_S which is larger than $|T|$. The values of $e_t^n, t \in T, n \in N$, can be obtained from s_n and f_n given by:

$$\begin{aligned} &\textbf{If} \quad s_n \le t \le s_n + f_n \quad \textbf{then} \\ &\qquad e_t^n = 1 \\ &\textbf{Else} \\ &\qquad e_t^n = 0 \end{aligned} \tag{5.1a}$$

$$e_t^n \in \{0, 1\}, \forall n \in N, t \in T. \tag{5.1b}$$

Based on the unavailability period, this work introduces the uncertainties of the beginning time slots and durations of unavailability periods in 5.2.2 and propose the model considering the uncertain availability schedule. The other definitions and formulations in this subsection are the same with those in Chapters 3 and 4 except for (5.2) and (5.3) because e_t^n is a variable in this work.

o_t^r can be expressed by:

$$o_t^r = \prod_{k \in K_r} \prod_{n \in N} ((x_{tn}^{rk} \odot x_{t-1,n}^{rk}) \wedge (1 - x_{tn}^{rk} \wedge e_t^n) \wedge (1 - x_{t-1,n}^{rk} \wedge e_{t-1}^n)), \forall r \in R,$$
$$t \in T \backslash \{1\}. \tag{5.2}$$

Here $\odot$ expresses exclusive NOR operation between two binary variables whose operations are: $1 \odot 1 = 1, 0 \odot 0 = 1, 1 \odot 0 = 0, 0 \odot 1 = 0$. $\wedge$ expresses the multiplication between two binary variables whose operations are: $1 \wedge 1 = 1, 0 \wedge 0 = 0, 1 \wedge 0 = 0, 0 \wedge 1 = 0$.

If it is necessary to avoid allocating functions to unavailable nodes, this work adds the following constraint:

$$x_{tn}^{rk} \wedge e_t^n = 0, \forall r \in R, t \in T, n \in N, k \in K_r. \tag{5.3}$$

According to the linearization process in Appendix A, (5.1) is linearized to (5.4a)-(5.4h), and (3.1) is linearized to (5.5a)-(5.5w) with some auxiliary variables.

$$t - s_n + \epsilon_1 \leq \eta_t^n \cdot B, \forall t \in T, n \in N, \tag{5.4a}$$
$$t - s_n + \epsilon_1 \geq (\eta_t^n - 1) \cdot B, \forall t \in T, n \in N, \tag{5.4b}$$
$$s_n + f_n - t + \epsilon_1 \leq \rho_t^n \cdot B, \forall t \in T, n \in N, \tag{5.4c}$$
$$s_n + f_n - t + \epsilon_1 \geq (\rho_t^n - 1) \cdot B, \forall t \in T, n \in N, \tag{5.4d}$$
$$e_t^n \leq \eta_t^n, \forall t \in T, n \in N, \tag{5.4e}$$
$$e_t^n \leq \rho_t^n, \forall t \in T, n \in N, \tag{5.4f}$$
$$e_t^n \geq \eta_t^n + \rho_t^n - 1, \forall t \in T, n \in N, \tag{5.4g}$$
$$e_t^n, \eta_t^n, \rho_t^n \in [0, 1], \forall t \in T, n \in N, \tag{5.4h}$$
$$\phi_{tn}^{rk} = 1 - x_{tn}^{rk} - x_{t-1,n}^{rk} + 2 \cdot h_{tn}^{rk}, \forall r \in R, t \in T \backslash \{1\}, k \in K_r, n \in N, \tag{5.5a}$$
$$h_{tn}^{rk} \leq x_{tn}^{rk}, \forall r \in R, t \in T \backslash \{1\}, k \in K_r, n \in N, \tag{5.5b}$$
$$h_{tn}^{rk} \leq x_{t-1,n}^{rk}, \forall r \in R, t \in T \backslash \{1\}, k \in K_r, n \in N, \tag{5.5c}$$
$$h_{tn}^{rk} \geq x_{tn}^{rk} + x_{t-1,n}^{rk} - 1, \forall r \in R, t \in T \backslash \{1\}, k \in K_r, n \in N, \tag{5.5d}$$
$$\alpha_{tn}^{rk} \leq x_{tn}^{rk}, \forall r \in R, t \in T, k \in K_r, n \in N, \tag{5.5e}$$
$$\alpha_{tn}^{rk} \leq e_t^n, \forall r \in R, t \in T, k \in K_r, n \in N, \tag{5.5f}$$
$$\alpha_{tn}^{rk} \geq x_{tn}^{rk} + e_t^n - 1, \forall r \in R, t \in T, k \in K_r, n \in N, \tag{5.5g}$$

$$\theta_{tn}^{rk} \leq \phi_{tn}^{rk}, \forall r \in R, t \in T\backslash\{1\}, k \in K_r, n \in N, \tag{5.5h}$$

$$\theta_{tn}^{rk} \leq 1 - \alpha_{tn}^{rk}, \forall r \in R, t \in T\backslash\{1\}, k \in K_r, n \in N, \tag{5.5i}$$

$$\theta_{tn}^{rk} \geq \phi_{tn}^{rk} - \alpha_{tn}^{rk}, \forall r \in R, t \in T\backslash\{1\}, k \in K_r, n \in N, \tag{5.5j}$$

$$\pi_{tn}^{rk} \leq \theta_{tn}^{rk}, \forall r \in R, t \in T\backslash\{1\}, k \in K_r, n \in N, \tag{5.5k}$$

$$\pi_{tn}^{rk} \leq 1 - \alpha_{t-1,n}^{rk}, \forall r \in R, t \in T\backslash\{1\}, k \in K_r, n \in N, \tag{5.5l}$$

$$\pi_{tn}^{rk} \geq \theta_{tn}^{rk} - \alpha_{t-1,n}^{rk}, \forall r \in R, t \in T\backslash\{1\}, k \in K_r, n \in N, \tag{5.5m}$$

$$w_t^{rk} \leq \pi_{tn}^{rk}, \forall r \in R, t \in T\backslash\{1\}, k \in K_r, n \in N, \tag{5.5n}$$

$$w_t^{rk} \geq \sum_{n \in N} \pi_{tn}^{rk} - |N| + 1, \forall r \in R, t \in T\backslash\{1\}, k \in K_r, \tag{5.5o}$$

$$o_t^r \leq w_t^{rk}, \forall r \in R, t \in T\backslash\{1\}, k \in K_r, \tag{5.5p}$$

$$o_t^r \geq \sum_{k \in K_r} w_t^{rk} - |K_r| + 1, \forall r \in R, t \in T\backslash\{1\}, \tag{5.5q}$$

$$o_1^r = 0, \forall r \in R, \tag{5.5r}$$

$$o_t^r \in \{0,1\}, \forall r \in R, t \in T, \tag{5.5s}$$

$$y_j^r \in \{0,1\}, \forall r \in R, j \in T, \tag{5.5t}$$

$$w_t^{rk} \in \{0,1\}, \forall r \in R, k \in K_r, t \in T\backslash\{1\}, \tag{5.5u}$$

$$\phi_{tn}^{rk}, h_{tn}^{rk}, \theta_{tn}^{rk}, \pi_{tn}^{rk} \in \{0,1\}, \forall r \in R, k \in K_r, t \in T\backslash\{1\}, n \in N, \tag{5.5v}$$

$$\alpha_{tn}^{rk}, x_{tn}^{rk}, \in \{0,1\}, \forall r \in R, k \in K_r, t \in T, n \in N. \tag{5.5w}$$

In the above equations, ϵ_1, ϵ_2, and B are given parameters, where $0 < \epsilon_1 < \frac{1}{s_n + f_n}$, $0 < \epsilon_2 < 1$, and B is larger than $s_n + f_n + 1 - t, t - s_n + \epsilon_2, n \in N, t \in T$, and 1. The minimum of B can be taken to be $|T| + 2$.

In summary, the following model is given for VNF allocation with considering the deterministic availability schedule:

$$\begin{aligned}
&\max\ \lambda + \epsilon \sum_{r \in R} \beta_r \\
&\text{s.t.}\ \ (3.2) - (3.5), (3.7), (3.9) - (3.11), (3.13) - (5.5\text{w}), (5.4\text{a}) - (5.5\text{w}), \\
&\qquad \beta_r, \lambda \in [1, |T|], \forall r \in R, i \in T, j \in T_i, \\
&\qquad z_i^{jr} \in \{0,1\}, \forall r \in R, i \in T, j \in T_i.
\end{aligned}$$

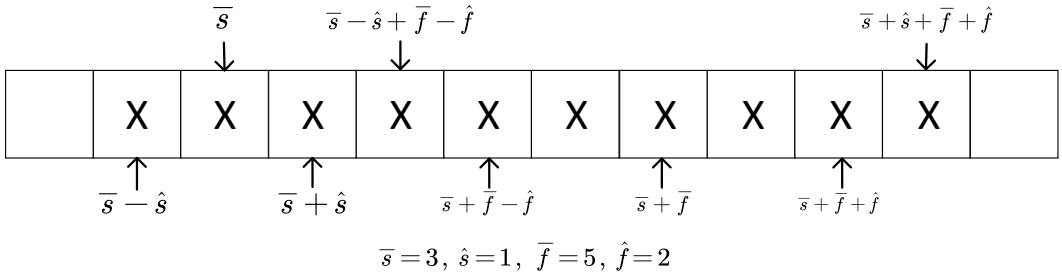


Figure 5.3: Type of uncertainty. "×" means unavailability.

5.2.2 Formulation with uncertain unavailability periods

In Section 5.2.1, s_n and f_n are deterministic. This subsection considers the uncertainty caused by an uncertain beginning time slot and an uncertain duration of each node, as shown in Fig. 5.3. The uncertainty set of s_n is symmetric about $\bar{s_n}$ and that of f_n is symmetric about $\bar{f_n}$, i.e., $s_n \in [\bar{s_n} - \hat{s_n}, \bar{s_n} + \hat{s_n}]$ and $f_n \in [\bar{f_n} - \hat{f_n}, \bar{f_n} + \hat{f_n}]$, respectively, where $s_n, s_n + f_n \in T$. If there is no unavailability marked in availability schedule on node $n \in N$, the corresponding $\bar{s_n}$ is set to B_S; $\hat{s_n}$ $\bar{f_n}$, and $\hat{f_n}$ are set to 0.

In (5.4a)-(5.4d), s_n and f_n are uncertain. This work models an robust optimization problem which can control the degree of robustness against uncertainties and conservation of solutions. This work defines Γ_n^S as the degree of robustness of $s_n, n \in N$ and $\Gamma_n^F \in [-1, 1]$ as the degree of robustness of $f_n, n \in N$. The larger Γ_n^S or Γ_n^F is, the higher the level of robustness is.

Γ_n^S controls the size of uncertainty set of beginning time slots $\mathcal{P}_n$ on node n. The larger Γ_n^S is, the more the considered possible beginning time slots of unavailability period on node n are. This work chooses $\lfloor \Gamma_n^S \cdot (2 \cdot \hat{s_n} + 1) \rfloor$ different beginning time slots from $[\bar{s_n} - \hat{s_n}, \bar{s_n} + \hat{s_n}]$ and forms a set which is an element in $\mathcal{P}_n$. $\mathcal{P}_n$ contains all possible choices. The size of $\mathcal{P}_n$ is $|\mathcal{P}_n| = \begin{pmatrix} 2 \cdot \hat{s_n} + 1 \\ \lfloor \Gamma_n^S \cdot (2 \cdot \hat{s_n} + 1) \rfloor \end{pmatrix}$. Since the size of each element in $\mathcal{P}_n$ is at least 1, i.e., $\lfloor \Gamma_n^S \cdot (2 \cdot \hat{s_n} + 1) \rfloor \geq 1$, $\Gamma_n^S \in [\frac{1}{2 \cdot \hat{s_n} + 1}, 1]$. The uncertainty set for all nodes is denoted by $\mathcal{P} = \{\mathcal{P}_1, \mathcal{P}_2, \cdots, \mathcal{P}_{|N|}\}$. One element in $\mathcal{P}$ is a combination of one element in each $\mathcal{P}_n$. The size of $\mathcal{P}$ is $|\mathcal{P}| = |\mathcal{P}_1| \cdot |\mathcal{P}_2| \cdot \cdots \cdot |\mathcal{P}_{|N|}|$. Let p denote an

element in $\mathcal{P}$; p_n denote the selected set from $\mathcal{P}_n$; p_n^q denote the qth element in p_n.

Γ_n^{F} controls the considered length of the duration of unavailability period on node n, which is $\bar{f}_n + \hat{f}_n \cdot \lfloor \Gamma_n^{\mathrm{F}} \rfloor$.

To deal the uncertain unavailability periods, this work develops an MILP formulation in terms of Γ_n^{S} and Γ_n^{F}. For the unavailability period on each node n under the uncertainty of $\mathcal{P}$, this work considers the worst case. The objective of the considered problem is presented by:

$$\max \min_{p \in \mathcal{P}} \{\lambda^p + \epsilon \sum_{r \in R} \beta_r^p\}. \tag{5.6}$$

Here, λ^p means the value of λ under the uncertainty set $p \in \mathcal{P}$. β_r^p means the value of β_r under the uncertainty set $p \in \mathcal{P}$. The selected set of beginning time slots p_n contains one or more time slot. Equations (5.4a)-(5.4h) are transformed to:

$$t - p_n^q + \epsilon_1 \leq \eta_{tn}^q \cdot B, \forall t \in T, n \in N, q \in [1, |p_n|], \tag{5.7a}$$

$$t - p_n^q + \epsilon_1 \geq (\eta_{tn}^q - 1) \cdot B, \forall t \in T, n \in N, q \in [1, |p_n|], \tag{5.7b}$$

$$p_n^q + \bar{f}_n + \Gamma_n^{\mathrm{F}} \cdot \hat{f}_n - t + \epsilon_1 \leq \rho_{tn}^q \cdot B, \forall t \in T, n \in N, q \in [1, |p_n|], \tag{5.7c}$$

$$p_n^q + \bar{f}_n + \Gamma_n^{\mathrm{F}} \cdot \hat{f}_n - t + \epsilon_1 \geq (\rho_{tn}^q - 1) \cdot B, \forall t \in T, n \in N, q \in [1, |p_n|], \tag{5.7d}$$

$$\tau_{tn}^q \leq \eta_{tn}^q, \forall t \in T, n \in N, q \in [1, |p_n|], \tag{5.7e}$$

$$\tau_{tn}^q \leq \rho_{tn}^q, \forall t \in T, n \in N, q \in [1, |p_n|], \tag{5.7f}$$

$$\tau_{tn}^q \geq \eta_{tn}^q + \rho_{tn}^q - 1, \forall t \in T, n \in N, q \in [1, |p_n|], \tag{5.7g}$$

$$\tau_{tn}^q, \eta_{tn}^q, \rho_{tn}^q \in [0, 1], \forall t \in T, n \in N, q \in [1, |p_n|]. \tag{5.7h}$$

The above equations use τ_{tn}^q to replace e_t^n in (5.4a)-(5.4h). If node n is unavailable at time slot t when the beginning time slot q is considered, τ_{tn}^q is set to 1, and otherwise 0. $e_t^n \in [0, 1], t \in T, n \in N$, is expressed by:

$$e_t^n = \vee_{q \in [1, |p_n|]} \tau_{tn}^q, \forall t, \in T, n \in N. \tag{5.8}$$

Here $\vee$ means OR operation between two binary variables whose operations are: $0 \vee 0 = 0, 0 \vee 1 = 1, 1 \vee 0 = 1, 1 \vee 1 = 1$. According to the linearization

process in Appendix A, (5.8) can be linearized to:

$$e_t^n \geq \frac{1}{|p_n|} \cdot \sum_{q \in [1,|p_n|]} \tau_{tn}^q, \forall t, \in T, n \in N, \tag{5.9a}$$

$$e_t^n \leq \sum_{q \in [1,|p_n|]} \tau_{tn}^q, \forall t, \in T, n \in N. \tag{5.9b}$$

According to the linearization process in Appendix A, the objective function (5.6) can be linearized to:

$$\max \Lambda \tag{5.10a}$$

$$\text{s.t. } \Lambda \leq \lambda^p + \epsilon \sum_{r \in R} \beta_r^p + (1 - \delta_{\mathrm{o}}^p) \cdot B_{\mathrm{O}}, \forall p \in \mathcal{P}, \tag{5.10b}$$

$$\Lambda \geq \lambda^p + \epsilon \sum_{r \in R} \beta_r^p - (1 - \delta_{\mathrm{o}}^p) \cdot B_{\mathrm{O}}, \forall p \in \mathcal{P}, \tag{5.10c}$$

$$\lambda^p + \epsilon \sum_{r \in R} \beta_r^p \leq (1 - \delta_{\mathrm{o}}^p) \cdot B_{\mathrm{O}} + \lambda^{p'} + \epsilon \sum_{r \in R} \beta_r^{p'},$$

$$\forall p \in \mathcal{P}, p' \in \mathcal{P} \backslash p, \tag{5.10d}$$

$$\sum_{p \in \mathcal{P}} \delta_{\mathrm{o}}^p = 1, \tag{5.10e}$$

$$\Lambda \leq \lambda^p + \epsilon \sum_{r \in R} \beta_r^p, \forall p \in \mathcal{P}, \tag{5.10f}$$

$$\delta_{\mathrm{o}}^p \in [0, 1], \forall p \in \mathcal{P}. \tag{5.10g}$$

Here, B_{O} is a given integer which is larger than $\lambda^p + \epsilon \sum_{r \in R} \beta_r^p$. The minimum of B_{O} can be taken to be $|T| + 2$.

To calculate the objective values in element $p \in \mathcal{P}$, this work performs the following transformations. Equations (3.2)-(3.5) in Chapter 3 are transformed to (5.11a)-(5.11d) by replacing o_t^r, z_i^{jr}, y_j^r to $o_t^{rp}, z_i^{jrp}, y_j^{rp}$, respectively. Equation (3.7) in Chapter 3 is transformed to (5.12a) by replacing λ, β_r to λ^p, β_r^p, respectively. Equations (3.9)-(3.11) in Chapter 3 are transformed to (5.13a)-(5.13c) by replacing x_{tn}^{rk} to x_{tn}^{rkp}, respectively. Equations (3.13)-(5.5w) are transformed to (5.14a)-(5.15w) by replacing $\beta_r, y_j^r, \delta_j^r, \phi_{tn}^{rk}, x_{tn}^{rk}, h_{tn}^{rk}, \alpha_{tn}^{rk}, \theta_{tn}^{rk}, \pi_{tn}^{rk}$, w_t^{rk}, o_t^r to $\beta_r^p, y_j^{rp}, \delta_j^{rp}, \phi_{tn}^{rkp}, x_{tn}^{rkp}, h_{tn}^{rkp}, \alpha_{tn}^{rkp}, \theta_{tn}^{rkp}, \pi_{tn}^{rkp}, w_t^{rkp}, o_t^{rp}$, respectively. Equations (5.7a)-(5.7h) are transformed to (5.16a)-(5.16h) by replacing e_t^n, η_{tn}^q, ρ_{tn}^q, τ_{tn}^q to $e_t^{np}, \eta_{tn}^{qp}, \rho_{tn}^{qp}, \tau_{tn}^{qp}$, respectively. Equations (5.9a)-(5.9b) are transformed to (5.17a)-(5.17b) by replacing e_t^n to e_t^{np}, respectively.

$$(\sum_{t=i}^{i+j-1} o_t^{rp}) - j + 1 \leq z_i^{jrp}, \forall i \in T, j \in T_i, r \in R, p \in \mathcal{P}, \tag{5.11a}$$

$$z_i^{jrp} \leq \frac{\sum_{t=i}^{i+j-1} o_t^{rp}}{j}, \forall i \in T, j \in T_i, r \in R, p \in \mathcal{P}, \tag{5.11b}$$

$$z_i^{jrp} \leq y_j^{rp}, \forall i \in T, j \in T_i, r \in R, p \in \mathcal{P}, \tag{5.11c}$$

$$y_j^{rp} \leq \sum_{t=1}^{|T|-j+1} z_t^{jrp}, \forall j \in T, r \in R, p \in \mathcal{P}, \tag{5.11d}$$

$$\lambda^p \leq \beta_r^p, \forall r \in R, p \in \mathcal{P}, \tag{5.12a}$$

$$\sum_{r \in R} \sum_{k \in K_r} x_{tn}^{rkp} \leq c_{nt}^s, \forall n \in N, t \in T, p \in \mathcal{P}, s \in S, \tag{5.13a}$$

$$\sum_{k \in K_r} x_{tn}^{rkp} \leq 1, \forall r \in R, n \in N, t \in T, p \in \mathcal{P}, \tag{5.13b}$$

$$\sum_{n \in N} x_{tn}^{rkp} = 1, \forall r \in R, k \in K_r, t \in T, p \in \mathcal{P}, \tag{5.13c}$$

$$\beta_r^p - 1 \leq j y_j^{rp} + \left(1 - \delta_j^{rp}\right) \cdot B, \forall j \in T, r \in R, p \in \mathcal{P}, \tag{5.14a}$$

$$\beta_r^p - 1 \geq j y_j^{rp} - \left(1 - \delta_j^{rp}\right) \cdot B, \forall j \in T, r \in R, p \in \mathcal{P}, \tag{5.14b}$$

$$j y_j^{rp} \geq \left(\delta_j^{rp} - 1\right) \cdot B + y_{j'}^{rp}, \forall j \in T, j' \in T \setminus \{j\}, r \in R, p \in \mathcal{P}, \tag{5.14c}$$

$$\sum_{j \in T} \delta_j^{rp} = 1, \forall r \in R, p \in \mathcal{P}, \tag{5.14d}$$

$$\beta_r^p - 1 \geq j y_j^{rp}, \forall j \in T, r \in R, p \in \mathcal{P}, \tag{5.14e}$$

$$\delta_j^{rp} \in \{0,1\}, \forall j \in T, r \in R, p \in \mathcal{P}, \tag{5.14f}$$

$$\phi_{tn}^{rkp} = 1 - x_{tn}^{rkp} - x_{t-1,n}^{rkp} + 2 \cdot h_{tn}^{rkp}, \forall r \in R, t \in T \setminus \{1\}, k \in K_r, n \in N, p \in \mathcal{P}, \tag{5.15a}$$

$$h_{tn}^{rkp} \leq x_{tn}^{rkp}, \forall r \in R, t \in T \setminus \{1\}, k \in K_r, n \in N, \tag{5.15b}$$

$$h_{tn}^{rkp} \leq x_{t-1,n}^{rkp}, \forall r \in R, t \in T \setminus \{1\}, k \in K_r, n \in N, p \in \mathcal{P}, \tag{5.15c}$$

$$h_{tn}^{rkp} \geq x_{tn}^{rkp} + x_{t-1,n}^{rkp} - 1, \forall r \in R, t \in T \setminus \{1\}, k \in K_r, n \in N, p \in \mathcal{P}, \tag{5.15d}$$

$$\alpha_{tn}^{rkp} \leq x_{tn}^{rkp}, \forall r \in R, t \in T, k \in K_r, n \in N, p \in \mathcal{P}, \tag{5.15e}$$

$$\alpha_{tn}^{rkp} \leq e_t^{np}, \forall r \in R, t \in T, k \in K_r, n \in N, p \in \mathcal{P}, \tag{5.15f}$$

$$\alpha_{tn}^{rkp} \geq x_{tn}^{rkp} + e_t^{np} - 1, \forall r \in R, t \in T, k \in K_r, n \in N, p \in \mathcal{P}, \tag{5.15g}$$

$$\theta_{tn}^{rkp} \leq \phi_{tn}^{rkp}, \forall r \in R, t \in T \backslash \{1\}, k \in K_r, n \in N, p \in \mathcal{P}, \tag{5.15h}$$

$$\theta_{tn}^{rkp} \leq 1 - \alpha_{tn}^{rkp}, \forall r \in R, t \in T \backslash \{1\}, k \in K_r, n \in N, p \in \mathcal{P}, \tag{5.15i}$$

$$\theta_{tn}^{rkp} \geq \phi_{tn}^{rkp} - \alpha_{tn}^{rkp}, \forall r \in R, t \in T \backslash \{1\}, k \in K_r, n \in N, p \in \mathcal{P}, \tag{5.15j}$$

$$\pi_{tn}^{rkp} \leq \theta_{tn}^{rkp}, \forall r \in R, t \in T \backslash \{1\}, k \in K_r, n \in N, p \in \mathcal{P}, \tag{5.15k}$$

$$\pi_{tn}^{rkp} \leq 1 - \alpha_{t-1,n}^{rkp}, \forall r \in R, t \in T \backslash \{1\}, k \in K_r, n \in N, p \in \mathcal{P}, \tag{5.15l}$$

$$\pi_{tn}^{rkp} \geq \theta_{tn}^{rkp} - \alpha_{t-1,n}^{rkp}, \forall r \in R, t \in T \backslash \{1\}, k \in K_r, n \in N, p \in \mathcal{P}, \tag{5.15m}$$

$$w_t^{rkp} \leq \pi_{tn}^{rkp}, \forall r \in R, t \in T \backslash \{1\}, k \in K_r, n \in N, p \in \mathcal{P}, \tag{5.15n}$$

$$w_t^{rkp} \geq \sum_{n \in N} \pi_{tn}^{rkp} - |N| + 1, \forall r \in R, t \in T \backslash \{1\}, k \in K_r, p \in \mathcal{P}, \tag{5.15o}$$

$$o_t^{rp} \leq w_t^{rkp}, \forall r \in R, t \in T \backslash \{1\}, k \in K_r, p \in \mathcal{P}, \tag{5.15p}$$

$$o_t^{rp} \geq \sum_{k \in K_r} w_t^{rkp} - |K_r| + 1, \forall r \in R, t \in T \backslash \{1\}, p \in \mathcal{P}, \tag{5.15q}$$

$$o_1^{rp} = 0, \forall r \in R, p \in \mathcal{P}, \tag{5.15r}$$

$$o_t^{rp} \in \{0,1\}, \forall r \in R, t \in T, p \in \mathcal{P}, \tag{5.15s}$$

$$y_j^{rp} \in \{0,1\}, \forall r \in R, j \in T, p \in \mathcal{P}, \tag{5.15t}$$

$$w_t^{rkp} \in \{0,1\}, \forall r \in R, k \in K_r, t \in T \backslash \{1\}, p \in \mathcal{P}, \tag{5.15u}$$

$$\phi_{tn}^{rkp}, h_{tn}^{rkp}, \theta_{tn}^{rkp}, \pi_{tn}^{rkp} \in \{0,1\}, \forall r \in R, k \in K_r, t \in T \backslash \{1\}, n \in N, p \in \mathcal{P}, \tag{5.15v}$$

$$\alpha_{tn}^{rkp}, x_{tn}^{rkp}, \in \{0,1\}, \forall r \in R, k \in K_r, t \in T, n \in N, p \in \mathcal{P}, \tag{5.15w}$$

$$t - p_n^{qp} + \epsilon_1 \leq \eta_{tn}^{qp} \cdot B, \forall t \in T, n \in N, q \in [1, |p_n|], p \in \mathcal{P} \tag{5.16a}$$

$$t - p_n^{qp} + \epsilon_1 \geq (\eta_{tn}^{qp} - 1) \cdot B, \forall t \in T, n \in N, q \in [1, |p_n|], p \in \mathcal{P} \tag{5.16b}$$

$$p_n^{qp} + \bar{f}_n + \Gamma_n^{\mathrm{F}} \cdot \hat{f}_n - t + \epsilon_1 \leq \rho_{tn}^{qp} \cdot B, \forall t \in T, n \in N, \quad q \in [1, |p_n|], p \in \mathcal{P} \tag{5.16c}$$

$$p_n^{qp} + \bar{f}_n + \Gamma_n^{\mathrm{F}} \cdot \hat{f}_n - t + \epsilon_1 \geq (\rho_{tn}^{qp} - 1) \cdot B, \forall t \in T, n \in N, \quad q \in [1, |p_n|], p \in \mathcal{P} \tag{5.16d}$$

$$\tau_{tn}^{qp} \leq \eta_{tn}^{qp}, \forall t \in T, n \in N, q \in [1, |p_n|], p \in \mathcal{P} \tag{5.16e}$$

$$\tau_{tn}^{qp} \leq \rho_{tn}^{qp}, \forall t \in T, n \in N, q \in [1, |p_n|], p \in \mathcal{P} \tag{5.16f}$$

$$\tau_{tn}^{qp} \geq \eta_{tn}^{qp} + \rho_{tn}^{qp} - 1, \forall t \in T, n \in N, q \in [1, |p_n|], p \in \mathcal{P} \tag{5.16g}$$

$$e_t^{np}, \eta_{tn}^{qp}, \rho_{tn}^{qp}, \tau_{tn}^{qp} \in [0, 1], \forall t \in T, n \in N, q \in [1, |p_n|], \quad p \in \mathcal{P} \tag{5.16h}$$

$$e_t^{np} \geq \frac{1}{|p_n|} \cdot \sum_{q \in [1, |p_n|]} \tau_{tn}^{qp}, \forall t, \in T, n \in N, p \in \mathcal{P}, \tag{5.17a}$$

$$e_t^{np} \leq \sum_{q \in [1, |p_n|]} \tau_{tn}^{qp}, \forall t, \in T, n \in N, p \in \mathcal{P}. \tag{5.17b}$$

From (5.11a) to (5.17b), a variable with the subscript p means the value of this variable under the uncertainty set $p \in \mathcal{P}$.

In summary, the robust optimization problem is given by:

$$\max \Lambda \tag{5.18a}$$

$$\text{s.t. } (5.10\text{b}) - (5.10\text{g}), (5.11\text{a}) - (5.17\text{b}), \tag{5.18b}$$

$$\Lambda \in [1, |T|], \tag{5.18c}$$

$$\beta_r^p, \lambda^p \in [1, |T|], \forall r \in R, i \in T, j \in T_i, p \in \mathcal{P}, \tag{5.18d}$$

$$z_i^{jrp} \in \{0,1\}, \forall r \in R, i \in T, j \in T_i, p \in \mathcal{P}. \tag{5.18e}$$

5.3 Heuristic algorithm

As the size of the ILP problem in Section 5.2 increases, the problem becomes difficult to solve in practical time. A feasible solution may not be obtained within admissible computational time, which can be specified by SPs. This work introduces a heuristic algorithm based on the genetic algorithm [81]. To accelerate the computation process, this work introduces an extension of this algorithm by using CPU and GPU acceleration [101].

5.3.1 Framework

The framework of this heuristic algorithm is shown in Algorithm 5.1. The set of possible beginning time slots and the length of considered unavailability period are calculated. The availability schedule is given by using (5.1) under a combination of possible beginning time set p in line 5. A set of initial feasible solutions whose size is *IPN* is given by Algorithm 5.2 in lines 6–7. From line 8 to line 31, the heuristic algorithm enters a loop that has *MG* cycles. In each cycle, the heuristic algorithm explores new feasible solutions by performing internal crossover (see function *cross_in* in Algorithm 5.3), external crossover (see function *cross_out* in Algorithm 5.3), and mutation (see Algorithm 5.4) according to three probabilities, *ICP*, *ECP*, and *MP*, respectively. Newly generated solutions at lines 12, 17, and 22 are stored in the new feasible solution set S_{n} once. They are added to the feasible solution set at a time in line 25. The heuristic algorithm calculates the fitness for each solution (see Algorithm 5.5). The heuristic algorithm finds the solution with the highest fitness score and stores it. Finally, if the size of the feasible solution set exceeds *ULPN*, the heuristic algorithm chooses *ULPN* feasible solutions as a new set of feasible solutions according to the roulette gambler (see Algorithm 5.6).

Algorithm 5.1 Framework

Input: N, T, R, K_r, $c_{nt}^s \in C$, $\bar{s_n}, \hat{s_n}, \bar{f_n}, \hat{f_n}, \Gamma_n^{\mathrm{S}}, \Gamma_n^{\mathrm{F}}$, *IPN, MG, ECP, ICP, MP, ULPN*

Output: allocation for all functions

1: $f_n \leftarrow \bar{f_n} + \Gamma_n^{\mathrm{F}} \cdot \hat{f_n}$
2: Calculate the possible choices of s_n by $\Gamma_n^{\mathrm{S}}, \bar{s_n}, \hat{s_n}$ and stored in $\mathcal{P}_n$
3: Calculate different combinations of $\mathcal{P}_n$ on all nodes and store them in set P
4: **for** $p \in \mathcal{P}$ **do**
5: Calculate the value of e_t^n according to (5.1) by using p and f_n
6: Define S^p as the feasible solution set for the possible uncertainty set p
7: $S^p \leftarrow$ Generate set of initial feasible solutions by using function *init_chromos* in Algorithm 3.2
8: **for** $step = 1 \rightarrow MG$ **do**
9: Define S_{n}^p as the new feasible solution set

10: **for** each solution in S^p **do**
11: **if** a random number in $[0, 1]$ >1- *ICN* **then**
12: S^p_{n} ←Generate a non-redundant and mutant solution by using function *cross_in* in Algorithm 5.3 whose inputs are the selected solution in S^p and random time slot t
13: **end if**
14: **end for**
15: **for** each solution in S^p except for the first one **do**
16: **if** a random number $[0, 1]$ >1- *ECP* **then**
17: S^p_{n} ←Generate a non-redundant and mutant solution by using function *cross_out* in Algorithm 5.3 whose inputs are the selected solution and its previous solution in S
18: **end if**
19: **end for**
20: **for** each solution in S^p **do**
21: **if** a random number $[0, 1]$ >1- *MP* **then**
22: S^p_{n} ←Generate a non-redundant and mutant solution by using function *mutation* in Algorithm 5.4 whose input is the selected solution in S^p
23: **end if**
24: **end for**
25: Integrate S^p_{n} into S
26: Calculate the fitness score of the solutions in S^p by using function *calc_fin_ness* in Algorithm 5.5
27: Store the solution with the highest fitness score
28: **if** size of $S^p > ULPN$ **then**
29: Reduce the size of the set to *UP* by using function *roulette_gambler* in Algorithm 3.6
30: **end if**
31: **end for**
32: Output the solution which has the smallest objective value among $S^p, p \in \mathcal{P}$
33: **end for**

5.3.2 Initial solution generation

The heuristic algorithm generates a set of initial feasible solutions by using Algorithm 5.2. Based on this set, more feasible solutions can be generated by Algorithms 5.3 and 5.4.

In the heuristic algorithm, each solution is a three-dimensional matrix. The first dimension represents time slots, the second one represents requests and the third one represents functions. The value of an element whose location is (t, r, k) is the allocation of the kth function of request r at time slot t, which belongs to N.

Algorithm 5.2 Initial solution

```
1: function INIT_CHROMOS(E)
2:     Set of initial solutions s ← φ
3:     Sort requests in R in a non-increasing order of K_r
4:     for each time slot in T do
5:         Sort nodes in N in a non-increasing order of time from time slot t
       to a time slot in which a node becomes unavailable. If the above values
       are the same for some n, sort them in a non-increasing order of time from
       time slot t to a time slot in which a node becomes unavailable secondly.
       If the above values are the same for some n, sort them in a non-increasing
       order of time from time slot t to the last time slot in which a node becomes
       unavailable.
6:         for r ∈ R do
7:             for f = 1 → K_r do
8:                 for n ∈ N do
9:                     if used capacity of n is less than c_n^t AND any other func-
       tions in r were not allocated to n then
10:                        Allocate the fth function in SFC r to n
11:                        Break
12:                    else
13:                        Continue
14:                    end if
15:                end for
16:            end for
```

17: **end for**
18: Store the allocation to s
19: **end for**
20: Duplicate a solution iteratively until the number of solutions in s becomes IPN
21: **return** s
22: **end function**

Algorithm 5.2 reorders set R according to the corresponding K_r from long to short first (line 3). Then, it performs function allocation one by one (lines 4–19). At each time slot, the heuristic algorithm reorders set N according to the occurrence of unavailabilities from late to early (line 5). Then the heuristic algorithm allocates the functions to physical nodes according to these new orders (lines 9–13). Finally, the heuristic algorithm duplicates a solution iteratively until the number of solutions becomes IPN (line 20).

5.3.3 New solution generation

There are three methods for generating new solutions in the heuristic algorithm.

The internal crossover, function *cross_in* in Algorithm 5.3, crosses adjacent time slots in the same solution. The aim of *cross_in* is to suppress the reallocations of VNFs between adjacent time slots.

The external crossover, function *cross_out* in Algorithm 5.3, crosses the same time slot between two solutions in the feasible solution set. A new solution is generated by modifying the VNF allocation in a randomly selected time slot of one solution based on that of another solution.

Algorithm 5.3 Crossover

1: **function** CROSS_IN($s \leftarrow$ the selected solution , $t \leftarrow$ the random time)
2: $s[t] \leftarrow s[t+1]$
3: **return** s
4: **end function**
5: **function** CROSS_OUT(s_1, s_2)
6: $location \leftarrow$ a random integer in $[1, |T|]$
7: $s_c \leftarrow s_1$

```
    s_c[location] ← s_2[location]
    return s_c
end function
```

The other function for generating new solutions is function *mutation* in Algorithm 5.4. *init_chromos* function in Algorithm 5.2 generates the initial solution set based on the length of each SFC. On the other hand, *mutation* function generates a new solution based on the SCAT of each SFC.

Algorithm 5.4 Mutation

```
function MUTATION(s ← the selected solution, E)
    Calculate the SCAT for all SFCs in solution s
    Sort requests in R in a weighted randomized order. The larger the
SCAT, the higher the order of the corresponding r.
    Sort nodes in N in a weighted randomized order. The larger the time
from the first time slot to a time slot where a node becomes unavailable,
the higher the order of the corresponding n.
    for r ∈ R do
        for f = 1 → K_r of request r do
            for n ∈ N do
                if used capacity of n is less than c_n^t AND any other functions
in r were not allocated to n then
                    Allocate the fth function in SFC r to n
                    Break
                else
                    Continue
                end if
            end for
        end for
    end for
    return new solution
end function
```

5.3.4 Calculation of fitness

The algorithm computes the fitness score for each solution by using the objective function.

Algorithm 5.5 Fitness calculation

```
1: function CALC_FIN_NESS(s ← the selected solution, E)
2:     Calculate the SCAT for all SFCs in solution s
3:     return min(SCAT)+sum(SCAT)/(|T| × |R|)
4: end function
```

5.3.5 Choice of solutions

The heuristic algorithm uses roulette wheel selection to create a new feasible solution set by choosing UP solutions from the feasible solution set.

In the *roulette_gambler* and *choice* functions in Algorithm 5.6, input $chroms$ is the set of solutions and fit_pros is the set of the fitness scores of the corresponding solutions in $chroms$.

Algorithm 5.6 Choice

```
1: function ROULETTE_GAMBLER(fit_pros, chroms)
2:     pick ← a random number in [0, 1]
3:     for j = 1 → |chroms| do
4:         pick ← pick − fit_pros[j]/sum(fit_pros)
5:         if pick ≤ 0 then
6:             return j
7:         end if
8:     end for
9:     return |chroms| − 1
10: end function
11: function CHOICE(chroms, fit_pros)
12:     choice_gens ← ϕ
13:     for i = 1 → min(|chroms|, UP) do
14:         j ← ROULETTE_GAMBLER(fit_pros, chroms)
15:         append chroms[j] to choice_gens
16:     end for
17:     return choice_gens
```

18: **end function**

5.3.6 Parallelization

In lines 4–33 of Algorithm 5.1, the algorithm calculates the allocation under different set P. These calculations can be parallelized. The algorithm initializes $|P|$ threads and performs the calculations at the same time. $|P|$ results are obtained. The final result is the smallest one among these results.

In order to further reduce the calculation time, this work can also make the parallelization of the functions called in the heuristic algorithm. The loops in lines 10–24 of Algorithm 5.1 can be parallelized. Each thread contains the cross and mutation of one solution in S^p. The new generated solutions are collected and merged into set S in line 25 of Algorithm 5.1. However, the number of solutions required to be processed in parallel is greater than the number of cores in a CPU. If the number of threads is small, the time consumed by thread switching slows down the computation time. A method to reduce the overhead time and the overall computation time is to apply GPU acceleration on these functions instead of CPU multi-threading [101].

5.4 Numerical Evaluations

5.4.1 Comparison with other models

In this subsection, this work compares the proposed model with a persistence allocation model, a single-slot allocation model, and a double-slot allocation model with deterministic and uncertain availability schedules in two tests, respectively. The given conditions $\hat{s_n}$ and $\hat{f}_n$ are set to 0 in the tests considering deterministic availability schedules.

The persistence allocation model does not consider the service interruptions caused by node unavailabilities. This model maximizes SSCAT by suppressing the interruptions caused by reallocations of VNFs. It determines a node to which each VNF is allocated randomly and keeps this allocation from the first time slot to the last one. This model has no ability to avoid unavailable nodes.

The single-slot allocation model considers the service interruptions caused by node unavailabilities at each time slot, regardless of VNF reallocations

between all adjacent time slots. This model minimizes the number of VNFs allocated to unavailable nodes at each time slot. This model independently determines VNF allocation of each time slot. This model tries to avoid allocating VNFs to unavailable nodes according to the availability schedule. However, this model does not consider the relationship between VNF allocations at adjacent time slots; different allocations at adjacent time slots cause service interruptions. For each $t \in T$, the optimization problem of single-slot allocation model is formulated as an ILP problem by:

$$\min \sum_{r \in R} \sum_{k \in K_r} \sum_{n \in N} x_{tn}^{rk} e_t^n \tag{5.19}$$
$$\text{s.t. } (3.9)\text{-}(3.11).$$

The double-slot allocation model is an improvement of the single-slot allocation model. This model computes the VNF allocation at time slot t by considering that in the last time slot, $x_{t-1,n}^{rk}, \forall n \in N, r \in R, k \in K_r$. The solution that minimizes the differences between time slots t and $t-1$ is chosen when there are multiple solutions that minimize $\sum_{r \in R} \sum_{k \in K_r} \sum_{n \in N} x_{tn}^{rk} e_t^n$. For each $t \in T$, the optimization problem of double-slot allocation model is formulated by:

$$\min \sum_{r \in R} \sum_{k \in K_r} \sum_{n \in N} (x_{tn}^{rk} e_t^n + \epsilon_3 x_{tn}^{rk} \odot x_{t-1,n}^{rk}) \tag{5.20}$$
$$\text{s.t. } (3.9) - (3.11).$$

A small number, ϵ_3, is multiplied to the second term to prioritize the first term over the second term. ϵ_3 is given by $\frac{1}{|N||R| \max_{r \in R}\{K_r\}}$.

The proposed model, the single-slot allocation model, and the double-slot allocation model are solved by the IBM(R) ILOG(R) CPLEX(R) Interactive Optimizer with version 12.7.1 [82], using Intel Core i7-7700 3.60 GHz 4-core CPU, 32 GB memory. The persistence allocation model is implemented by Python 3.7 and runs on the same hardware.

In the first test, the evaluation compares the proposed model with the three baseline models considering deterministic availability schedules in terms of the objective value. Seven cases are examined in this situation. The conditions of these cases are shown in Table 5.1. In this table, *Nodes* means the number

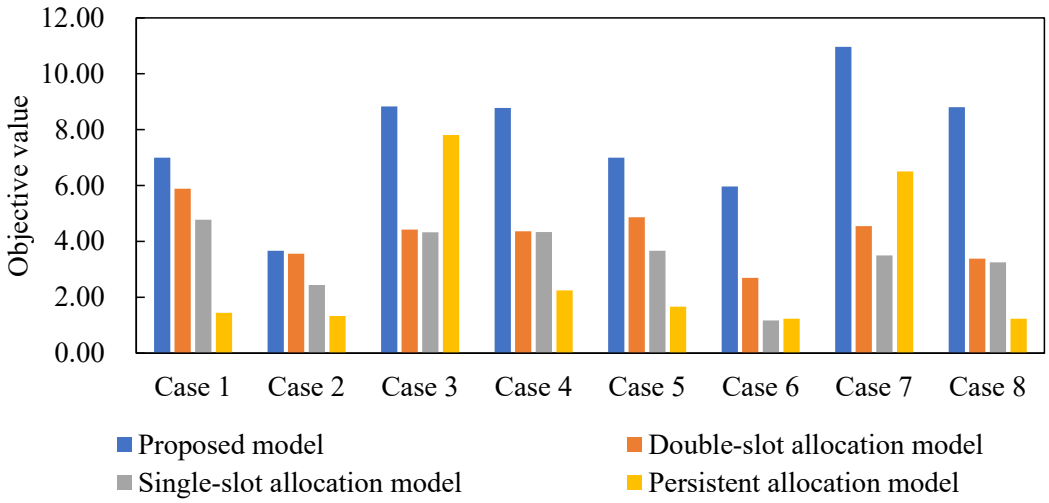


Figure 5.4: Evaluation results of test 1.

of nodes; *Capacity* means the capacity of each node at each time slot; *Requests* means the number of functions in each request; *Functions* means the number of functions in each request; *Time slots* means the number of time slots; *Unavailability parameters* shows the beginning time slot and duration of each unavailability period. The results are shown in Fig. 5.4. It shows that the proposed model outperforms the three baseline models when availability schedule is deterministic in terms of the objective value.

In the second test, the evaluation compares the proposed model with the three baseline models considering uncertain availability schedule in terms of the objective value. The three baseline models obtain the value of e_t^n by using (5.7a)-(5.7h), and (5.9a)-(5.9b) and consider the worst case in uncertainty set $\mathcal{P}$. The evaluation considers two cases in this text. In the first case, a four-node network is considered. The evaluation determines the allocation of functions among six time slots in two requests. The lengths of two requests are two and three, respectively. The uncertainty set is given by: $\bar{s_1} = 2, \hat{s_1} = 1, \bar{f_1} = 1, \bar{s_2} = 2, \bar{f_2} = 2, \hat{f_2} = 1$, other $\bar{s}$ are B_S, and $\hat{s}, \bar{f}$, and $\hat{f}$ are zero. In the second case, a ten-node network is considered. The evaluation determines the allocation of functions among seven time slots in three requests. The lengths of three requests are two, three and five, respectively. The uncertainty set is given by: $\bar{s_1} = 3, \bar{f_1} = 1, \hat{f_1} = 1, \bar{s_2} = 2, \bar{f_2} = 2, \bar{s_3} = 6, \bar{f_3} = 1, \bar{s_5} = 4, \hat{s_5} = 1, \bar{f_5} = 1, \hat{f_5} =$

Table 5.1: Evaluation conditions in test 1.

Case	Nodes	Capacity	Request	Functions	Time slots	Unavailability parameters
1	5	2	3	2, 2, 3	6	$s_1 = 2, f_1 = 3$, other $s_n = B_{\mathrm{S}}$ and $f_n = 0$
2	5	2	3	2, 2, 3	6	$s_1 = 2, f_1 = 3, s_2 = 1, f_2 = 2, s_3 = 5, f_3 = 0$, other $s_n = B_{\mathrm{S}}$ and $f_n = 0$
3	5	2	3	2, 2, 3	12	$s_1 = 2, f_1 = 3, s_2 = 1, f_2 = 2, s_3 = 5, f_3 = 0$, other $s_n = B_{\mathrm{S}}$ and $f_n = 0$
4	5	2	3	2, 2, 3	12	$s_1 = 2, f_1 = 3, s_2 = 1, f_2 = 3, s_3 = 6, f_3 = 4$, other $s_n = B_{\mathrm{S}}$ and $f_n = 0$
5	15	2	5	2, 3, 4, 5, 7	6	$s_1 = 2, f_1 = 3, s_2 = 1, f_2 = 3, s_3 = 6, f_3 = 0$, other $s_n = B_{\mathrm{S}}$ and $f_n = 0$
6	15	2	5	2, 3, 4, 5, 7	6	$s_1 = 2, f_1 = 3, s_2 = 1, f_2 = 3, s_3 = 6, f_3 = 0, s_5 = 3,$ $f_5 = 3, s_7 = 4, f_7 = 2, s_9 = 1, f_9 = 2$, other $s_n = B_{\mathrm{S}}$ and $f_n = 0$
7	15	2	5	2, 3, 4, 5, 7	12	$s_1 = 2, f_1 = 3, s_2 = 1, f_2 = 3, s_3 = 6, f_3 = 0, s_5 = 3,$ $f_5 = 3, s_7 = 4, f_7 = 2, s_9 = 1, f_9 = 2$, other $s_n = B_{\mathrm{S}}$ and $f_n = 0$
8	15	2	5	2, 3, 4, 5, 7	12	$s_1 = 2, f_1 = 3, s_2 = 1, f_2 = 3, s_3 = 6, f_3 = 4, s_5 = 3, f_5 = 4,$ $s_7 = 4, f_7 = 8, s_9 = 1, f_9 = 2, s_{10} = 6, f_{10} = 1, s_{12} = 6, f_{12} = 5,$ $s_{14} = 1, f_{14} = 0, s_{15} = 9, f_{15} = 2$, other $s_n = B_{\mathrm{S}}$ and $f_n = 0$

$1, \bar{s_6} = 2, \hat{s_6} = 1, \bar{s_7} = 5, \bar{f_7} = 1, \bar{s_9} = 1$, other $\bar{s}$ are B_S, and $\hat{s}, \bar{f}$, and $\hat{f}$ are zero. The evaluation evaluates the SSCAT values and the sum of SCATs of all requests in different Γ_n^F and Γ_n^S. The result is shown in Table 5.2. Table 5.2 shows that the proposed model outperforms the three baseline models under all levels of robustness in terms of the objective value. With the increasing of the level of robustness of the solution, the objective value decreases. By comparing the results between the proposed model and the three baseline models, the allocation considering availability schedule provides higher SSCAT than the allocation without considering availability schedule and the longer time slots the model considers, the allocation with higher SSCAT the model provides.

5.4.2 Evaluation of different uncertainty sets

In this subsection, this work evaluates the influences of different uncertainty sets with the same graph and robustness level. Because the durations of unavailability periods $f_n, n \in N$ are determined after the robustness levels Γ_n^F are given. The evaluation forces on the influence of $f_n, n \in N$ on the objective values. The evaluation is performed in a graph which has five nodes whose capacities are two. Three requests need to be allocated in the graph in seven time slots, whose lengths are two, two, three, respectively. There are four tests with different $\bar{s_n}$ and $\hat{s_n}$. The parameters and availability schedules are shown in Fig. 5.5.

This work evaluates the relationship between different shapes of availability schedules and the objective values under the same value of $\Gamma_n^S, n \in N$, which is $\frac{1}{\hat{s_n}}$. The objective values and the availability schedule under the worst case in each test are shown in Fig. 5.6. When unavailabilities are concentrated in the middle of the availability schedule, it is most likely to be the availability schedule under the worst case as shown in Fig. 5.6(a), 5.6(b), and 5.6(c). When the unavailabilities are distributed among nodes in different time slots, it is most likely to be the availability schedule under the worst case as shown in Fig. 5.6(a) and 5.6(b). By taking advantage of this observation, a number of possible choices, such as the availability schedule with the unavailabilities concentrated in one time slot, can be removed and the computational time can be reduced. The worst case availability schedules in Fig. 5.5(a) and Fig. 5.5(c)

Table 5.2: Evaluation results of test 2.

(a) Objective values in case 1.

$(\Gamma_2^{\mathrm{F}}, \Gamma_1^{\mathrm{S}})$	$(-1, \frac{1}{3})$	$(-1, \frac{2}{3})$	$(-1, 1)$	$(0, \frac{1}{3})$	$(0, \frac{2}{3})$	$(0, 1)$	$(1, \frac{1}{3})$	$(1, \frac{2}{3})$	$(1, 1)$
Proposed model	3.75	3.75	3.75	2.67	2.67	2.67	2.67	2.67	2.67
Double-slot allocation model	3.58	3.67	3.67	2.50	2.58	2.58	2.50	2.58	2.58
Single-slot allocation model	2.33	2.33	2.33	2.33	2.42	2.42	1.25	1.33	1.33
Persistent allocation model	2.33	2.33	2.33	2.33	2.33	2.33	1.17	1.17	1.17

(b) Objective values in case 2.

$(\Gamma_1^{\mathrm{F}}, \Gamma_5^{\mathrm{F}}, \Gamma_5^{\mathrm{S}}, \Gamma_6^{\mathrm{S}})$	$(-1, -1, \frac{1}{3}, \frac{1}{3})$	$(-1, -1, 1, 1)$	$(0, 0, \frac{1}{3}, \frac{1}{3})$	$(0, 0, 1, 1)$	$(1, 1, \frac{1}{3}, \frac{1}{3})$	$(1, 1, 1, 1)$
Proposed model	4.81	4.81	4.76	4.76	4.76	4.76
Double-slot allocation model	2.33	2.38	2.33	2.38	2.33	2.38
Single-slot allocation model	2.29	2.29	1.24	1.19	1.29	1.19
Persistent allocation model	1.14	0.10	1.14	0.10	1.14	0.10

Time slot \ Node	1	2	3	4	5
1	P	P	P	P	P
2	P	P	P	P	P
3	P	P	P	P	P
4	P	P	P	P	P
5	P	P	P	P	P
6	P	P	P	P	P
7	P	P	P	P	P

(a) Availability schedule 1. $\bar{s_n} = 4, \hat{s_n} = 3, \forall n \in N$.

Time slot \ Node	1	2	3	4	5
1	P	P	P	P	P
2	P	P	P	P	P
3	P	P	P	P	P
4	P	P	P	P	P
5	P	P	P	P	P
6					
7					

(b) Availability schedule 2. $\bar{s_n} = 3, \hat{s_n} = 2, \forall n \in N$.

Time slot \ Node	1	2	3	4	5
1	P				
2	P	P			
3	P	P	P		
4		P	P	P	
5			P	P	P
6				P	P
7					P

(c) Availability schedule 3. $\bar{s_n} = n + 1, \hat{s_n} = 1, \forall n \in N$.

Time slot \ Node	1	2	3	4	5
1	P	P			
2	P	P			
3	P	P			
4					
5			P	P	
6			P	P	
7			P	P	

(d) Availability schedule 4. $\bar{s_1} = \bar{s_2} = 2, \bar{s_3} = \bar{s_4} = 6, \hat{s_n} = 1$.

Figure 5.5: Possible locations of unavailabilities in four availability schedules. If "P" is marked in a time slot, it is a possible location of an unavailability, and otherwise it is available.

are the same, which is Fig. 5.6(a). The work can reduce the input parameters of Fig. 5.5(a) to those of Fig. 5.5(c) in order to reduce the size of the uncertainty set.

5.4.3 Effect of heuristic algorithm

This work evaluates the performance of the heuristic algorithm in terms of the objective value and the computation time, which is called test 3. The evaluation uses cases 1, 4, and 6 in Table 5.1 in this evaluation with uncertain parameters in Table 5.3. In each case, the evaluation randomly sets several

Time slot \ Node	1	2	3	4	5
1					
2	U				
3		U			
4			U		
5				U	
6					U
7					

(a) Worst case availability schedule 1 and 3. Obj. value is 3.52.

Time slot \ Node	1	2	3	4	5
1	U				
2		U			
3			U		
4				U	
5					U
6					
7					

(b) Worst case availability schedule 2. Obj. value is 3.57.

Time slot \ Node	1	2	3	4	5
1					
2					
3	U	U			
4					
5			U	U	
6					
7					

(c) Worst case availability schedule 4. Obj. value is 4.57.

Figure 5.6: Worst case availability schedule in four tests. If "U" is marked in a time slot, it is unavailable, and otherwise available.

Table 5.3: Uncertain conditions in test 3.

Case	Uncertain unavailability parameters
1	$\bar{s_1} = 2, \hat{s_1} = 1, \bar{f}_1 = 1, \bar{s_2} = 2, \bar{f}_2 = 2, \hat{f}_2 = 1$, other $\bar{s}$ are $\boldsymbol{B}_{\mathrm{S}}$, and $\hat{s}$, $\bar{f}$, and $\hat{f}$ are zero.
4	$\bar{s_1} = 2, \hat{s_1} = 1, \bar{f}_1 = 3, \hat{f}_1 = 1, \bar{s_2} = 1, \bar{f}_2 = 3, \hat{f}_2 = 2, \bar{s_3} = 6, \bar{f}_3 = 4$, other $\bar{s}$ are $\boldsymbol{B}_{\mathrm{S}}$, and $\hat{s}$, $\bar{f}$, and $\hat{f}$ are zero.
6	$\bar{s_1} = 2, \bar{f}_1 = 3, \bar{s_2} = 1, \bar{f}_2 = 3, \hat{f}_2 = 2, \bar{s_3} = 6, \hat{s_3} = 3, \bar{s_5} = 3, \bar{f}_5 = 3, \bar{s_9} = 3, \bar{f}_9 = 2, \hat{f}_9 = 1$, other $\bar{s}$ are $\boldsymbol{B}_{\mathrm{S}}$, and $\hat{s}$, $\bar{f}$, and $\hat{f}$ are zero.

different values of $\boldsymbol{\Gamma}_n^{\mathrm{F}}$ and $\boldsymbol{\Gamma}_n^{\mathrm{S}}$ for each case. The evaluation computes the allocations with these different robustnesses by using the heuristic algorithm and the MILP approach. Table 5.4 shows the parameter setting used in this evaluation. The heuristic algorithm is implemented by C++ 15, compiled by Microsoft Visual C++ 2017 v15.9.16, using Intel Core i7-7700 3.60 GHz 4-core CPU, 32 GB memory.

Table 5.5 compares the results of each test obtained by using the heuristic algorithm with those obtained by the MILP approach. The evaluation observe that the objective values obtained by the heuristic algorithm have smaller differences with those of the MILP approach, as the size of the network is

Table 5.4: Parameter setting.

Parameter	Setting
MG	250
IPN	12
$ULPN$	100
ICP	0.6
ECP	0.3
MP	0.3

smaller or the number of unavailabilities is smaller. Note that, when the size of the network is larger or the number of unavailabilities is larger, the difference becomes large; they are 13.79% in rows 7 and 8 of case 4 and 24.41% in row 3 of case 6 in Table 5.5. The difference in test 3 between the MILP approach and the heuristic algorithm is 3.37% on average.

Table 5.6 shows the average computation times for the MILP approach, the heuristic algorithm, and the CPU accelerated heuristic algorithm for each case. The evaluation observes that the heuristic algorithm without CPU-accumulation reduces the computation time by 99.47% compared with the MILP approach on average; the larger the problem size is, the more the computation time of the heuristic algorithm is reduced compared with that of the MILP approach. The CPU-accumulation reduces the computation time by 71.07% compared with the heuristic algorithm without CPU-accumulation.

5.4.4 Large-scale evaluation

This work compares the proposed model with the baseline models with larger test cases in this subsection. The MILP approach spends so much time that it cannot be applied in large-scale tests cases as shown in Table 5.6. This evaluation compares the results obtained from the heuristic algorithm and three baseline models. The parameters of the algorithm are shown in Table 5.4. The evaluation prepares four test cases with different given conditions as shown

Table 5.5: Comparison of objective values in test 3.

Case	Values of Γ^{F} and Γ^{S}†	Objective value		
		MILP	Heuristic algorithm	Difference
Case 1	$(-1, \frac{1}{3})$	3.83	3.83	0.00%
	$(-1, \frac{2}{3})$	3.83	3.83	0.00%
	$(-1, 1)$	3.83	3.83	0.00%
	$(0, \frac{1}{3})$	3.67	3.67	0.00%
	$(0, \frac{2}{3})$	3.67	3.50	4.63%
	$(0, 1)$	3.67	3.67	0.00%
	$(1, \frac{1}{3})$	3.67	3.67	0.00%
	$(1, \frac{2}{3})$	3.67	3.50	4.63%
	$(1, 1)$	3.67	3.67	0.00%
Case 4	$(-1, -1, \frac{1}{3})$	7.81	7.81	0.00%
	$(-1, -1, 1)$	7.81	7.81	0.00%
	$(-1, 1, \frac{1}{3})$	6.69	6.69	0.00%
	$(-1, 1, 1)$	6.69	6.69	0.00%
	$(1, -1, \frac{1}{3})$	6.67	6.50	2.55%
	$(1, -1, 1)$	6.67	6.50	2.55%
	$(1, 1, \frac{1}{3})$	6.67	5.75	13.79%
	$(1, 1, 1)$	6.67	5.75	13.79%
Case 6	$(-1, -1, \frac{1}{3})$	4.93	4.90	0.61%
	$(-1, -1, 1)$	4.93	4.90	0.61%
	$(1, 1, 1)$	3.80	2.88	24.21%

†$(\Gamma_2^{\mathrm{F}}, \Gamma_1^{\mathrm{S}})$ for Case 1; $(\Gamma_1^{\mathrm{F}}, \Gamma_2^{\mathrm{F}}, \Gamma_1^{\mathrm{S}})$ for Case 4; $(\Gamma_2^{\mathrm{F}}, \Gamma_9^{\mathrm{F}}, \Gamma_3^{\mathrm{S}})$ for Case 6.

Table 5.6: Comparison of average computation times in test 3.

Case	MILP (s)	Heuristic algorithm (s)	CPU accelerated heuristic algorithm (s)
Case 1	21.43	6.19	2.86
Case 4	249.66	10.69	5.84
Case 6	11310.59	44.89	9.17

in Table 5.7.

The evaluation calculates the allocation of VNFs with given conditions by using the heuristic algorithm and three baseline models. The objective values of the above four methods are shown in Table 5.8. Allocations obtained from the heuristic algorithm have larger objective values than those from the compared baseline models, especially in cases 9 and 10, where the available space for functions is large and the number of functions is small. Even if there are performance gaps between the results obtained from the heuristic algorithm and those from the proposed model, the results of the heuristic algorithm are still better than those of the baseline models in examined cases.

5.5 Discussions

5.5.1 Maintenance ability

Sometimes the maintenance ability of SP is limited. The number of unavailable nodes in one time slot is limited. This section assumes that the maintenance ability on time slot t is limited from M_t^{L} to M_t^{U}, where $M_t^{\mathrm{L}} \leq \sum_{n \in N} e_n^t \leq M_t^{\mathrm{U}}$. By using the known maintenance ability, the size of the uncertainty set $\mathcal{P}_n$ can be reduced to reduce the computation time. The reduction is performed by using the following algorithm which is applied before Algorithm 3.1.

If the MILP approach is applied to solve this model, this work adds the following equations to reduce the number of possible choices and computational

Table 5.7: Evaluation conditions in Section 5.4.4.

Case	Nodes	Capacity	Request	Length†	Time slots	Unavailability parameters
9	24	2	12	2:8, 3:2, 4:2	6	$\bar{s_1} = \bar{s_2} = \bar{s_7} = \bar{s_8} = \bar{s_{13}} = \bar{s_{14}} = \bar{s_{19}} = \bar{s_{20}} = 2, \hat{s_1} = \hat{s_7} = \hat{s_{13}} = \hat{s_{19}} = 1, \bar{f_1} = \bar{f_7} = \bar{f_{13}} = \bar{f_{19}} = 1, \bar{f_2} = \bar{f_8} = \bar{f_{14}} = \bar{f_{20}} = 2, \hat{f_2} = \hat{f_8} = \hat{f_{14}} = \hat{f_{20}} = 1$, other $s_n = B_{\mathrm{S}}$ and $f_n = 0$.
10	24	2	15	2:10, 3:3, 4:2	6	$\bar{s_1} = \bar{s_2} = \bar{s_7} = \bar{s_8} = \bar{s_{13}} = \bar{s_{14}} = \bar{s_{19}} = \bar{s_{20}} = 2, \hat{s_1} = \hat{s_7} = \hat{s_{13}} = \hat{s_{19}} = 1, \bar{f_1} = \bar{f_7} = \bar{f_{13}} = \bar{f_{19}} = 1, \bar{f_2} = \bar{f_8} = \bar{f_{14}} = \bar{f_{20}} = 2, \hat{f_2} = \hat{f_8} = \hat{f_{14}} = \hat{f_{20}} = 1$, other $s_n = B_{\mathrm{S}}$ and $f_n = 0$.
11	24	2	18	2:12, 3:3, 4:3	6	$\bar{s_1} = \bar{s_2} = \bar{s_7} = \bar{s_8} = \bar{s_{13}} = \bar{s_{14}} = \bar{s_{19}} = \bar{s_{20}} = 2, \hat{s_1} = \hat{s_7} = \hat{s_{13}} = \hat{s_{19}} = 1, \bar{f_1} = \bar{f_7} = \bar{f_{13}} = \bar{f_{19}} = 1, \bar{f_2} = \bar{f_8} = \bar{f_{14}} = \bar{f_{20}} = 2, \hat{f_2} = \hat{f_8} = \hat{f_{14}} = \hat{f_{20}} = 1$, other $s_n = B_{\mathrm{S}}$ and $f_n = 0$.
12	36	2	18	2:12, 3:3, 4:3	6	$\bar{s_1} = \bar{s_2} = \bar{s_7} = \bar{s_8} = \bar{s_{13}} = \bar{s_{14}} = \bar{s_{19}} = \bar{s_{20}} = \bar{s_{25}} = \bar{s_{26}} = \bar{s_{31}} = \bar{s_{32}} = 2, \hat{s_1} = \hat{s_7} = \hat{s_{13}} = \hat{s_{19}} = \hat{s_{25}} = \hat{s_{31}} = 1, \bar{f_1} = \bar{f_7} = \bar{f_{13}} = \bar{f_{19}} = \bar{f_{25}} = \bar{f_{31}} = 1, \bar{f_2} = \bar{f_8} = \bar{f_{14}} = \bar{f_{20}} = \bar{f_{26}} = \bar{f_{32}} = 2\hat{f_2} = \hat{f_8} = \hat{f_{14}} = \hat{f_{20}} = \hat{f_{26}} = \hat{f_{32}} = 1$, other $s_n = B_{\mathrm{S}}$ and $f_n = 0$.

†x:y: there are y requests whose length are x.

Table 5.8: Evaluation results in Section 5.4.4.

Case	Obtained from	$(\Gamma_2^{\mathrm{F}}, \Gamma_1^{\mathrm{S}})$					
		$(0, \frac{1}{3})$	$(0, \frac{2}{3})$	$(0, 1)$	$(1, \frac{1}{3})$	$(1, \frac{2}{3})$	$(1, 1)$
9	Heuristic algorithm	7.00	7.00	7.00	7.00	7.00	7.00
	Double-slot allocation model	4.82	4.82	4.82	4.82	4.82	4.82
	Single-slot allocation model	2.33	2.33	2.33	2.33	2.33	2.33
	Persistent allocation model	2.67	2.67	2.67	1.58	1.58	1.58
10	Heuristic algorithm	2.91	2.91	3.50	2.91	2.87	2.87
	Double-slot allocation model	2.76	2.76	2.76	2.76	2.76	2.76
	Single-slot allocation model	2.33	2.33	2.33	1.31	1.31	1.31
	Persistent allocation model	2.69	2.69	2.69	1.62	1.62	1.62
11	Heuristic algorithm	2.83	2.81	2.85	2.41	1.81	2.56
	Double-slot allocation model	2.67	2.67	2.67	2.67	2.67	2.67
	Single-slot allocation model	2.33	2.33	2.33	1.27	1.27	1.27
	Persistent allocation model	2.67	2.67	2.67	1.59	1.59	1.59
12	Heuristic algorithm	7.00	7.00	7.00	7.00	7.00	7.00
	Double-slot allocation model	4.82	4.82	4.82	4.82	4.82	4.82
	Single-slot allocation model	2.33	2.33	2.33	2.33	2.33	2.33
	Persistent allocation model	2.66	2.66	2.66	1.59	1.59	1.59

Algorithm 5.7 Uncertainty set reduction by using maintenance ability

Input: M_t^{L}, M_t^{U}, $\mathcal{P}$, Γ_n^{F}, $\bar{s_n}$, $\hat{s_n}$, $\bar{f_n}$, $\hat{f_n}$, T, N

Output: new uncertainty set

1: **for** $p \in \mathcal{P}$ **do**
2: Calculate the value of e_t^n according to (5.1) by using p and f_n
3: **for** $t \in T$ **do**
4: **if** $\sum_{n \in N} e_t^n < M_t^{\mathrm{L}}$ or $\sum_{n \in N} e_t^n > M_t^{\mathrm{U}}$ **then**
5: Delete the current p from $\mathcal{P}$
6: **end if**
7: **end for**
8: **end for**

time by using the maintenance ability:

$$\begin{aligned}&\textbf{If} \quad M_t^{\mathrm{L}} \le \textstyle\sum_{n\in N} e_t^{np} \le M_t^{\mathrm{U}} \quad \textbf{then}\\ &\qquad \Delta_t^p = 1\\ &\textbf{Else}\\ &\qquad \Delta_t^p = 0,\end{aligned} \tag{5.21a}$$

$$\Delta_t^p \in \{0,1\}, \forall p \in \mathcal{P}, t \in T, \tag{5.21b}$$

which can be linearized by using Appendix A. (5.6) is changed to:

$$\max \min_{p\in\mathcal{P}} \left\{ (\lambda^p + \epsilon \sum_{r\in R} \beta_r^p) \cdot \sum_{t\in T} \left\{ \Delta_t^p \cdot B \right\} \right\}. \tag{5.22}$$

Here, B is a large constant which is larger than $(|R|+1)\cdot|T|$.

5.5.2 More than one unavailability period per node

Deterministic unavailability period

The model in Section 5.2 assumes that there is at most one unavailability period on each node. Let us give an extension for supporting more than one unavailability period on each node in this model.

A_n is a set for unavailability periods on node $n \in N$. s_n^a is the starting time slot of the ath unavailability period in set A_n. f_n^a is the duration of the ath unavailability period in set A_n. Equation (5.1) is replaced by:

$$\begin{aligned}&\textbf{If} \quad s_n^a \le t \le s_n^a + f_n^a \quad \textbf{then}\\ &\qquad e_{tn}^a = 1\\ &\textbf{Else}\\ &\qquad e_{tn}^a = 0,\end{aligned} \tag{5.23a}$$

$$e_t^n = \vee_{a\in A_n} e_{tn}^a, \forall n \in N, t \in T, \tag{5.23b}$$

$$e_{tn}^a \in \{0,1\}, \forall n \in N, t \in T, a \in A_n, \tag{5.23c}$$

$$e_t^n \in \{0,1\}, \forall n \in N, t \in T. \tag{5.23d}$$

According to the linearization process in Appendix A, (5.23) is linearized and (5.4a)-(5.4h) are replaced by the following equations:

$$t - s_n^a + \epsilon_1 \le \eta_{tn}^a \cdot B, \forall t \in T, n \in N, a \in A_n, \tag{5.24a}$$

$$t - s_n^a + \epsilon_1 \geq (\eta_{tn}^a - 1) \cdot B, \forall t \in T, n \in N, a \in A_n, \tag{5.24b}$$

$$s_n^a + f_n^a - t + \epsilon_1 \leq \rho_{tn}^a \cdot B, \forall t \in T, n \in N, a \in A_n, \tag{5.24c}$$

$$s_n^a + f_n^a - t + \epsilon_1 \geq (\rho_{tn}^a - 1) \cdot B, \forall t \in T, n \in N, a \in A_n, \tag{5.24d}$$

$$e'_{atn} \leq \eta_{tn}^a, \forall t \in T, n \in N, a \in A_n, \tag{5.24e}$$

$$e'_{atn} \leq \rho_{tn}^a, \forall t \in T, n \in N, a \in A_n, \tag{5.24f}$$

$$e'_{atn} \geq \eta_{tn}^a + \rho_{tn}^a - 1, \forall t \in T, n \in N, a \in A_n, \tag{5.24g}$$

$$e_t^n \geq \frac{1}{|A_n|} \cdot \sum_{a \in A_n} e'_{atn}, \forall t \in T, n \in N, \tag{5.24h}$$

$$e_t^n \leq \sum_{a \in A_n} e'_{atn}, \forall t \in T, n \in N, \tag{5.24i}$$

$$e'_{atn}, \eta_{tn}^a, \rho_{tn}^a \in [0, 1], \forall t \in T, n \in N, a \in A_n, \tag{5.24j}$$

$$e_t^n \in \{0, 1\}, \forall n \in N, t \in T. \tag{5.24k}$$

Uncertain unavailability period

Confronted with uncertain unavailability periods and multiple unavailability periods, Γ_n^{S} still controls the size of uncertainty set of beginning time slots $\mathcal{P}_n$ on node n. The larger Γ_n^{S} is, the more the considered possible beginning time slots of unavailability periods on node n are. For unavailability period $a \in A_n$, this work chooses $\lfloor \Gamma_n^{\mathrm{S}} \cdot (2 \cdot \hat{s_n^a} + 1) \rfloor$ different beginning time slots from $[\bar{s_n^a} - \hat{s_n^a}, \bar{s_n^a} + \hat{s_n^a}]$ and form a set which is an element in $\mathcal{P}_n^a$. $\mathcal{P}_n^a$ contains all possible choices of uncertainty period a on node n. The size of $\mathcal{P}_n^a$ is $|\mathcal{P}_n^a| = \begin{pmatrix} 2 \cdot \hat{s_n^a} + 1 \\ \lfloor \Gamma_n^{\mathrm{S}} \cdot (2 \cdot \hat{s_n^a} + 1) \rfloor \end{pmatrix}$. The uncertainty set for all nodes is denoted by $\mathcal{P} = \{\{\mathcal{P}_1^1, \cdots, \mathcal{P}_1^{|A_n|}\}, \{\mathcal{P}_2^1, \cdots\}, \cdots, \{\cdots, \mathcal{P}_{|N|}^{|A_n|}\}\}$. One element in $\mathcal{P}$ is a combination of one element in each $\mathcal{P}_n$. One element in $\mathcal{P}_n$ is a combination of one element in each $\mathcal{P}_n^a$. Let p denote an element in $\mathcal{P}$; p_n denotes the selected set from $\mathcal{P}_n$; p_n^q denotes the qth element in p_n; p_n^{qa} denotes the selected starting time slot of ath unavailability period in the element in p_n^q.

Γ_n^{F} controls the considered length of the duration of unavailability period on node n, which is $\bar{f_n^a} + \hat{f_n^a} \cdot \lfloor \Gamma_n^{\mathrm{F}} \rfloor$.

Equations (5.7a)-(5.7h) are replaced by:

$$t - p_n^{qa} + \epsilon_1 \leq \eta_{tn}^{qa} \cdot B, \forall t \in T, n \in N, q \in [1, |p_n|], a \in A_n, \tag{5.25a}$$

$$t - p_n^{qa} + \epsilon_1 \geq (\eta_{tn}^{qa} - 1) \cdot B, \forall t \in T, n \in N, q \in [1, |p_n|], a \in A_n, \tag{5.25b}$$

$$p_n^{qa} + \bar{f}_n^a + \Gamma_n^{\mathrm{F}} \cdot \hat{f}_n^a - t + \epsilon_1 \leq \rho_{tn}^{qa} \cdot B, \forall t \in T, n \in N, q \in [1, |p_n|], a \in A_n, \tag{5.25c}$$

$$p_n^{qa} + \bar{f}_n^a + \Gamma_n^{\mathrm{F}} \cdot \hat{f}_n^a - t + \epsilon_1 \geq (\rho_{tn}^{qa} - 1) \cdot B, \forall t \in T, n \in N, q \in [1, |p_n|], a \in A_n, \tag{5.25d}$$

$$\tau'_{tnqa} \leq \eta_{tn}^{qa}, \forall t \in T, n \in N, q \in [1, |p_n|], a \in A_n, \tag{5.25e}$$

$$\tau'_{tnqa} \leq \rho_{tn}^{qa}, \forall t \in T, n \in N, q \in [1, |p_n|], a \in A_n, \tag{5.25f}$$

$$\tau'_{tnqa} \geq \eta_{tn}^{qa} + \rho_{tn}^{qa} - 1, \forall t \in T, n \in N, q \in [1, |p_n|], a \in A_n, \tag{5.25g}$$

$$\tau_{tn}^{q} \geq \frac{1}{|A_n|} \cdot \sum_{a \in A_n} \tau'_{tnqa}, \forall t \in T, n \in N, q \in [1, |p_n|], \tag{5.25h}$$

$$\tau_{tn}^{q} \leq \sum_{a \in A_n} \tau'_{tnqa}, \forall t \in T, n \in N, q \in [1, |p_n|], \tag{5.25i}$$

$$\tau'_{tnqa}, \eta_{tn}^{qa}, \rho_{tn}^{qa} \in [0, 1], \forall t \in T, n \in N, q \in [1, |p_n|], a \in A_n, \tag{5.25j}$$

$$\tau_{tn}^{q} \in [0, 1], \forall t \in T, n \in N, q \in [1, |p_n|]. \tag{5.25k}$$

5.5.3 Unpredictable unavailabilities

This work assumes that the availability schedules used in the proposed model are known by maintenance schedules in the proposed model. In maintenance schedules, the locations of availabilities and unavailabilities are given so that the availability schedules can be expressed by binary matrixes. By comparison, there are some availabilities and unavailabilities whose locations cannot be specified, such as burst unavailabilities and probabilistic unavailabilities.

Burst unavailabilities are caused by burst hardware and software failures of servers. The influences of burst unavailabilities cannot be avoided by the scheduling of the proposed model. The common ways to suppress the influence of the burst unavailabilities are backup and replication. Probabilistic unavailabilities from prediction systems give probabilistic results instead of the exact results. If this work wants to use the probabilistic results as binary availability schedules, quantization of the probabilistic results are necessary, which will bring extra uncertainty. If the administrators consider that the cost is accept-

able, they can set a threshold of probability. If the probability exceeds the threshold, it is considered to be one; otherwise, zero.

5.6 Directions to extend proposed model

5.6.1 Separate request from SFC

This work assumes that one request corresponds to one SFC in Section 5.2. This work does not consider sharing VNFs among different SFCs in the proposed model. This work gives a direction on how to separate the requests from the SFCs in this subsection. This work redefines the requests, SFCs, and VNFs to replace the definitions in Section 5.2. The definition of decision variable x_{tn}^{rk} in Section 5.2 is changed as follows.

R represents the set of requests from the users. C represents the set of SFCs waiting for provisions. Each SFC is an ordered set of VNFs. F is the set of functions. $F^c \subseteq F$ is the ordered set of VNFs used in SFC $c \in C$. Binary given parameter ψ_{fc} is set to 1 if function $f \in F$ is used in SFC $c \in C$. Binary given parameter γ_{cr}, $r \in R$, $c \in C$, is set to 1 if request r requests SFC c; 0 otherwise. Given parameter q_s^f represents the amount of resource $s \in S$ which function $f \in F$ requires. This work uses binary decision variable x_{tn}^f to represent the allocation; x_{tn}^f is set to 1 if function $f \in F$ is assigned to node $n \in N$ at time slot $t \in T$, and 0 otherwise.

According to the above redefinitions, the work gives a new version of the related parameters and constraints as follows.

$$o_t^r = \prod_{n\in N}\Bigg\{\Bigg(\prod_{c\in C}\prod_{f\in F^c}\psi_{fc}\gamma_{cr}(x_{tn}^f \odot x_{t-1,n}^f)\Bigg)\wedge\Bigg(1-(\prod_{c\in C}\prod_{f\in F^c}$$
$$\psi_{fc}\gamma_{cr}x_{tn}^f)\wedge e_t^n\Bigg)\wedge\Bigg(1-(\prod_{c\in C}\prod_{f\in F^c}\psi_{fc}\gamma_{cr}x_{t-1,n}^f)\wedge e_{t-1}^n\Bigg)\Bigg\},$$
$$\forall r \in R, t \in T\backslash\{1\}, \tag{5.26a}$$
$$\sum_{f\in F} x_{tn}^f q_s^f \le c_{nt}^s, \forall n \in N, t \in T, s \in S, \tag{5.26b}$$
$$\sum_{f\in F^c} x_{tn}^f \le 1, \forall c \in C, n \in N, t \in T, \tag{5.26c}$$

$$\sum_{n \in N} x_{tn}^{f} = 1, \forall f \in F, t \in T, \tag{5.26d}$$

$$x_{tn}^{f} \wedge e_{t}^{n} = 0, \forall t \in T, n \in N, f \in F. \tag{5.26e}$$

Equation (5.26a) replaces x_{tn}^{rk} in (3.1) with $\prod_{c \in C} \prod_{f \in F^c} \psi_{fc}\ \gamma_{cr} x_{tn}^{f}$. Equation (5.26b) replaces (3.9) in Chapter 3 and ensures that each node's computational resources must not exceed its capacity during allocation. Equation (5.26c) replaces (3.10) in Chapter 3 and assumes that one service chain does not allocate multiple VNFs in this chain on one VM to avoid the influence of the reallocation of VMs [44]. Equation (5.26d) replaces (3.11) in Chapter 3 and ensures that all functions are allocated in the network.

5.6.2 Multiple active replicas for a VNF

This work assumes that only one replica is active for each VNF at the same time regardless of the required processing abilities of requests and the processing abilities which can be provided by VNFs in Section 5.2. Actually, multiple replicas can be active at the same time and the processing abilities of the active replicas need to meet the required processing abilities of the requests. Based on Section 5.6.1, this work introduces the following parameters about replicas and constraints.

Each VNF $f \in F$ can be replaced by a pool of A^f replica VNFs, each of which requires different capacities with an extra overhead on capacity but behaves collectively as the original one; the capacity of each replica VNF can be different from each other associated with the same original VNF. p_{fa} represents the processing ability of the ath replica of VNF $f \in F$. q_s^{fs} represents the resource requirement of the ath replica of VNF $f \in F$ for resource $s \in S$. The required processing ability of request $r \in R$ is denoted by g_r, which is a given parameter. This work uses binary decision variable x_{tn}^{fa} to represent the allocation; x_{tn}^{fa} is set to 1 if the ath replica of function $f \in F$ is assigned to node $n \in N$ at time slot $t \in T$, and 0 otherwise.

$$o_t^r = \prod_{n \in N} \Bigg\{ \Bigg(\prod_{c \in C} \prod_{f \in F^c} \psi_{fc} \gamma_{cr} \prod_{a \in A^f} (x_{tn}^{fa} \odot x_{t-1,n}^{fa}) \Bigg) \wedge \Bigg(1 - (\prod_{c \in C}$$

$$\prod_{f\in F^c}\psi_{fc}\gamma_{cr}\prod_{a\in A^f}x_{tn}^{fa})\wedge e_t^n\Big)\wedge\Big(1-(\prod_{c\in C}\prod_{f\in F^c}\psi_{fc}\gamma_{cr}\prod_{a'\in A^f}$$

$$x_{t-1,n}^{fa})\wedge e_{t-1}^n\Big)\Big\},\forall r\in R,t\in T\backslash\{1\},\tag{5.27a}$$

$$\sum_{f\in F}\sum_{a\in A^f}x_{tn}^{fa}q_s^{fa}\le c_{nt}^s,\forall n\in N,t\in T,s\in S,\tag{5.27b}$$

$$\sum_{a\in A^f}x_{tn}^{fa}\le 1,\ \forall c\in C,n\in N,t\in T,f\in F,\tag{5.27c}$$

$$x_{tn}^{fa}\wedge e_t^n=0,\ \forall t\in T,n\in N,f\in F,a\in A^f,\tag{5.27d}$$

$$\sum_{n\in N}\sum_{a\in A^f}p_{fa}x_{tn}^{fa}\ge\sum_{c\in C}\psi_{fc}\sum_{r\in R}\gamma_{cr}g_r,\forall t\in T,f\in F.\tag{5.27e}$$

Equations (5.26a)-(5.26e) are replaced by (5.27a)-(5.27d). Equation (5.27b) ensures that each node's computational resources must not exceed its capacity during allocation. Equation (5.27c) ensures that the replicas of the same VNF cannot be assigned to the same node. Equation (5.27e) ensures that the sum of the processing abilities of the replicas of each VNF meets the required processing abilities from the requirements at each time slot.

5.6.3 Network-aware placement

The proposed model does not consider the routing between VNFs in the same SFC or the recovery path from the unavailable VNFs to their new locations. In a real deployment, some paths cannot be chosen because of the limitation of the characteristic of links, such as bandwidth and latency. This subsection gives a direction on how to handle the routing problems by taking advantage of the network topology. Each virtual link $(i,j)\in L$ corresponds to a connection between two VMs with transmission resource b_{ij} and length l_{ij}. The transmission resources demanded by request $r\in R$ is d_r. The latency of request $r\in R$ is required to be less than ι_r.

Routing of SFCs

This work presents the following constraints to compute the VNF allocation and the routes of all SFCs.

The flow constraint of SFC paths is given by: $\forall w \in N, r \in R, k \in K_r, t \in T$,

$$\sum_{(i,j)\in L} \alpha_{w,ij}\rho_{rt}^{k,ij} = \begin{cases} -1, \text{ if } x_{tw}^{rk} = 1 \\ 1, \text{ if } x_{tw}^{r,k+1} = 1 \\ 0, \text{ if } x_{tw}^{rk} = x_{tw}^{r,k+1} = 0. \end{cases} \tag{5.28}$$

For each request, there are three types of nodes: source node (a node at which the first function of a request is allocated), destination node (a node at which the last function of a request is allocated), and others. This work defines indicator $\alpha_{w,ij}, w \in N, (i,j) \in L$, to represent the adjacency of nodes on directed graph G, where $\alpha_{w,ij} = 1$ if node w is the tail of the directed link (i,j), i.e., $w = j$; $\alpha_{w,ij} = -1$ if node w is the head of the directed link (i,j), i.e., $w = i$; $\alpha_{w,ij} = 0$ otherwise. This work uses binary variable $\rho_{rt}^{k,ij}$ to express the route. If link (i,j) is a segment link between the kth function and the $k+1$th function of request $r \in R$ at time slot $t \in T$, $\rho_{rt}^{k,ij} = 1$, and 0 otherwise. This work have the following constraints. According to (3.10) in Chapter 3, x_{tw}^{rk} and $x_{tw}^{r,k+1}$ cannot be 1 at the same time. Thus (5.28) can be simplified to:

$$\sum_{(i,j)\in L} \alpha_{w,ij}\rho_{rt}^{k,ij} = -x_{tw}^{rk} + x_{tw}^{r,k+1}, \forall k \in K_r \setminus \{|K_r|\}, r \in R, t \in T, w \in N. \tag{5.29}$$

The link capacity constraint is given by:

$$\sum_{r\in R}\sum_{k\in K_r} \rho_{rt}^{k,ij} d_r \leq b_{ij}, \forall (i,j) \in L, t \in T, \tag{5.30}$$

which ensures that each link's transmission resource is not overused. The latency constraint is given by:

$$\sum_{t\in T}\sum_{k\in K_r}\sum_{(i,j)\in L} l_{ij}\rho_{rt}^{k,ij} \leq \iota_r, \forall r \in R, \tag{5.31}$$

which ensures that each request meets the requirement of latency.

Routing during recovery

For stateful application, the state information needs to be synchronized between the old and new nodes, which consumes the transmission resource of

links. The flow constraint of recovery is given by: $\forall w \in N, r \in R, k \in K_r, t \in T\backslash\{1\}$,

$$\sum_{(i,j)\in L} \alpha_{w,ij}\tau_{rt}^{k,ij} = \begin{cases} -1, \text{ if } x_{t-1,w}^{rk} = 1 \\ 1, \text{ if } x_{tw}^{r,k} = 1 \\ 0, \text{ if } x_{t-1,w}^{rk} = x_{tw}^{r,k} = 0. \end{cases} \tag{5.32}$$

This work uses binary variable $\tau_{rt}^{k,ij}$ to express the route. If link (i, j) is a segment link between the nodes where the kth function of request $r \in R$ is allocated at time slot $t \in T\backslash\{1\}$ and time slot $t-1$, $\tau_{rt}^{k,ij} = 1$, and 0 otherwise. The link capacity constraint except for the first time slot is given by:

$$\sum_{r\in R}\sum_{k\in K_r} (\rho_{rt}^{k,ij} + \tau_{rt}^{k,ij})d_r \leq b_{ij}, \forall (i,j) \in L, t \in T\backslash\{1\}. \tag{5.33}$$

5.7 Summary

This chapter proposed a robust VNF allocation model for improving the continuous available time of service function chains with considering the uncertain availability schedule. This work formulated the proposed model as an MILP problem. Numerical results showed that the proposed model improves the continuous available time of SFCs, compared with the persistent allocation model, the single-slot allocation model, and the double-slot allocation model in both deterministic and uncertain availability schedules. In the cases examined, the proposed model can provide longer continuous available time slots compared with the three baseline models under different levels of the robustness of uncertain availability schedule. The developed heuristic algorithm reduces the computation time by 99.85% compared with the MILP approach with a limited performance penalty by 3.37% in our evaluations. We evaluated the relationship between availability schedules and objective values. The size of the uncertainty set can be reduced according to our observations. This work gave three discussions of the proposed model: for maintenance ability, for multiple unavailability periods on each node, and for unpredictable unavailabilities. In addition, this work provided three directions to extend the proposed model.

Chapter 6

Fault-tolerant resource allocation model considering joint diversity and redundancy for static requests

This chapter proposes an optimization model to derive a resilient virtual network function allocation in service function chains aiming to reduce the end-to-end (E2E) latency during the migrations from the primary functions to backup functions considering VNF diversity and redundancy ensuring k-fault tolerance [102, 103].

This work notes that k is a parameter to be set by an SP considering the worst-case scenarios, each of which is associated with k node failures. This work considers that such scenarios possibly occur, all of which should be incorporated; the sum of the maximum E2E latencies among functions under all possible failure patterns with k node failures can reflect the fault tolerance of services, instead of considering only one worst-case pattern to minimize the maximum E2E latency.

An example of recovery is shown in Fig. 6.1. VNFs 2 and 3 are realized by VNF replicas 2.1-2.3 and 3.1-3.2, respectively. Each replica is allocated to a node. Different nodes are holding different replicas. VNFs 2.3 and 3.1 fail at a time. Two replicas, VNFs 2.4 and 2.5, are chosen from the replica

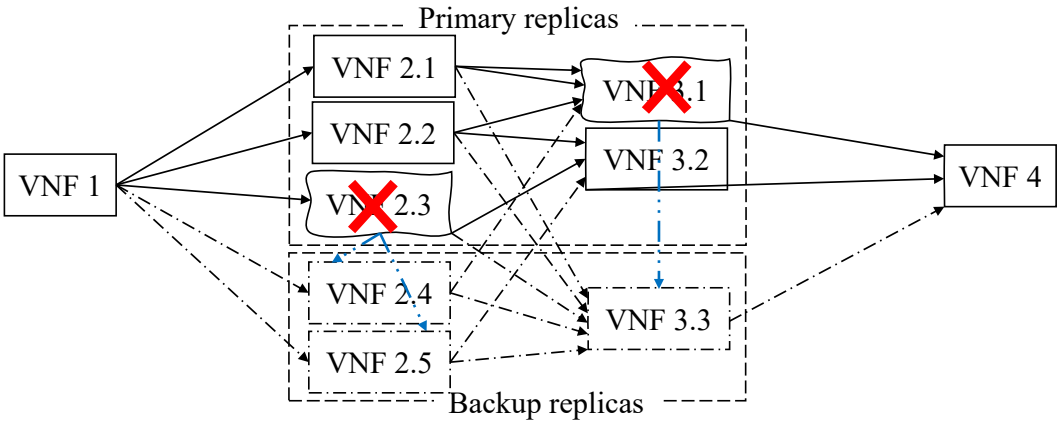


Figure 6.1: Example of the recovery. Red "X" means that a node which holds a replica is unavailable, i.e., a replica is unavailable. Load balancers are ignored in this figure.

pool of VNF 2 for backup VNFs. When the primary VNF 2.3 fails, VNF 2.3 is replaced with backup replicas VNFs 2.4 and 2.5. One replica, VNF 3.1, is chosen from the replica pool of VNF 3 for backup VNFs. When the primary VNF 3.1 fails, primary VNF 3.1 is replaced with backup VNF 3.3. As the blue lines in Fig. 3.1, the data of VNF 2.3 and VNF 3.1, which fail, need to be transferred to backup VNF replicas 2.4 and 2.5, and VNF 3.3, respectively. The E2E latency for recovering VNF 2 is determined by the transmission time from VNF 2.3 to VNF 2.4 and the transmission time from VNF 2.3 to VNF 2.5; the recovery time is the larger one of the above two transmission times. The E2E latency between the primary and backup instances of VNF 3 under the current failures is the transmission time between the primary VNF 3.1 and the backup VNF 3.3.

The rest of this chapter is organized as follows. Section 6.1 describes the model. Section 6.2 presents two approximate approaches to solve the proposed model. Section 6.3 presents numerical results that show the performances of the proposed model and the introduced approaches in different cases. Section 6.4 discusses the boundaries of resiliency level. Section 6.5 summarizes the key points of this paper and future directions.

6.1 Problem formalization

6.1.1 VNF allocation model

This work considers a directed graph $G(N, L)$, where N is a set of nodes and L is a set of directed links. Different nodes have different computational resources for supporting functions allocated to them. This work considers an abstract metric of different types of resources, such as CPU, memory, and storage, for each node. Node $n \in N$ has m_n units of available resources in total. Set $N_S \subseteq N$ stores the nodes that have available resources for function allocations, i.e., node $n \in N$ is included in N_S if $m_n > 0$. This work assumes that the required time for data forwarding, i.e. latency, between two nodes can be estimated [90], which is dependent on the network congestion and physical distance. The estimated time for data forwarding through directed link $(i, j) \in L$ is d_{ij}.

C is the set of SFCs waiting for provisions. Each SFC $c \in C$ is an ordered set of VNFs. F is the set of functions. $F^c \subseteq F$ is the ordered set of VNFs used in SFC $c \in C$. Binary given parameter ψ_{fc} is set to 1 if function $f \in F$ is included in SFC $c \in C$.

Each VNF $f \in F$ can be replaced by a set of replicas selected from pools of replica VNFs in terms of the processing ability, which is denoted by set $A_f^{[\bullet]}$ for primary usage ($\bullet$ is $\mathcal{P}$) and backup usage ($\bullet$ is $\mathcal{B}$). The capacity of each replica VNF can be different from each other associated with the same original VNF. Since the backup replicas may have special functions to synchronize states so that primary replicas can be backed up, the pools for primary and backup replicas may be different, e.g., the images for primary and backup replicas are different when the backups need to be initialized before becoming primary replicas [104], and the backup replicas request specific snapshots when the failures are so critical that the only remedy is to restore an earlier version of the system [105]. This work considers that an NFV orchestrator detects failures and notify them to the corresponding backup replicas. To increase the feasibility of function selection and placement, the replica pools for primary and backup replicas are separated, each of which can be given differently. Decision binary variable $x_{nfa}^{[\bullet]}$ is set to 1 if the $a \in [1, |A_f^{[\bullet]}|]$th primary replica ($\bullet$ is $\mathcal{P}$) or backup replica ($\bullet$ is $\mathcal{B}$) of VNF $f \in F$ is assigned to node $n \in N$; 0

otherwise.

This work assumes that a software problem that causes a node to become unavailable, where the VNFs running on the node also become unavailable for simplicity, which is the worst scenario due to the software problem. This work calls this unavailability a failure. Note that this work does not considers that some VNFs in a node fail, while others in the node survive, which should be addressed for further study. The node which hosts the failed VNFs has a copy of data and status information for stateful VNFs. This work assumes that the backup VNFs can fetch the data from the node. As an example, this work can consider a shared storage system which is independent from the failed VNF for holding the data of VNF as the system presented in [106]. The node failures do not affect data accessibility. A backup VNF cannot start to run until the data from the primary one are ready. k is a given integer parameter, which indicates the number of failed nodes. The k-resiliency backup strategy is applied in this paper. As long as there are k nodes failures, the proposed strategy guarantees that the VNF instances running on these nodes can be relocated to non-failed nodes without affecting other VNFs. Since there are k failures of nodes at the maximum in the network, this work uses set U to contain all possible combinations of k nodes from N_S. This work calls element $u \in U$, which is a set of nodes $u \subseteq N_S$, an *error pattern*.

$P_{fa}^{[\bullet]}$ represents the processing ability of the $a \in [1, |A_f^{[\bullet]}|]$th primary replica ($\bullet$ is $\mathcal{P}$) or backup replica ($\bullet$ is $\mathcal{B}$) of VNF $f \in F$. $S_{fa}^{[\bullet]}$ represents the resource requirement of the ath primary replica or backup replica of VNF $f \in F$.

R is the set of requests from users. Binary given parameter $\gamma_r^c, r \in R, c \in C$, is set to 1 if request r requests SFC c; 0 otherwise. The required processing ability for SFC $c \in C$ of request $r \in R$ is denoted by g_r^c, which is a given parameter.

Binary decision variable $p_{ij}^{faa'}, (i,j) \in L, f \in F, a \in A_f^{\mathcal{P}}, a' \in A_f^{\mathcal{B}}$ is set to 1 if the migration route from the ath primary replica of VNF f to the a'th backup replica of f includes link (i,j); 0 otherwise. $p_{ij}^{faa'}$ is defined by:

$\forall w \in N, f \in F, a \in A_f^{\mathcal{P}}, a' \in A_f^{\mathcal{B}}$,

$$\sum_{(i,j)\in L} \alpha_{ij}^{w} p_{ij}^{faa'} = \begin{cases} -1, \text{ if } x_{wfa}^{\mathcal{P}} = \sum_{n\in N} x_{nfa'}^{\mathcal{B}} = 1 \\ 1, \text{ if } x_{wfa'}^{\mathcal{B}} = \sum_{n\in N} x_{nfa}^{\mathcal{P}} = 1 \\ 0, \text{ otherwise,} \end{cases} \tag{6.1}$$

where α_{ij}^{w} is a given parameter that satisfies $\alpha_{ij}^{w} = 1$ if node w is the head of (i, j), i.e., $w = j$; $\alpha_{ij}^{w} = -1$ if node w is the tail of (i, j), i.e., $w = i$; $\alpha_{ij}^{w} = 0$ otherwise. $\sum_{n\in N} x_{nfa'}^{\mathcal{B}} = 1$ means that there is a backup replica of VNF $f \in F$ that uses the a'th replica in the pool; 0 otherwise. $\sum_{n\in N} x_{nfa}^{\mathcal{P}} = 1$ means that there is a primary replica of VNF $f \in F$ that uses the ath replica in the pool; 0 otherwise. Equation (6.1) can be summarized to:

$$\sum_{(i,j)\in L} \alpha_{ij}^{w} p_{ij}^{faa'} = -x_{wfa}^{\mathcal{P}} \sum_{n\in N} x_{nfa'}^{\mathcal{B}} + x_{wfa'}^{\mathcal{B}} \sum_{n\in N} x_{nfa}^{\mathcal{P}},$$
$$\forall w \in N, f \in F, a \in A_f^{\mathcal{P}}, a' \in A_f^{\mathcal{B}}. \tag{6.2}$$

Decision variable μ_u indicates the maximum sum of E2E latencies from the primary replicas to the backup replicas among all VNFs in all chains under error pattern $u \in U$, which is given by:

$$\mu_u = \max_{a\in A_f^{\mathcal{P}}, a'\in A_f^{\mathcal{B}}, f\in F^c, c\in C} \sum_{(i,j)\in L} \left\{ d_{ij} p_{ij}^{faa'} \sum_{n\in u} x_{nfa}^{\mathcal{P}} \sum_{n'\in N\setminus\{u\}} x_{n'fa'}^{\mathcal{B}} \right\}, \forall u \in U. \tag{6.3}$$

The lower bound of μ_u is 0. The upper bound of μ_u is the largest E2E latency between any two nodes among all the replicas.

The proposed model aims to minimize the sum of the maximum E2E latencies among functions for transferring data under all error patterns with k failed nodes; if there are multiple solutions, this work chooses the solution which has the minimum of the sum of the resource requirements of all VNF replicas among all error patterns. The objective function is given by:

$$\min \sum_{u\in U} \mu_u + \epsilon \sum_{n\in N} \sum_{f\in F} \left(\sum_{a\in A_f^{\mathcal{P}}} S_{fa}^{\mathcal{P}} x_{nfa}^{\mathcal{P}} + \sum_{a\in A_f^{\mathcal{B}}} S_{fa}^{\mathcal{B}} x_{nfa}^{\mathcal{B}} \right). \tag{6.4}$$

Equation (6.4) has the primary and secondary objectives prioritized by ϵ. The primary objective is to minimize the maximum E2E latencies among all considered VNFs and error patterns, i.e., $\sum_{u \in U} \mu_u$. The secondary objective is to minimize the sum of all consumed resources of primary and backup VNF replicas, i.e., $\sum_{n \in N} \sum_{f \in F} \left(\sum_{a \in A_f^{\mathcal{P}}} S_{fa}^{\mathcal{P}} x_{nfa}^{\mathcal{P}} + \sum_{a \in A_f^{\mathcal{B}}} S_{fa}^{\mathcal{B}} x_{nfa}^{\mathcal{B}} \right)$. ϵ is a sufficiently small value to prioritize the primary objective over the secondary objective so that the solution which has the minimum value of the secondary objective is chosen when there is more than one solution which has the minimum value of primary objective. The value of ϵ must be smaller than $\frac{\min_{(i,j) \in L} d_{ij}}{\sum_{f \in F} \left(\sum_{a \in A_f^{\mathcal{P}}} S_{fa}^{\mathcal{P}} + \sum_{a \in A_f^{\mathcal{B}}} S_{fa}^{\mathcal{B}} \right)}$ so that a change within the second term will not exceed the smallest unit within the first term.

The constraints in the optimization problem are as follows:

$$\sum_{f \in F} \left(\sum_{a \in A_f^{\mathcal{P}}} S_{fa}^{\mathcal{P}} x_{nfa}^{\mathcal{P}} + \sum_{a \in A_f^{\mathcal{B}}} S_{fa}^{\mathcal{B}} x_{nfa}^{\mathcal{B}} \right) \leq m_n, \forall n \in N, \tag{6.5}$$

$$\sum_{n \in N} \sum_{a \in A_f^{\mathcal{P}}} P_{fa}^{\mathcal{P}} x_{nfa}^{\mathcal{P}} \geq \sum_{c \in C} \psi_{fc} \sum_{r \in R} \gamma_r^c \cdot g_r^c, \forall f \in F, \tag{6.6}$$

$$\sum_{n \in N \setminus \{u\}} \left(\sum_{a \in A_f^{\mathcal{P}}} P_{fa}^{\mathcal{P}} x_{nfa}^{\mathcal{P}} + \sum_{a \in A_f^{\mathcal{B}}} P_{fa}^{\mathcal{B}} x_{nfa}^{\mathcal{B}} \right) \geq \sum_{c \in C} \psi_{fc} \cdot \sum_{r \in R} \gamma_r^c \cdot$$

$$g_r^c, \forall f \in F, u \in U, \tag{6.7}$$

$$\sum_{n \in N} x_{nfa}^{\mathcal{P}} \leq 1, \forall f \in F, a \in A_f^{\mathcal{P}}, \tag{6.8}$$

$$\sum_{n \in N} x_{nfa}^{\mathcal{B}} \leq 1, \forall f \in F, a \in A_f^{\mathcal{B}}, \tag{6.9}$$

Equation (6.5) ensures that the resources allocated to VNF instances do not exceed the available resources on each node. Equation (6.6) ensures that the primary replicas of each VNF meet the processing ability requirements of the requests. Equation (6.7) ensures that the primary and backup replicas of each VNF meet the processing ability requirements of the requests under all error patterns. Equations (6.8) ensures that each primary replica must be mapped at most once. Equations (6.9) ensures that each backup replica must be mapped at most once.

In the above equations, (6.2)-(6.4) are not linear. They are transformed into linear formulations according to Appendix A. Equation (6.2) is linearlized to (6.10a)-(6.10g). Equation (6.3) is linearlized to (6.11a)-(6.11i). For the sake of brevity and readability, let $(w, f, a, a') \in \Phi$, $(i, j, f, u, a, a') \in \Omega$, and $(f, u, a, a') \in \Xi$ denote $w \in N, f \in F, a \in A_f^{\mathcal{P}}, a' \in A_f^{\mathcal{B}}$, $(i, j) \in L, f \in F, a \in A_f^{\mathcal{P}}, a' \in A_f^{\mathcal{B}}, u \in U$, and $f \in F, u \in U, a \in A_f^{\mathcal{P}}, a' \in A_f^{\mathcal{B}}$, respectively.

$$\sum_{(i,j)\in L} \alpha_{ij}^{w} p_{ij}^{faa'} = \eta'_{wfaa'} - \eta_{wfaa'}, \forall (w, f, a, a') \in \Phi, \tag{6.10a}$$

$$\eta_{wfaa'} \leq x_{wfa}^{\mathcal{P}}, \forall (w, f, a, a') \in \Phi, \tag{6.10b}$$

$$\eta_{wfaa'} \leq \sum_{n\in N} x_{nfa'}^{\mathcal{B}}, \forall (w, f, a, a') \in \Phi, \tag{6.10c}$$

$$\eta_{wfaa'} \geq x_{wfa}^{\mathcal{P}} + \sum_{n\in N} x_{nfa'}^{\mathcal{B}} - 1, \forall (w, f, a, a') \in \Phi, \tag{6.10d}$$

$$\eta'_{wfaa'} \leq x_{wfa'}^{\mathcal{B}}, \forall (w, f, a, a') \in \Phi, \tag{6.10e}$$

$$\eta'_{wfaa'} \leq \sum_{n\in N} x_{nfa}^{\mathcal{P}}, \forall (w, f, a, a') \in \Phi, \tag{6.10f}$$

$$\eta'_{wfaa'} \geq x_{wfa'}^{\mathcal{B}} + \sum_{n\in N} x_{nfa}^{\mathcal{P}} - 1, \forall (w, f, a, a') \in \Phi, \tag{6.10g}$$

$$\theta_{ij}^{ufaa'} \leq p_{ij}^{faa'}, \forall (i, j, f, u, a, a') \in \Omega, \tag{6.11a}$$

$$\theta_{ij}^{ufaa'} \leq \sum_{n\in u} x_{nfa}^{\mathcal{P}}, \forall (i, j, f, u, a, a') \in \Omega, \tag{6.11b}$$

$$\theta_{ij}^{ufaa'} \leq \sum_{n'\in N\setminus\{u\}} x_{n'fa'}^{\mathcal{B}}, \forall (i, j, f, u, a, a') \in \Omega, \tag{6.11c}$$

$$\theta_{ij}^{ufaa'} \geq \sum_{n\in u} x_{nfa}^{\mathcal{P}} + \sum_{n'\in N\setminus\{u\}} x_{n'fa'}^{\mathcal{B}} - 2, \forall (i, j, f, u, a, a') \in \Omega, \tag{6.11d}$$

$$\mu_u \leq \sum_{(i,j)\in L} \{d_{ij}\theta_{ij}^{ufaa'}\} + B - \delta'_{aa'fu}B, \forall (f, u, a, a') \in \Xi, \tag{6.11e}$$

$$\mu_u \geq \sum_{(i,j)\in L} \{d_{ij}\theta_{ij}^{ufaa'}\} - B + \delta'_{aa'fu}B, \forall (f, u, a, a') \in \Xi, \tag{6.11f}$$

$$\sum_{(i,j)\in L} \{d_{ij}\theta_{ij}^{ufaa'}\} \geq (\delta'_{aa'fu} - 1)B + \sum_{(i,j)\in L} \{d_{ij}\theta_{ij}^{uf'a''a'''}\}, \forall (f, u, a, a') \in (F, U, A_f^{\mathcal{P}}, A_f^{\mathcal{B}}), (f', a'', a''') \in (F, A_f^{\mathcal{P}}, A_f^{\mathcal{B}})\setminus\{(f, a, a')\}, \tag{6.11g}$$

$$\sum_{f\in F} \sum_{a\in A_f^{\mathcal{P}}} \sum_{a'\in A_f^{\mathcal{B}}} \delta'_{aa'fu} = 1, \forall u \in U, \tag{6.11h}$$

$$\mu_u \geq \sum_{(i,j)\in L} \{d_{ij}\theta_{ij}^{ufaa'}\}, \forall (f,u,a,a') \in \Xi, \tag{6.11i}$$

$$\eta_{wfaa'}, \eta'_{wfaa'} \in \{0,1\}, \forall (w,f,a,a') \in \Phi, \tag{6.12a}$$

$$\theta_{ij}^{fnn'aa'} \in \{0,1\}, \forall (i,j,f,u,a,a') \in \Omega, n \in u, n' \in N \backslash \{u\}, \tag{6.12b}$$

$$\delta'_{aa'fu} \in \{0,1\}, \forall (f,u,a,a') \in \Xi. \tag{6.12c}$$

B is sufficiently large to ensure that its value is larger than $\sum_{(i,j)\in L} \left\{ d_{ij} \sum_{n\in u} \sum_{n'\in N\backslash\{u\}} \theta_{ij}^{fnn'aa'} \right\}, \forall (f,u,a,a') \in \Xi$.

In summary, the proposed model is formulated as an MILP problem:

$$\min \sum_{u\in U} \mu_u + \epsilon \sum_{n\in N} \sum_{f\in F} \left(\sum_{a\in A_f^{\mathcal{P}}} S_{fa}^{\mathcal{P}} x_{nfa}^{\mathcal{P}} + \sum_{a\in A_f^{\mathcal{B}}} S_{fa}^{\mathcal{B}} x_{nfa}^{\mathcal{B}} \right) \tag{6.13a}$$

$$\text{s.t. } (6.5) - (6.9), (6.10a) - (6.12c), \tag{6.13b}$$

$$p_{ij}^{faa'} \in \{0,1\}, \forall (i,j) \in L, f \in F, a \in A_f^{\mathcal{P}}, a' \in A_f^{\mathcal{B}}, \tag{6.13c}$$

$$x_{nfa}^{\mathcal{P}} \in \{0,1\}, \forall n \in N, f \in F, a \in A_f^{\mathcal{P}}, \tag{6.13d}$$

$$x_{nfa}^{\mathcal{B}} \in \{0,1\}, \forall n \in N, f \in F, a \in A_f^{\mathcal{B}}. \tag{6.13e}$$

It should be noted that, as U contains all possible combinations of k nodes from N_S, it is possible that the size of U becomes large depending on N_S and k, which may make the size of MILP problem rapidly increase.

6.1.2 NP-completeness

This work defines a subproblem of the VNF allocation problem (SVNFA) in the proposed model and prove that SVNFA is NP-complete [77, 78]. In the subproblem, this work focuses on the hardness of deciding whether there is a feasible solution that can provide the required processing ability under all error patterns with k node failures regardless of the latency and routing among replica VNFs. This work assumes that there is one VNF with its replica pool waiting to be allocated to satisfy the constraints of processing ability and node capacity in SVNFA. If SVNFA can be proved to be NP-complete, the decision version of original problem in the proposed model, which covers SVNFA, is

also NP-complete. Note that this work can show that the decision version of original problem is NP in the similar way to the proof that SVNFA is NP in Theorem 6.1, as described below.

Definition 6.1 *Given a set of nodes N with the capacity m_n of each node $n \in N$, and a pool of VNF replicas A, in which each replica has its processing ability $P_a, a \in A$, and resource requirement $S_a, a \in A$, each replica in the pools can be allocated at most once. Is it possible to find an allocation of replicas from the pools to satisfy the required processing ability g with ensuing k-resiliency ($k \geq 2$)?*

Lemma 6.1 *Let $\mathcal{A}_{nf}$ denote the total processing ability of the replicas of VNF $f \in F$ which are allocated to node $n \in N_S$; this work defines $A_f = \{A_{nf} | n \in N_S\}$. For VNF $f \in F$, if and only if the sum of the $|N_S| - k$ smallest elements in $\mathcal{A}_f$ is larger than or equal to the required ability g_f which is calculated by the right side of* (6.6)*, the allocation is k resilient for VNF f.*

Proof: This work proves Lemma 6.1 from both sides. If the sum of the $|N_S| - k$ smallest elements in $\mathcal{A}_f$ is larger than or equal to g_f, the sum of any $|N_S| - k$ elements in $\mathcal{A}_f$ is also larger than or equal to g_f, so that the allocation is k resilient for VNF f. If the allocation is k resilient for VNF f, any combination of $|N_S| - k$ elements in $\mathcal{A}_f$ must be larger than or equal to g_f, which includes the sum of the $|N_S| - k$ smallest elements in $\mathcal{A}_f$. Thus, Lemma 6.1 is true. ■

Theorem 6.1 *SVNFA is NP-complete.*

Proof: The SVNFA problem is NP, as this work can verify whether the allocation satisfies the required processing abilities with ensuing k-resiliency in polynomial-time $O(|A| + |N_S| \log |N_S| + k)$ according to Lemma 6.1. It takes $O(|A|)$ to obtain the processing abilities provide by each node. Finding the k smallest elements takes $O(|N_S| \log |N_S|)$. The time complexity of summing k numbers is $O(k)$.

This work presents that the partition problem, which is a known NP-complete problem [79], is polynomial time reducible to SVNFA. The partition problem is defined as: whether a given multi-set I of positive integers can be

partitioned into two subsets I^1 and I^2 such that the sum of the numbers in I^1 equals that in I^2.

First, this work constructs an instance of SVNFA from any instance of the partition problem. An instance of the partition problem consists of a set of positive integers $I = \{I_i : i \in [1, |I|]\}$. An instance of SVNFA is constructed with the following steps.

1. Consider a set of $2k$ nodes. The capacity of each node is set to $\frac{1}{2}\sum_{I_i \in I} I_i$, i.e., $m_n = \frac{1}{2}\sum_{I_i \in I} I_i, \forall n \in N$.

2. This work sets $|A| = |I| + 2k - 2$. A consists of two disjoint subsets, A' and A''. This work sets $A' = I$. A'' has $2k - 2$ replicas whose processing abilities and resource requirements are $\frac{1}{2}\sum_{I_i \in I} I_i$. The processing ability of each replica $P_a, a \in A'$, is $I_i \in I$ and the resource requirement of each replica $S_a, a \in A'$, is also $I_i \in I$.

3. The required processing ability g is given by $\frac{k}{2}\sum_{I_i \in I} I_i$.

The presented construction has a polynomial complexity of $O(|I| + k)$, which transforms any instance of the partition problem into an instance of SVNFA.

Consider that a partition problem instance is a Yes instance, which indicates that there exist two subsets of I^1 and I^2 with $\sum_{I_i \in I^1} I_i = \sum_{I_i \in I^2} I_i$. By using the presented construction to define the corresponding SVNFA instance from any Yes instance of the partition problem, each replica is assigned to a node with a requirement of I_i units resources and presents I_i units processing ability. The replicas with processing ability $I_i \in I^1$ are allocated to one node. The replicas with processing ability $I_i \in I^2$ are allocated to another node. Each replica with processing ability $\frac{1}{2}\sum_{I_i \in I} I_i$ is allocated to a node separately. The sum of resource requirements of the replicas on each node is $\sum_{I_i \in I^1} I_i = \sum_{I_i \in I^2} I_i = \frac{1}{2}\sum_{I_i \in I} I_i$, which satisfies the resource constraint. The sum of providable processing ability of replicas over any k nodes is $k\sum_{I_i \in I^1} I_i = k\sum_{I_i \in I^2} I_i = \frac{k}{2}\sum_{I_i \in I} I_i$, which is equal to the required processing ability, i.e, ensuring k-resiliency. The SVNFA instance is a Yes instance.

Conversely, this work shows that, if an SVNFA instance is a Yes instance, the corresponding partition problem instance is a Yes instance. Since the

SVNFA instance is a Yes instance, the allocation of replicas in the pools ensures k-resiliency and satisfies the node capacity constraint. The sum of processing abilities over any k nodes is no less than the required processing ability $\frac{k}{2}\sum_{I_i \in I} I_i$. The sum of resource requirements of the replicas on each node is no more than $\frac{1}{2}\sum_{I_i \in I} I_i$. For the $2k-2$ nodes with $\frac{1}{2}\sum_{I_i \in I} I_i$ units capacities, the replicas with processing abilities $\frac{1}{2}\sum_{I_i \in I} I_i$ are allocated to them separately with satisfying the resource requirement constraint, since the resource requirements of them are equal to the node capacities, which are $\frac{1}{2}\sum_{I_i \in I} I_i$. For the other two nodes, to ensure that each node has no less than $\frac{1}{2}\sum_{I_i \in I} I_i$ units processing ability, the sum of processing abilities of the replicas allocated to one node must be $\frac{1}{2}\sum_{I_i \in I} I_i$. All the processing abilities of the replicas allocated to the same node must belong to the same set, I^1 or I^2. The processing abilities in set I are partitioned into two subsets I^1 and I^2 such that the sum of the numbers in I^1 equals that in I^2, which is $\frac{1}{2}\sum_{I_i \in I} I_i$. Thus, if an SVNFA instance is a Yes instance, the corresponding partition problem instance is also a Yes instance, i.e., $\sum_{I_i \in I^1} I_i = \sum_{I_i \in I^2} I_i$. ■

6.2 Heuristic approaches

As the size of the problem presented in Section 6.1 increases, the problem becomes difficult to solve in practical time. A feasible solution may not be obtained within admissible computational time, which can be specified by SPs. This work introduces two approaches, greedy approach and probabilistic heuristic approach, in this section.

The greedy approach finds a local optimal solution by traversal, which is introduced in Section 6.2.1. Section 6.2.2 analyzes the greedy algorithm the optimality of the provided solutions in a special cas. Based on the initial solution given by the greedy algorithm, Section 6.2.3 provides a probabilistic heuristic approach to explore a better solution.

6.2.1 Greedy approach

A greedy approach is a simple heuristic approach which finds the best decision at each step aiming to find a near optimal solution. Once the deci-

sion is made at each step, it is not changed at the latter steps. The greedy approach traverses all the nodes for placing each function and find a suitable location, where the deployment does not violate any constraints. The greedy approach chooses the position for each replica of each VNF, which has the minimum objective value at this step. Finally, the approach gets a final allocation for all VNFs. The greedy approach employs Algorithm 6.1. In lines 2–22, this algorithm allocates the primary replicas for each function with satisfying the requirement of required processing ability. In lines 3–5, this algorithm checks if there is any feasible solution under the given condition by (6.6). In lines 19–21, if the allocated primary replicas cannot satisfy the required processing ability after traversing all replicas and nodes, there is no feasible solution provided by Algorithm 6.1. In lines 23–41, this algorithm allocates the backup replicas for each function with satisfying the requirement of required processing ability. In lines 38–40, this algorithm checks if the allocated replicas ensure k-resiliency. The time complexity of Algorithm 6.1 is $O\Big(|F|\Big(|C||R| + \max_{f\in F}|A_f^{\mathcal{P}}|\log(\max_{f\in F}|A_f^{\mathcal{P}}|) + \max_{f\in F}|A_f^{\mathcal{B}}|\log(\max_{f\in F}|A_f^{\mathcal{B}}|) + |N|\log|N|\big(\max_{f\in F}|A_f^{\mathcal{P}}| + |L|\max_{f\in F}|A_f^{\mathcal{B}}|\big)\Big)\Big)$.

The greedy approach has two limitations as follows. Firstly, the approach gives priority to satisfy the constraints, such as k-resiliency, rather than seeking the optimal solution (line 28), which leads to the performance degradation of the greedy approach. Secondly, in order to avoid the verification of k-resiliency in factorial time, this algorithm uses a conservative verification method (lines 39–41), which may lead to the false negative, i.e., the infeasible solution obtained by the greedy approach may be feasible.

Algorithm 6.1 Algorithm used in greedy approach

Input: Given parameters in Section 6.1

Output: allocation for selected replicas

1: Calculate the required processing ability g_f for each function $f \in F$ by using the right side of (6.6).
2: **for** $f \in F$ **do**
3: **if** $\sum_{a\in A_f^{\mathcal{P}}} P_{fa}^{\mathcal{P}} < g_f$ **then**
4: **return** Infeasible.
5: **end if**

For the replicas whose processing abilities are larger than or equal to g_f, sort $A_f^{\mathcal{P}}$ according to $P_{fa}^{\mathcal{P}}, a \in A_f^{\mathcal{P}}$, non-decreasingly; for the other replicas, sort $A_f^{\mathcal{P}}$ according to $P_{fa}^{\mathcal{P}}, a \in A_f^{\mathcal{P}}$, non-increasingly and put them after the previous replicas.
for $a \in A_f^{\mathcal{P}}$ **do**
Sort N_S according to the providable processing ability of each node, non-decreasingly.
for $n \in N_S$ **do**
if there is enough capacity on n for a **then**
Allocate replica a to node n.
Break
end if
end for
if allocated replicas for VNF f can provide g_f units processing ability **then**
Combine the remaining replicas in $A_f^{\mathcal{P}}$ to $A_f^{\mathcal{B}}$.
Break
end if
end for
if allocated replicas for VNF f cannot provide g_f units processing ability **then**
return Infeasible.
end if
end for
for $f \in F$ **do**
For the replicas whose processing abilities are larger than or equal to g_f, sort $A_f^{\mathcal{B}}$ according to $P_{fa}^{\mathcal{B}}, a \in A_f^{\mathcal{B}}$, non-decreasingly; for the other replicas, sort $A_f^{\mathcal{B}}$ according to $P_{fa}^{\mathcal{B}}, a \in A_f^{\mathcal{B}}$, non-increasingly and put them after the previous replicas. The original primary replicas are allocated before the backup replicas.
for $a \in A_f^{\mathcal{B}}$ **do**
Let N_f denote the nodes where the primary replicas of function f are allocated.
Sort N_S according to the providable processing ability of each non-

decreasingly. If there are nodes have the same processing ability, sort them according to the highest latency among all nodes in N_f, increasingly.

for $n \in N_S$ **do**
  if there is enough capacity on n for a **then**
    Allocate replica a to node n
    Break
  end if
end for
if $|N_S| - k$ nodes with minimum processing abilities can provide g_f units processing ability **then**
  Break
end if
end for
if $|N_S| - k$ nodes with minimum processing abilities cannot provide g_f units processing ability **then**
return Infeasible.
end if
end for

6.2.2 Analysis of greedy approach in a special case

The greedy approach is difficult to analyze under a general case in the aspect of approximate ratio, since the E2E latencies, processing abilities, and required resources of the replicas are given arbitrarily. This work discusses the optimality of the solutions provided by the greedy approach in terms of the first term in the objective function under the special case that the capacity of each node in N_S is larger than or equal to $\sum_{f \in F} \sum_{a \in A_f^{\mathcal{P}}} S_{fa}^{\mathcal{P}}$.

Observation 6.1 *The first term in the objective function is zero under two cases. One is that the backup replicas are not allocated and multiple primary replica with sufficient processing ability can achieve k-fault tolerance without any latency due to backup migration. The other one is that the primary and backup replicas of the same VNF are allocated to the same node so that there does not exist migration latency.*

Observation 6.2 *If the primary and backup replicas of the same VNF are*

allocated to the same node, the resiliency level of the VNF is zero.

Observation 6.3 *If the required resiliency level of a VNF is zero, only primary replicas are enough for required processing ability according to* (6.6).

Property 6.1 *Let* $\mathcal{L}$ *be the maximum latency between two nodes in graph* G. *The first term in* (6.4) *of the feasible solution provided by Algorithm 6.1 is at most* $\max_{f \in F}(\binom{|N_S|}{k} - \mathcal{X}_f)\mathcal{L}$ *under the special case.* $\mathcal{X}_f$ *is an integer value, which is calculated by:*

$$\mathcal{X}_f = \begin{cases} 0, \ \textit{if } |N_S| - |A_f^{\mathcal{P}}| < k, \\ \binom{|N_S|-|A_f^{\mathcal{P}}|}{k}, \ \textit{otherwise} \end{cases} \quad , \forall f \in F. \tag{6.14}$$

Proof: If only the primary replicas are allocated, the first term in (6.4) is zero. Since the work cares about the upper bound of the E2E latency, this work discusses the situation that both primary and backup replicas are allocated in the proof. Algorithm 6.1 allocates all primary replicas before the backup replicas. When $|N_S| \leq |A_f^{\mathcal{P}}|$, at least one primary replica is allocated to each node. If a node fails, the E2E latency between from the primary replica to a backup replica is at most $\mathcal{L}$. The E2E latency between the primary and backup instances of VNF $f \in F$ is at most $\binom{|N_S|}{k}\mathcal{L}$.

When $|N_S| > |A_f^{\mathcal{P}}|$, primary replicas are not allocated to $|N_S| - |A_f^{\mathcal{P}}|$ nodes and backup replicas are preferentially allocated to these nodes. If $|N_S| - |A_f^{P}| < k$, any error pattern causes the failures of primary replicas and the E2E latency between the primary and backup instances of VNF $f \in F$ is at most $\binom{|N_S|}{k}\mathcal{L}$. Otherwise, only backup replicas fail among $\binom{|N_S|-|A_f^{\mathcal{P}}|}{k}$ error patterns and the replicas do not need to be recovered. The E2E latency between the primary and backup instances of VNF $f \in F$ is at most $(\binom{|N_S|}{k} - \binom{|N_S|-|A_f^{\mathcal{P}}|}{k})\mathcal{L}$. ■

Property 6.2 *Let the replica pools of VNFs be sorted in line 6 of Algorithm 6.1. When the first term of the feasible solution provided by Algorithm 6.1 is zero under the special case, the second term in* (6.4) *of the solution is at most* $\sum_{f \in F} \sum_{a \in [1, \mathcal{Y}_f]} S_{fa}^{\mathcal{P}}$ *under the special case, where* $\mathcal{Y}_f, f \in F$, *equals*

to o^f_{k+1}*, if* o^f_{k+1} *exists;* $|A^{\mathcal{P}}_f|$*, otherwise.* $o^f_i, f \in F$*, is defined by:*

$$o^f_i = \begin{cases} 0, \ \textit{if } i = 0, \\ \underset{\mathcal{O} \in [o^f_{i-1}+1, |A^{\mathcal{P}}_f|]}{\operatorname{argmin}} \sum_{a \in [o^f_{i-1}+1]} , \mathcal{O}] P^{\mathcal{P}}_{fa} \geq g_f, \ \textit{otherwise.} \end{cases} \tag{6.15}$$

Proof: Let $\mathcal{K}_f$ define the number of primary replicas of VNF $f \in F$ whose processing abilities are not less than g_f. If $\mathcal{K}_f \geq k+1$, the first $k+1$ replicas in $A^{\mathcal{P}}_f$ are enough for k-fault tolerance of VNF $f \in F$, whose total resource requirements are $\sum_{a \in [1,k+1]} S^{\mathcal{P}}_{fa}$. If $\mathcal{K}_f < k+1$ for VNF $f \in F$, the primary replicas of VNF $f \in F$ whose processing abilities are less than g_r need to be allocated. The total processing ability provided by $(o^f_{i-1}+1)$th - o^f_ith replicas of VNF $f \in F$ is not less than the required one. For each $i \in [1, k+1]$, if the above replicas are allocated to the same node which no replicas are allocated to, the allocation satisfies k-fault tolerance of VNF $f \in F$. The number of replicas using for satisfying k-fault tolerance of VNF $f \in F$ is o^f_{k+1}, whose total resource requirements are $\sum_{a \in [1, o^f_{k+1}]} S^{\mathcal{P}}_{fa}$. In Algorithm 6.1, the number of allocated replicas is not larger than o^f_{k+1} as well as the required capacities according to the suitable combination of replicas with different processing abilities. The total required capacity, i.e., the second term in (6.4), is at most $\sum_{f \in F} \sum_{a \in [1, \mathcal{Y}_f]} S^{\mathcal{P}}_{fa}$. ■

6.2.3 Probabilistic heuristic approach

The greedy approach can achieve the optimal solution in the special case introduced in Section 6.2.2. The optimal solution has the minimum total recovery latency among all error patterns in a general case. The time complexity of traversing all errors is not in polynomial time. Different VNFs have different required processing abilities and replica pools. The selection of replicas in limited capacity cases and the order of allocating VNFs are not handled in the greedy approach. Since the greedy approach has the above limitations on obtaining better solutions, a probabilistic heuristic (PH) approach based on simulated annealing algorithm [107] is introduced in this subsection, which can improve the accuracy of solution with the cost of longer computation time. The parameters used in Algorithm 6.2 are shown in Table 6.1.

Table 6.1: Parameters in Algorithm 6.2.

Parameter	Description
IT	Initial temperature
RT	Minimum temperature limit
CF	Cooling factor, between 0 and 1
MC	Markov chain length

The PH approach obtains the initial solution based on the greedy approach. There are two ways for generating new solutions: selecting an allocation randomly, changing the allocation to a random location or deleting the allocation; selecting two allocations randomly, exchanging their locations. If the generated solution exceeds the capacity limitation on a node, an allocation on the node is randomly selected and removed until the capacity limitation is not exceeded. If the objective value calculated from the newly generated solution is better than the currently best one and the k-fault tolerance is satisfied, the best result is updated and the following procedure is based on the new solution; otherwise, the new solution is used for the following procedure with a probability related to the quality of the solution and the current temperature. The above procedure is repeated with $\boldsymbol{MC}$ cycles under each temperature. The PH approach ends when the temperature achieves $\boldsymbol{RT}$. If the difference between the values of RT and IT is small, the solution space will not be searched sufficiently; if the difference is large, the algorithm will take longer to end. A balance needs to be examined empirically in the choice of parameters. This work determines such a small value of RT that our obtained solution is not affected by RT.

Algorithm 6.2 Algorithm used in PH approach

Input: Parameters in Table 6.1, given conditions in Section 6.1

Output: Allocations of replicas

1: Calculate an initial allocation S_{N} obtained by Algorithm 6.1.
2: $S_{\mathrm{C}} \leftarrow S_{\mathrm{N}}, S_{\mathrm{B}} \leftarrow S_{\mathrm{N}}$
3: $O_{\mathrm{B}} \leftarrow$ Objective value calculated from S_{B}, $O_{\mathrm{C}} \leftarrow O_{\mathrm{B}}$
4: $\boldsymbol{T} \leftarrow IT$

while $T \geq RT$ **do**
  for 1 to MC **do**
    $r \leftarrow$ a randomly generated value between 0 and 1.
    if $r \geq 0.5$ **then**
      Randomly select a primary or backup replica and a location $\in [0, |N_S|]$.
      if Location 0 is selected **then**
        Remove the allocation of the selected replica if exists, sorted the solution as S_N.
      else
        Change the allocation of the selected replica to the selected location, sorted the solution as S_N.
      end if
    else
      Randomly select two primary or backup replicas
      Exchange the location of two selected allocation, sorted the solution as S_N.
    end if
    Calculate the required capacity of each node based on S_N.
    for $n \in N_S$ **do**
      while the required capacity of node n by the current allocation $> m_n$ **do**
        Release an allocation on node n randomly.
      end while
    end for
    $O_N \leftarrow$ Objective value calculated from S_N
    if $O_N < O_C$ and S_N satisfies the required k-fault tolerance **then**
      $O_C \leftarrow O_N, S_C \leftarrow S_N$.
      if $O_N < O_B$ **then**
        $O_B \leftarrow O_N, S_B \leftarrow S_N$.
      end if
    else
      if $r < \frac{O_C - O_N}{T}$ **then**
        $O_C \leftarrow O_N, S_C \leftarrow S_N$.

34: **else**
35: $O_{\mathrm{N}} \leftarrow O_{\mathrm{C}}$.
36: **end if**
37: **end if**
38: **end for**
39: $T \leftarrow T \cdot CF$.
40: **end while**
41: Return $S_{\mathrm{B}}, O_{\mathrm{B}}$.

6.3 Evaluations

This section prepares four tests. Test 1 compares the proposed model to five baseline models with different k in terms of the objective value obtained by (6.4). Test 2 compares the objective values obtained by the proposed model with different given parameters to investigate the relationship between k and each given parameter. Test 3 evaluates the impacts of different replica pools. Test 4 evaluates the performances of the approaches introduced in Section 6.2.

The MILP approach of the proposed model is solved by the IBM® ILOG® CPLEX® Interactive Optimizer, version 12.10.0, running on an Intel Core i7-7700 3.60 GHz 4-core CPU, 32 GB memory. The considered baseline models and the approximate approaches introduced in Section 6.2 are implemented by Python 3.8.5 and run on the same hardware.

6.3.1 Comparison between proposed model and baseline models

In the objective function of the proposed model, this work focuses on the latency and the number of deployed replicas. Each of the five baseline models represents different random and greedy algorithms with different concerns. Each baseline focuses on only one objective instead of two of the proposed models. The five baseline models that select the suitable replicas and backups from the pools and determine the locations of them are introduced as follows:

- Random selection and random placement with a central backup server (RSRP-CBS): data are re-fetched from the central storage server whose

location is determined randomly. RSPR-CBS does not take the network configuration into consideration. The replicas of each VNF are randomly selected from the pool and placed.

- Random selection and random placement with distributed backup servers (RSRP-DBS): data are re-fetched from the corresponding backup servers. RSPR-DBS does not take the network configuration into consideration. The primary and backup replicas of each VNF are randomly selected from the pool and placed.

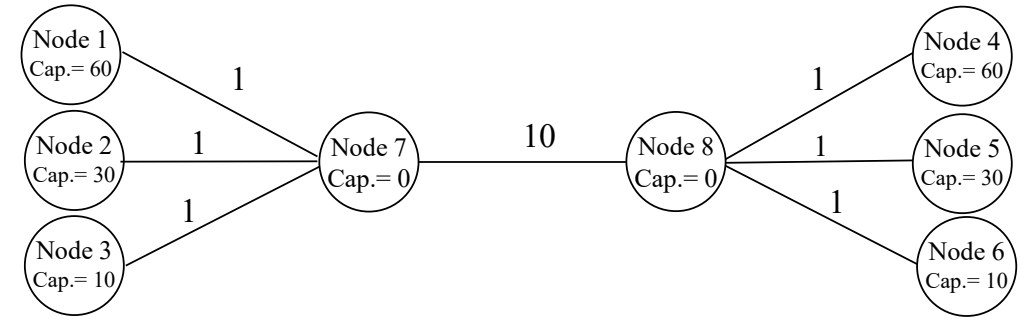
- Selection to minimize the number of instances for each VNF and random placement (minISRP): data are re-fetched from the corresponding backup servers. minISRP does not take the network configuration into consideration. It chooses as few as possible primary and backup replicas for each VNF. The placement of them are randomly determined.

- Selection to maximize the number of instances selection for each VNF and random placement (maxISRP): data are re-fetched from the corresponding backup servers. maxISRP does not take the network configuration into consideration. It chooses as many as possible primary and backup replicas for each VNF. The placement of them is randomly determined.

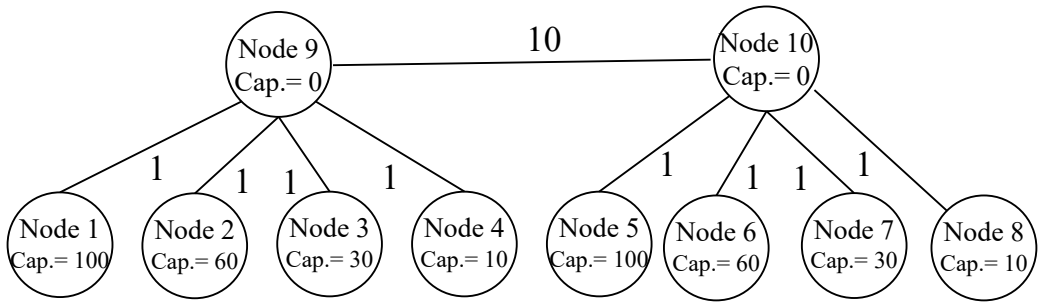
- Random selection and placement to minimize the distance of each VNF replica instances and corresponding backups (RSminDP): data are re-fetched from the corresponding backup servers. RSminDP takes the network configuration into consideration. The primary and backup replicas are randomly chosen from the pool of each VNF. Each backup is placed as close to its primary as possible.

RSRP-CBS , RSRP-DBS and RSminDP focus on the placements of functions instead of how to select replicas from the replica pools. minISRP and maxISRP are concerned with how to select replicas from the replica pools but not how to place the selected functions.

Note that, the work in [10] aims to reduce the inherent cost due to diversity (overhead) and redundancy (backup resources), which is different from our



(a) 8-node directed graph.



(b) 10-node directed graph.

Figure 6.2: Graphs used for evaluations. Each link is bi-directional. Each number attached to a link means the estimated latency for the link.

objective. This work cannot directly compare two models with different objective functions. As a complement, the work gives the five baseline models that consider the same objective function as the proposed model. Among them, minISRP has a similar idea to [10], which considers both reliability assurance and minimum cost deployment, i.e., the number of chosen replicas.

The settings of the cases in test 1 are given as follows. Without loss of generality, this evaluation uses arbitrary units. In case 1.1, the evaluation uses the directed graph in Fig. 6.2 (a). There are two SFCs and two requests in this case. The required processing abilities of the requests are two and three, respectively. The ordered function sets of two SFCs are $\{1, 2\}$, and $\{2, 3\}$, respectively. The sizes of the replica pools for all functions are three. The processing abilities of the replicas of three functions are [1, 1, 2], [2, 3, 5], and [1, 1, 2], respectively. The required capacities of the replicas of three functions are [2, 2, 3], [3, 4, 6], and [2, 2, 3], respectively. In case 1.2, the evaluation

uses the directed graph in Fig. 6.2 (b) and the other parameters are the same as those in case 1.1.

The result of the MILP approach of the proposed model and the average results of the five baseline models over 100 trials are shown in Table 6.2(a). The evaluation only counts the feasible solutions provided by the baseline models into the calculation of average results. Since the baseline models do not consider the k-resiliency and the remaining processing ability of the functions are not enough under the k-node failures according to the allocation provided by the baseline models, the requirements may not be satisfied. This work calls such an allocation an unaccepted allocation. This work uses α_k in Table 6.2(a) to represent the unaccepted ratios of the requirements under different error patterns whose total number of failures is k, which is calculated by $\frac{\text{the number of unaccepted allocations among all error patterns in } U}{|U|}$.

Table 6.2(a) shows that the proposed model ensures k-resiliency, i.e., the unaccepted ratios are 0. Since RSRP-CBS has one central storage server, the unaccepted ratio is always larger than zero and increases with the increase of k. RSRP-DBS has several backup locations so that the E2E latency is smaller than that of RSRP-CBS when k is larger than one at the cost of higher unaccepted ratios compared with RSRP-CBS. The number of function instances used in minISRP is chosen to be as small as possible. The number of function instances used in maxISRP is chosen to be as large as possible. The unaccepted ratios of these two models are smaller than those of RSRP models with smaller objective values. maxISRP has smaller unaccepted ratios than minISRP since more instances take up more nodes which more likely fail. minISRP has smaller objective values than maxISRP since the backup replicas are distributed widely in the network and they reduce the average distance between primary replicas and backup replicas. RSminDP has the smallest objective value among the proposed and five baseline models. However, it cannot guarantee k-resiliency, and the unaccepted ratios are higher than all the other baseline models in the examined cases. With the increase of k, the unaccepted ratio increases rapidly.

Table 6.2: Objective values and unaccepted ratios obtained by proposed and baseline models.

(a) Test 1: objective values (former) and unaccepted ratios (latter) calculated by baseline models under different k in cases 1.1 and 1.2

Model	Case 1.1		
	$k = 1$	k=2	$k = 3$
Proposed model	2.028, 0.000	68.037, 0.000	188.048, 0.000
RSRP-CBS	24.51, 0.167	75.08, 0.333	89.97, 0.500
RSRP-DBS	28.83, 0.108	63.89, 0.355	47.51, 0.634
minISRP	20.58, 0.115	59.02, 0.373	51.94, 0.633
maxISRP	36.42, 0.008	96.86, 0.104	70.60, 0.360
RSminDP	1.706, 0.237	6.066, 0.493	5.006, 0.727
Model	Case 1.2		
Proposed model	2.028, 0.000	26.037, 0.000	452.048, 0.000
RSRP-CBS	28.71, 0.125	139.1, 0.250	292.2, 0.375
RSRP-DBS	31.55, 0.093	137.5, 0.224	235.4, 0.405
minISRP	24.72, 0.055	109.1, 0.208	208.7, 0.432
maxISRP	47.76, 0.004	202.4, 0.026	334.8, 0.132
RSminDP	4.026, 0.136	16.59, 0.345	32.41, 0.464

(b) Test 2: objective values obtained by proposed model. "-" means no feasible solution.

Case / k	1.1	1.2	2.1	2.2	2.3	2.4	2.5
1	2.028	2.028	2.029	0.021	0.021	0.021	0.019
2	68.037	26.037	94.038	18.033	10.030	0.039	0.035
3	188.048	452.048	-	-	118.044	80.048	0.053
4	-	-	-	-	134.048	98.060	60.062
5	-	-	-	-	-	36.078	24.072
6	-	-	-	-	-	-	-

6.3.2 Relationship between resiliency level k and given parameters

In addition to the test cases in test 1, this evaluation gives five more cases for this evaluation. In case 2.1, the evaluation sets the capacities of nodes 1-8 to 6, 3, 10, 6, 3, 10, 0, 0, respectively. The other parameters are the same as those in case 1.1. In case 2.2, the evaluation considers one SFC, whose ordered function set is {1, 2, 3}, and one request, whose required processing ability is two. The other parameters are the same as those in case 2.1. In case 2.3, the evaluation considers one SFC, whose ordered function set is {1, 2, 3}, and one request, whose required processing ability is two. The other parameters are the same as those in case 1.1. In case 2.4, the evaluation increases the processing abilities of the replicas of functions 1 and 3 to [2, 3, 5]. The other parameters are the same as those in case 2.3. In case 2.5, the evaluation enlarges the size of the replica pools of three functions. The processing abilities of the replicas of three functions are [1, 2, 2, 5, 5], [2, 3, 3, 5, 5], and [1, 2, 2, 5, 5], respectively. The other parameters are the same as those in case 2.3.

Table 6.2(b) observes that several conditions increase the largest resiliency (k) under which the proposed model can have feasible solutions based on the examined cases. This work calls the largest k under which a feasible solution can be obtained by the proposed model in a case, a *resiliency level*. In the comparison between cases 1.1 and 1.2, the increase of the number of available nodes cannot provide a higher resiliency level if the total available resources of nodes are enough for all the replicas both before and after the increase. In the comparison between cases 1.1 and 2.1, the increase of the capacity of each node can provide a higher resiliency level if the currently available resources of nodes are not enough for all the replicas. In the comparison among cases 1.1, 2.1, 2.2, and 2.3, the reduction of the number of required resources from the requests can provide a higher resiliency level. In the comparison among cases 2.3, 2.4, and 2.5, the increase of the size of replica pools and the processing ability of each replica can increase the resiliency level if the currently available resources of nodes are enough for all the replicas.

6.3.3 Impact of replica pool design in terms of objective value

The replica pools in the given condition is designed by SPs. SPs provide the processing ability and resource requirement of each replica in the replica pools. Different replica pools may lead to different allocations and objective values, as shown in Table 6.2. In this subsection, this work shows the impact of replica pools with different features on the objective values of the proposed model. This evaluation aims to give directions for SPs on the design of replicas pools in order to obtain better objective values. The features of primary and backup replica pools which this work focuses on include the average values of the processing abilities of the replicas in the pools, the variances of the processing abilities of the replicas in the pools, and the relationship between the processing ability of each replica and the required processing ability.

This work designs different test cases for the evaluation. In these test cases, this work uses a directed graph in Fig. 6.2 (a). This work considers one SFC whose ordered VNF set is {1,2,3}, and one request whose required processing ability is five. The value of resource requirements is the same as that of processing ability for each replica. The settings of replica pools are shown in Table 6.3.

The evaluation results are shown in Table 6.3. In the examined cases, if the average processing abilities in the pools are the same, the replica pool with the smallest variance of processing abilities has the lowest objective value. However, as shown in case 3.5, a replica pool with the smallest variance of processing abilities may lead to decreasing the resiliency level, since the smallest variance of processing abilities means that the types of replicas are limited and the nodes with limited resources may not be able to hold enough replicas for backup. As well, the computation time is the shortest when the replica pools with the smallest variance of processing abilities are given, since the processing abilities and resource requirements of replicas are the same and the selection of replicas is omitted. If the variances of processing abilities are the same, the replica pool with the largest average processing ability has the lowest objective value and the shortest computation time.

If the resources of nodes are sufficiently large to accommodate all repli-

cas, it is recommended that SPs provide the replicas with the same processing ability and the processing ability should be as large as possible. Otherwise, if the average processing abilities are larger than the required one, the variance of the processing abilities should be as small as possible; if the average processing abilities are less than the required one, it is better to provide more replicas whose processing abilities can reach the required one in the comparison between cases 3.7 and 3.9.

Compared with the other models which do not consider the diversity of VNFs, the proposed model is more suitable for the scenarios where the capacities of the nodes in the cluster are requested to be fully utilized, e.g., the resource allocation in the edge computing network.

6.3.4 Comparison of MILP and approaches in Section 6.2

This subsection compares the approaches introduced in Section 6.2 with the MILP approach in terms of the objective values and computation times. This work performs the comparisons under cases 1.1-1.2 and cases 2.1-2.5. The parameter settings for the PH approach used in this evaluation are: $IT = 10000$; $RT = 1$; $CF = 0.98$; $MC = 100$. The results are shown in Table 6.4.

In the examined cases, the three approaches, two approaches in Section 6.2 and the MILP approach, can obtain feasible solutions. The number of solvable cases of the MILP approach, the greedy approach, and the PH approach are 24, 21, and 21, respectively. The average computation times of the MILP approach, the greedy approach, and the PH approach among 21 cases are 35670.29 [s], 0.0042 [s], and 28.0167 [s], respectively; the objective values of the MILP approach, the greedy approach, and the PH approach among 21 cases are 59.942, 98.704, and 87.659, respectively. This work can observe that the greedy approach is the fastest approach among these three approaches. The PH approach is more exact than the greedy approach. Note that the greedy and PH approaches cannot get feasible solutions in some cases, where the capacities of nodes are relatively small and the required resiliency level is relatively high.

The greedy approach reduces 99.98% computation time compared with the MILP approach on average among 21 test cases. The difference in the

Table 6.3: Objective values obtained by proposed model under different settings of replica pools. "-" means no feasible solution.

Case	Processing abilities	Average	Variance	Objective values					Computation time [s]				
				$k = 1$	$k = 2$	$k = 3$	$k = 4$	$k = 5$	$k = 1$	$k = 2$	$k = 3$	$k = 4$	$k = 5$
3.1	3, 4, 5, 6, 7	5	1.4	0.033	0.075	0.075	60.084	-	18.9700	35716.1	48685.8	76143.2	-
3.2	5, 5, 5, 5, 5	5	0.0	0.030	0.045	0.060	0.075	-	8.13000	40.5400	56.2800	226.170	-
3.3	1, 1, 5, 8, 8	5	3.1	0.039	0.063	88.078	108.102	-	14.8800	270.380	83212.8	20247.1	-
3.4	5, 6, 7, 8, 9	7	1.4	0.033	0.054	0.078	0.105	-	10.1900	100.120	1306.94	549.760	-
3.5	7, 7, 7, 7, 7	7	0.0	0.042	0.063	0.084	-	-	8.59000	53.9600	65.8800	-	-
3.6	1, 1, 5, 13, 13	7	5.4	0.054	0.093	88.108	134.147	-	11.0000	82.2000	12508.7	1770.59	-
3.7	1, 2, 3, 4, 5	3	1.4	0.027	0.042	110.051	134.059	-	26.0800	2978.43	249696	755039	-
3.8	3, 3, 3, 3, 3	3	0.0	0.027	0.036	0.045	60.0540	-	9.25000	62.1700	69.4100	6225.21	-
3.9	1, 1, 3, 5, 5	3	1.8	0.030	0.045	88.060	108.069	-	31.1400	478.560	133244	246275	-

objective value between the results obtained by the MILP approach and the greedy approach is 31.96% on average among 21 test cases. The difference in the objective value between the results obtained by the MILP approach and the PH approach is 23.76% on average among 21 test cases.

In summary, the greedy approach can reduce the computation time at the cost of performance loss when k is relatively small; the greedy approach may not be able to provide feasible solutions, otherwise. The PH approach performs better than the greedy approach at the cost of longer computation time. If this work considers that the accuracy of solution is the first priority, this work uses the MILP approach as long as the computation time is allowed. If the computation time of MILP is not allowed or the computation cannot be achieved due to the memory constraint, this work needs to adopt a suitable heuristic such as the greedy or PH approach. When an SP selects a suitable heuristic, it should consider the differences of objectives between the MILP and greedy approaches and between the MILP and PH approaches, where the differences are 31.96% and 23.76%, respectively, with taking an advantage of computation time of each heuristic. The selection of greedy and PH approaches can be made considering the two aspects: computation time and accuracy of solution.

6.4 Discussion on boundaries of resiliency level

The resiliency level is a key point for the service providers to evaluate the ability of service to resist failures. However, this work can only get the accurate resiliency level by calculating the optimal allocation based on different k under the given conditions and check if there is any feasible solution, which is time and resource consuming. This work gives approaches to estimate the boundaries of the resiliency level in this subsection.

This work qualitatively observed the relationship between resiliency level and given conditions in Section 6.3. Furthermore, this work discusses the upper and lower boundaries of the maximum resiliency level which can be reached under the given conditions including the node capacities, VNF pools,

Table 6.4: Objective values obtained by MILP approach and two approaches introduced in Section 6.2 with respective computation time. “-" means no feasible solution.

(a) Objective values.

k	Case 1.1			Case 1.2			Case 2.1			Case 2.2		
	MILP	Greedy	PH	MILP	Greedy	PH	MILP	Greedy	PH	MILP	Greedy	PH
1	2.028	4.026	2.031	2.028	4.026	2.028	2.029	36.026	26.030	0.021	0.021	0.021
2	68.037	144.040	76.045	26.037	194.042	114.046	94.038	-	-	18.033	124.033	110.033
3	188.048	228.052	228.052	452.048	552.052	552.052	-	-	-	-	-	-

k	Case 2.3			Case 2.4			Case 2.5		
	MILP	Greedy	PH	MILP	Greedy	PH	MILP	Greedy	PH
1	0.021	0.021	0.021	0.021	0.021	0.021	0.019	0.019	0.019
2	10.030	60.030	58.033	0.039	0.039	0.039	0.035	0.035	0.035
3	118.044	176.038	176.037	80.048	196.042	154.068	0.053	0.053	0.053
4	134.048	144.048	144.048	98.060	150.060	138.062	60.062	60.069	60.067
5	-	-	-	36.078	-	-	24.072	-	-

(b) Computation times [s].

k	Case 1.1			Case 1.2			Case 2.1			Case 2.2		
	MILP	Greedy	PH	MILP	Greedy	PH	MILP	Greedy	PH	MILP	Greedy	PH
1	25.13	0.0044	7.3280	39.55	0.0010	12.1412	18.27	0.0010	3.7487	1.44	0.0010	2.9018
2	603.41	0.0040	17.9595	2350.13	0.0032	50.9420	1155.22	-	-	315.69	0.0040	8.1250
3	6026.50	0.0146	34.4165	716331.42	0.0305	123.1822	-	-	-	-	-	-

k	Case 2.3			Case 2.4			Case 2.5		
	MILP	Greedy	PH	MILP	Greedy	PH	MILP	Greedy	PH
1	1.66	0.0020	5.5360	1.22	0.0010	4.9958	4.81	0.0010	11.6418
2	140.39	0.0010	23.5640	24.45	0.0017	19.7506	255.69	0.0010	19.9479
3	1681.27	0.0017	34.2160	2674.87	0.0013	47.4689	676.98	0.0038	38.7976
4	83.02	0.0048	24.3760	2132.97	0.0020	34.3816	15687.15	0.0040	62.9296
5	-	-	-	122.97	-	-	1773.23	-	-

and required abilities.

This work uses the following notations in this subsection. A set of VNFs F is given with required processing ability g_f for VNF $f \in F$, which is calculated by the right side of (6.6). Each replica of VNF $f \in F$ is selected from the corresponding replica pool, $A_f^{[\bullet]}$, for primary ($\bullet$ is $\mathcal{P}$) or backup ($\bullet$ is $\mathcal{B}$) VNFs. A replica has its capacity requirement $S_{fa}^{[\bullet]}, a \in A_f^{[\bullet]}, f \in F$, and processing ability $P_{fa}^{[\bullet]}$. If the replica is a primary one, $\bullet$ is $\mathcal{P}$; if the replica is a backup one, $\bullet$ is $\mathcal{B}$.

Before giving the boundaries of the maximum resiliency level, this work gives the following assumptions.

Assumption 1 (sufficient processing ability in pool): This work assumes that the processing abilities of the primary replicas in the pool satisfy the required processing ability for each VNF, i.e., $\sum_{a \in A_f^{\mathcal{P}}} P_{fa}^{\mathcal{P}} \geq g_f, \forall f \in F$.

If Assumption 1 is untenable, there is no feasible solution under the given conditions.

Assumption 2 (relationship between capacity and processing ability of a replica): This work assumes that the required resource of a replica = overhead + $\mathcal{F}$(processing ability of the replica), in which function $\mathcal{F}$ is a non-decreasing function on the processing ability of the replica. The higher processing ability of a replica has, the more resources it requires.

Assumption 2 ensures that, under the same requirement of processing ability, the deployment of one replica with larger processing ability does not require larger capacity than that of more than one replica with smaller processing abilities.

There are some impossible cases when not all the replicas are allocated, e.g., all the primary replicas are not allocated, which is against (6.6). These cases are not considered.

Lemma 6.2 *Regardless of the limitation of node capacities, for a VNF, if this work allocates all the replicas in the pool, its resiliency level is not less than that of other allocations, in each of which only a part of replicas in the pool are allocated.*

Proof: If only a part of replicas in the pool are allocated, the allocation of one more replica does not reduce the resiliency level. Adding another replica

may increase the resiliency level. For example, a replica of VNF $f \in F$ whose processing ability is larger than g_f is allocated to a node where no replica of this VNF has been allocated. In conclusion, the allocations of all the replicas in the pool provide not less resiliency level than the resiliency level of the allocations in which a part of replicas in the pool are allocated. ■

This work uses ξ^{P}_{fn} to denote the sum of processing abilities of replicas of VNF $f \in F$, which are allocated to node $n \in N$.

Lemma 6.3 *Let N_f be an ordered set of nodes with replicas of VNF f, which is sorted by a non-increasing order of $\xi^{P}_{fn}, n \in N_S, f \in F$. For VNF $f \in F$, if the allocated replicas are determined, the allocation with the lowest value of $\sum_{n \in N_S \setminus \{|N_S|\}} \sum_{n' \in [n+1, |N_S|]} (\xi^{P}_{fn} - \xi^{P}_{fn'})$, i.e., allocating all the replicas in the pools to the nodes evenly so that the processing capacity of each node can be balanced as well as possible, provides not less resiliency level than other allocations.*

Proof: This proof considers a situation that the total processing ability of VNFs on each node of a given allocation satisfies the minimum of $\sum_{n \in N_{\mathrm{S}} \setminus \{|N_{\mathrm{S}}|\}} \sum_{n' \in [n+1, |N_{\mathrm{S}}|]} (\xi^{\mathrm{P}}_{fn} - \xi^{\mathrm{P}}_{fn'})$. The proof considers the resiliency level of the allocation as k, which is given by maximizing k under the constraint $\sum_{n \in [k+1, |N_f|]} \xi^{\mathrm{P}}_{fn} \geq g_f, \forall f \in F$.

This work uses the proof by contradiction. This proof assumes that there is another allocation that can provide a resiliency level which is higher than k. The allocation with higher k can be obtained by moving a replica based on the current allocation. If this work moves a replica from one node to another node, the value of $\sum_{n \in N_{\mathrm{S}} \setminus \{|N_{\mathrm{S}}|\}} \sum_{n' \in [n+1, |N_{\mathrm{S}}|]} (\xi^{\mathrm{P}}_{fn} - \xi^{\mathrm{P}}_{fn'})$ does not decrease since the value before moving the replica is the minimum in the considered situation.

Let $\xi^{\mathrm{P}\prime}_{fn}$ denote the sum of processing abilities of replicas of VNF $f \in F$, which are allocated to node $n \in N$ after the moving of the replica. Let N'_f be an ordered set of nodes with replicas of VNF f, which is re-sorted by a non-increasing order of $\xi^{\mathrm{P}\prime}_{fn}, n \in N_{\mathrm{S}}, f \in F$. This work considers the resiliency level of the allocation after the moving as k', which is given by maximizing k' under the constraint $\sum_{n \in [k'+1, |N_f|]} \xi^{\mathrm{P}\prime}_{fn} \geq g_f, \forall f \in F$.

After the moving of a replica and the re-sorting of the processing ability of each node, $\sum_{n \in [k+1, |N_f|]} \xi^{\mathrm{P}\prime}_{fn}$ is no more than $\sum_{n \in [k+1, |N_f|]} \xi^{\mathrm{P}}_{fn}$ as well as $\sum_{n \in [1,k]} \xi^{\mathrm{P}\prime}_{fn}$ is no less than $\sum_{n \in [1,k]} \xi^{\mathrm{P}}_{fn}$. k' is no more than k, i.e., the re-

siliency level of the allocation after the moving is no more than that of the allocation before the moving, which contradicts the assumption that there is another allocation can provide a resiliency level which is higher than k. Thus, the allocation with the lowest value of $\sum_{n \in N_S \setminus \{1\}} (\xi^{\mathrm{P}}_{fn} - \xi^{\mathrm{P}}_{f,n-1})$ provides not less resiliency level than other allocations. ■

6.4.1 Lower bound of resiliency level

Let $P^{\mathcal{P}}_f$ and $P^{\mathcal{B}}_f$ be the sets of processing abilities of primary and backup replicas in the pool, respectively. First, this work sorts the elements in $P^{\mathcal{P}}_f \cup P^{\mathcal{B}}_f, f \in F$, which is a set of the processing abilities of replicas in the primary and backup pools of VNF $f \in F$, by a non-decreasing order. Let P_f denote the ordered set of the processing abilities of replicas $P^{\mathcal{P}}_f \cup P^{\mathcal{B}}_f, f \in F$. This work obtains the value of $\xi^{\mathrm{P}}_{fn}, f \in F, n \in N$, by following the strategies in Lemmas 6.2 and 6.3, i.e., allocating all the replicas in the pools so that the processing capacity of each node can be balanced as well as possible.

Theorem 6.2 (Lower bound related to the replica pools) *If there are sufficient capacities of the nodes for the given replicas, a lower bound of the maximum resiliency level under the given conditions can be given by maximizing k under the constraint of $g_f \leq \sum_{n \in [1,k]} \xi^{P}_{fn} \leq \sum_{n \in [k+1,|N_f|]} \xi^{P}_{fn}, \forall f \in F$.*

Proof: This proof divides the processing abilities of primary and backup replicas and split them into two parts; the former part of the set has k elements while the latter part has $|N_f| - k$ elements. The proof considers that the nodes corresponds to k elements in the former part fail and the processing abilities of the nodes corresponds to the $|N_f| - k$ elements in the latter part that can replace the failed nodes and provide the sufficient required processing abilities. When the number of the elements in the latter part is smaller than that of the former part, i.e., $|N_f| - k < k$, the remaining VNFs must not reach the required processing abilities, since $\xi^{\mathrm{P}}_{fn}, n \in N$ is in a non-increasing order for VNF $f \in F$.

The larger k is, the more nodes fail with the more unavailable processing abilities. The worst case corresponds to the situation that the maximum k nodes fail while the total processing abilities of remaining $|N_f| - k$ VNFs can

reach the required processing abilities. The processing ability provided by the available replicas must be larger than the required processing ability for each VNF for a feasible solution. Thus, a lower bound of the maximum resiliency level under the given conditions is given by: $\max k$ under the constraint of $g_f \leq \sum_{n\in[1,k]} \xi^{\mathrm{P}}_{fn} \leq \sum_{n\in[k+1,|N_f|]} \xi^{\mathrm{P}}_{fn}, \forall f \in F$, if node $n \in N_f$ can provide ξ^{P}_{fn} processing ability for VNF $f \in F$ under the constraint of node capacity. ■

6.4.2 Upper bound of resiliency level

Theorem 6.3 *Regardless of the limitation of node capacities, this work allocates all the replicas in the pools with minimizing* $\max_{n\in N} \xi^{P}_{fn}$ *for each VNF* $f \in F$. *Such an allocation gives an upper bound of the maximum resiliency level under the given conditions, which is obtained by maximizing* k *under the constraint* $\sum_{n\in[k+1,|N_f|]} \xi^{P}_{fn} \geq g_f, \forall f \in F$.

Proof: According to Lemmas 6.2 and 6.3, the allocation stated in this theorem gives the highest resiliency level under the given conditions. For each VNF, the proof assumes that k nodes with the highest providable processing ability fail. If the providable processing ability after the failures is equal to or larger than the required processing ability for each VNF, the resiliency level is k. The maximum resiliency level that can reach is the maximum resiliency level under the given conditions. ■

6.4.3 Examples

Seven cases are evaluated in Section 6.3. Their maximum resiliency levels are shown in Table 6.2. This work calculates the upper and lower boundaries by Theorems 6.2 and 6.3 and compare them with their actual maximum resiliency levels as examples. The results are listed in Table 6.5. The accurate values are between the upper bounds and lower bounds. The lower bound cannot be estimated when the capacities of nodes are sufficiently small, e.g., cases 2.1 and 2.2.

Table 6.5: Comparison between accurate resiliency level provided in Section 6.3 and boundaries of resiliency level calculated by Theorems 6.2 and 6.3

Case	1.1	1.2	2.1	2.2	2.3	2.4	2.5
Lower bound	2	2	N/A	N/A	2	2	3
Accurate value	3	3	2	3	4	5	5
Upper bound	3	3	3	5	5	5	5

N/A: not applicable.

6.5 Summary

This chapter proposed a k-resilient VNF allocation model for reducing the E2E latency during recovery migration with VNF diversity and redundancy. This work formulated the proposed model as an MILP problem. This work proved that the subproblem of the VNF allocation problem in the proposed model is NP-complete. Numerical results showed that the proposed model reduces the E2E latencies between the primary replicas and the backup replicas, compared with five baseline models. This work developed and analyzed two approximate algorithms to solve the proposed model in a shorter computation time on average at the cost of accuracy compared with the MILP approach. In the examined cases, the greedy approach reduces 99.98% computation time compared with the MILP approach at the cost of 31.96% performance loss on average. The greedy and PH approaches become valuable to be applied as the problem size increases. The performance loss of the PH approach is lower than that of the greedy approach, which is 23.76% on average in the examined cases. This work evaluated the impact of replica pool design and gave the suggestions to SPs for better allocations. This work investigated the relationship between the given conditions and resiliency level. This work derived the theorems to give the upper and lower bounds of the resiliency level. This work showed the practical methods to manage the replicas with the required amount of processing ability and recover the data after the failures.

Chapter 7

Fault-tolerant resource allocation model considering diversity for dynamic requests based on reinforcement learning

This chapter proposes a VNF allocation model to maximize the number of accepted requests which ensures the required resiliency levels by choosing suitable replicas of VNFs from the corresponding pools and allocates them using RL method to suitable locations for dynamic requests from users [108].

The rest of this chapter is organized as follows. Section 7.1 presents MDP for handling the dynamic requests and the proposed model of the VNF deployment problem. Section 7.2 introduces an RL-based approach for solving the proposed model. Section 7.3 presents the evaluations that show the performance of the proposed model compared with the two baseline models in different cases. Section 7.4 summarizes the key points of this chapter.

7.1 Problem formalization and Model description

This work introduces the dynamic NFV system in Section 7.1.1. The features of network form dynamically arriving requests are captured by MDP described in Section 7.1.2. For the accepted requests, the proposed model of the VNF allocation problem is introduced in Section 7.1.3.

7.1.1 Dynamic network function virtualization system

An NFV system provides network functions by using the VNF deployed in the physical or virtual machines. This chapter calls the physical or virtual machines for VNF deployments *nodes*. The set of nodes is denoted by N. Each node has a list of providable resources for VNFs, e.g., central processing unit (CPU), memory, and storage. Without loss of generality, this work considers one type of resource in this paper. The resource has a limitation in each node, which is called the *capacity* of the node and denoted by m_n for node $n \in N$.

VNFs take up the resources of the node where they are located and provide services with a certain amount of ability, e.g., throughput for deep package inspection and firewall. Let F denote the set of providable VNFs by SPs. A_f denotes a replica pool of VNF $f \in F$. Different replicas have different processing abilities or resource requirements. Replica a in pool A_f is represented by A_{fa}, whose providable processing ability is P_{fa} with a requirement of resource S_{fa}.

A dynamic system receives the service requests in sequence over time. R represents the set of all received requests. Each request contains a set of VNFs and the required processing ability for each VNF. If VNF $f \in F$ is selected by request $r \in R$, g_{rf} is set to the required processing ability from request r; otherwise, 0. F_r is a set that contains the functions requested by $r \in R$, which is defined by $F_r = \{f \in F | g_{rf} > 0\}$. The number of instances which are instantiated from replica $a \in A_f$ of function $f \in F_r$ for request $r \in R$ has an upper bound Δ_{rfa}. Δ_{rfa} is a non-negative integer and can be infinite if the upper bound is not required. If the system can provide enough processing abilities by allocating replica instances within the given limitations

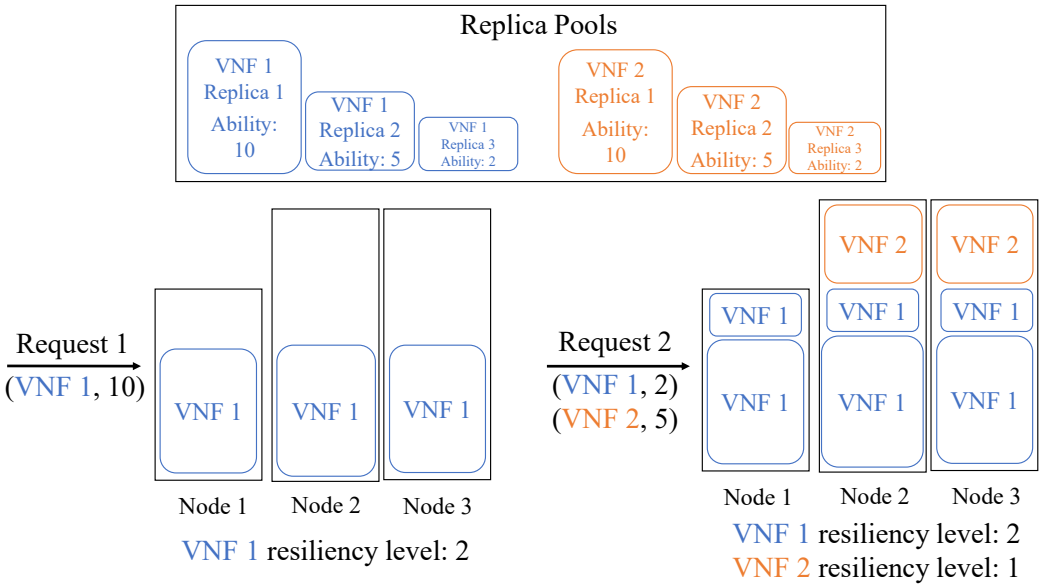


Figure 7.1: Example of VNF allocations with dynamic requests. (VNF x, y) means the request includes VNF x with required processing ability y.

on numbers in a request to suitable locations, the request can be accepted; otherwise, the request is rejected. Request $r \in R$ arrives stochastically with required processing ability for each VNF, time to live (TTL) δ_r, priority ρ_r, and required resiliency level k_r. The TTL is the duration of a request. The priority is an integer value which has a relationship with the importance of request, arrival time, urgency, etc.; the larger ρ_r is, the higher the priority of allocating request $r \in R$ is. The system is considered to accept as many requests as possible with considering their priorities, and then improve the resiliency of the VNF allocations. An example is shown in Fig. 7.1.

7.1.2 Markov decision process for handling real-rime requests

For handling the network variations caused by the dynamic requests in sequence, this section introduces the concept of *time slot*. T denotes the ordered set of time slots. The last element in T is the current time slot. A time slot is a time duration whose length is decided by SPs considering the types of

services, e.g., one hour for long-lived services or one second for time-sensitive services. At each time slot, the system performs the following actions in order: deleting the processing ability requirements from timeout requests; receiving arriving requests; ordering the received requests by their priorities and arriving time; checking if each request can be accepted; deciding the VNF allocations; updating the network state.

Non-negative integer decision variable x_{tn}^{rfa} denotes the number of replica instances instantiated from replica a in pool $A_f, f \in F$, which are allocated to node $n \in N$ at time slot $t \in T$. This work assumes that the creation, deletion, and migration of the replicas can be completed in one time slot, which can be handled by automated management tools.

Request $r \in R$ arrives at time slot $v_r \in T$. TTL δ_r of request $r \in R$ is defined as the integer multiple of the length of a time slot. If binary variable $w_{rt} = 1$, request $r \in R$ is active at time slot $t \in T$; 0, otherwise. w_{rt} is given by: $\forall t \in T, r \in R$,

$$w_{rt} = \begin{cases} 1, v_r \leq t \leq v_r + \delta_r, \\ 0, \text{otherwise.} \end{cases} \tag{7.1}$$

Let set R_t represent the set of active requests at time slot $t \in T$, which is defined by $R_t = \{r \in R | w_{rt} = 1\}$.

MDP is described by tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$ in which $\mathcal{S}$ is the discrete state set, $\mathcal{A}$ is the discrete action set, $\mathcal{P}$ is the state transition probability, $\mathcal{R}$ is the reward function, $\gamma \in [0, 1]$ is the discount factor, which is used to calculate the cumulative reward.

State definition

State $s \in \mathcal{S}$ is defined as a vector $(\widehat{M}_s, \widehat{A}_s, \widehat{R}_s, \widehat{P}_s, \widehat{c}_s, \widehat{t}_s, \widehat{v}_s, \widehat{\delta}_s)$. $\widehat{M}_s = (\widehat{M}_s^1, \cdots, \widehat{M}_s^{|N|})$ represents the remaining resource of each node. $\widehat{A}_s = (\widehat{A}_s^1, \cdots, \widehat{A}_s^{|F||N|})$ represents the providable abilities of all VNFs for the currently being processed request on each node. $\widehat{R}_s = (\widehat{R}_s^1, \cdots, \widehat{R}_s^{|F|})$ represents the required ability for each VNF of currently being processed. $\widehat{P}_s = (\widehat{P}_s^1, \cdots, \widehat{P}_s^{|F|})$ represents the providable ability for each VNF of current allocating replica. $\widehat{c}_s$ represents the required capacity of current allocating replica. $\widehat{t}_s$ represents the current time

slot. $\widehat{v}_s$ represents the arriving time slot of currently being processed request. $\widehat{\delta}_s$ represents the TTL of currently being processed request.

Action definition

An action indicates an allocation of a replica of a VNF to a node, which is chosen from a discrete set. This work labels the nodes in set N with an integer index $1, 2, \cdots, |N|$. Action $\alpha \in \mathcal{A}$ is a non-negative integer, $\mathcal{A} = \{0, 1, 2, \cdots, |N|\}$. $\alpha = 0$ represents the case that VNF is not allocated to any node; otherwise, α denotes the specific index of node in N, which means that the agent tries to allocate the current replica to the αth node. The size of action set is $|N| + 1$.

Reward definition

The reward is related to the number of accepted requests. The mathematical formulation of the reward function is given in Section 7.1.3. The reward shaping in the RL approach is described in Section 7.2.2.

State transition

A state transition is defined as $(s, \alpha_s, r_s, s_{\text{next}})$, where $s \in \mathcal{S}$ is the current network state, $\alpha_s \in \mathcal{A}$ is the action taken for allocating a replica instance for request $r_s \in R$ at state s, r_s is the currently being processed at state s, and $s_{\text{next}} \in \mathcal{S}$ is the next network state after taking the action. The processing procedure is introduced in Section 7.2.2.

7.1.3 VNF allocation model for accepted requests

The resiliency and fault tolerance of VNFs are formulated as follows. Binary decision variable κ^k_{rft} is set to one if VNF $f \in F_r$ ensures the required processing ability under any k fault nodes for request $r \in R_t$ at time slot $t \in T$, and zero otherwise. Set U_k contains all possible combinations of k nodes from N. This work calls element $u \in U$, which is a set of nodes $u \subseteq N$, an *error pattern.* κ^k_{rft}

is defined by: $\forall k \in [0, |N|], f \in F_r, r \in R_t, t \in T$,

$$\begin{aligned} &\textbf{If } \textstyle\sum_{n \in N \setminus u} \sum_{a \in A_f} x_{tn}^{rfa} P_{fa} \geq g_{rf}, \forall u \in U_k \quad \textbf{then} \\ &\quad \kappa_{rft}^k = 1 \\ &\textbf{Else} \\ &\quad \kappa_{rft}^k = 0 \end{aligned} \tag{7.2a}$$

$$\kappa_{rft}^k \in \{0, 1\}. \tag{7.2b}$$

Non-negative integer decision variable ι_{rft} denotes the *resiliency level* of function $f \in F$ for request $r \in R_t$ at time slot $t \in T$. If the resiliency level of a function is k, the processing ability of the function is ensured under at most k node failures, which is defined by:

$$\iota_{rft} = \max_{k \in [0,|N|]} k \kappa_{rft}^k, \forall f \in F_r, t \in T, r \in R_t. \tag{7.3}$$

Non-negative integer decision variable ω_r denotes the *resiliency level* of request $r \in R$, which is the minimum resiliency level among all required functions during the request active time slots. ω_r is given by:

$$\omega_r = \min_{f \in F_r, t \in [v_r, v_r + \delta_r]} \iota_{rft}, \forall r \in R. \tag{7.4}$$

Let $R_t^{\mathbb{A}}$ represent the set of accepted active requests at time slot $t \in T$. If the required resiliency level, k_r, of request $r \in R_t$ is satisfied, the request is accepted, which is represented by:

$$\omega_r \geq k_r, \forall r \in R_t^{\mathbb{A}}, t \in T. \tag{7.5}$$

Let $R_t^{\mathbb{N}}$ denote the set of newly arrival requests at time slot $t \in T$, which is defined by $R_t^{\mathbb{N}} = \{r \in R | v_r = t\}$. Let $R_t^{\mathbb{AN}}$ denote the set of accepted requests that arrive at time slot $t \in T$, which is defined by $R_t^{\mathbb{AN}} = \{r \in R_t^{\mathbb{A}} | v_r = t\}$.

The objective is to maximize the sum of priorities of accepted requests, which is expressed by:

$$\max \sum_{r \in R_t^{\mathbb{AN}}} \sum_{t \in T} \rho_r. \tag{7.6}$$

The diversity of VNFs is mapped as different replica instances in the model. The constraints about the replica instances are as follows:

$$\sum_{n \in N} x_{tn}^{rfa} \leq \Delta_{rfa}, \forall f \in F_r, a \in A_f, r \in R_t, t \in T, \tag{7.7}$$

$$\sum_{n\in N}\sum_{a\in A_f} x_{tn}^{rfa} P_{fa} \leq g_{rf}, \forall r \in R_t^{\mathbb{A}}, f \in F_r, t \in T, \tag{7.8}$$

$$\sum_{r\in R_t^{\mathbb{A}}}\sum_{f\in F_r}\sum_{a\in A_f} x_{tn}^{rfa} S_{fa} \leq m_n, \forall n \in N, t \in T. \tag{7.9}$$

Equation (7.7) ensures that the number of instances of a replica does not exceed the given limitation. If Δ_{rfa} is infinite, (7.7) is omitted. Equation (7.8) ensures that the replica instances of each VNF meet the processing ability requirements of the requests. Equation (7.9) ensures that the resources allocated to replica instances do not exceed the available resources on each node at each time slot.

In summary, the proposed model is formulated by:

$$\max \sum_{t\in T}\sum_{r\in R_t^{\mathbb{AN}}} \rho_r \tag{7.10a}$$

$$\text{s.t. } (7.1)-(7.5), (7.7)-(7.9), \tag{7.10b}$$

$$x_{tn}^{fa} \in \{0,1\}, \forall n \in N, f \in F, a \in A^f, t \in T. \tag{7.10c}$$

7.2 Policy gradient based deep reinforcement learning approach

7.2.1 Overall structure

With MDP introduced in Section 7.1.2, the network state transitions caused by the allocation of replica instances and the variations of dynamic requests can be captured. However, the state transition probabilities in $\mathcal{P}$ are unknown for us. This work needs an appropriate VNF replica instance allocation policy to choose the suitable actions in each state so as to achieve a higher expected reward. Thus, this work addresses the RL-VNFA approach to handle the VNF allocation problem confronting dynamic requests with ensuring the required processing abilities, which is modeled in Section 7.1.3.

The architecture of the RL-VNFA approach is shown in Fig. 7.2. The RL-VNFA agent observes the state information from the RL-VNFA environment and automatically selects an action suggested by a policy as a return. After the action is taken, the RL-VNFA environment returns the reward and the state to the agent. The agent updates related policies according to the reward

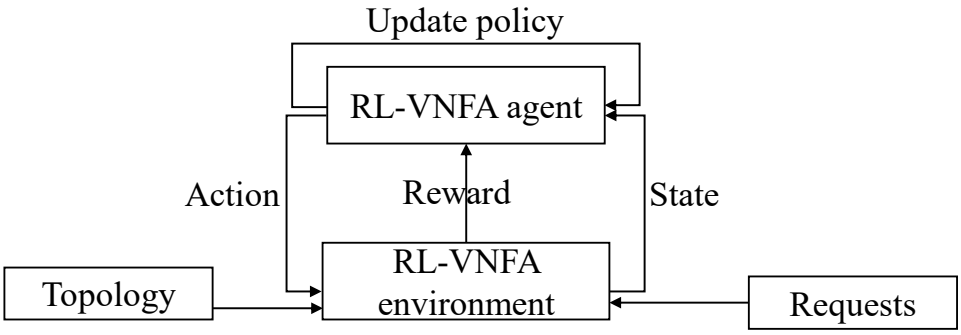


Figure 7.2: Structure of reinforcement-learning based approach.

by a policy gradient (PG) based reinforcement learning algorithm described in Section 7.2.3. The policy defines the agent's behavior at a given time. The above procedure is repeated until the environment returns a finish signal or the procedure repeats a given number of loops.

The policy in the agent is designed as a multi-layer fully connected DNN [109] based on the backpropagation neural network [110], which has an input layer, several hidden layers, and an output layer as shown in Fig 7.3. The input layer is the state vector and the output layer is the actions' probability distribution. The number of hidden layers is related to the number of nodes and the size of replica pools. More input features require more hidden layers. This work chooses the rectified linear unit (ReLU) as the activation function for hidden layers, which introduces non-linear features to the neural network. In terms of deciding the numbers of neurons in hidden layers, this work adopts an empirical equation in [29]: $N_h = \sqrt{N_s \cdot N_a} + \beta$ in which N_h is the width of a hidden layer, N_s is the width of the input layer of the hidden layer, N_a is the width of the output layer of the hidden layer, and β is a constant in $[0, 10]$ for adjusting the performance of the backpropagation network.

The input data is normalized to 0-1 range in order to make the neural network easy to be trained [111]. In order to keep the quantitative relations of input data such as the resource utilization and processing ability requirements, this work compresses the input data in each state $s \in \mathcal{S}$ with a constant. For the input data related to the capacity, the element $\widehat{c}_s$ and the elements in $\widehat{M}_s$ are divided by $\max_{n \in N} m_n$. For the input data related to the processing ability, the elements in $\widehat{A}_s$, $\widehat{R}_s$, and $\widehat{P}_s$ are divided by an estimated constant which is

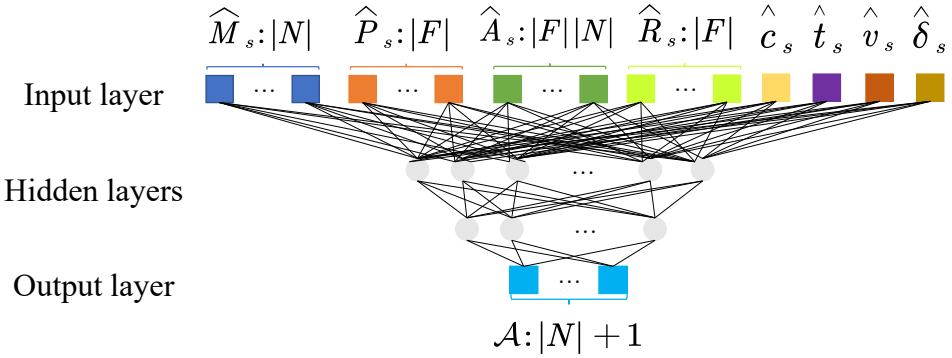


Figure 7.3: Example of neural network design with two hidden layers. A Value after ":" is the number of nodes.

equal to or larger than $\max\{\max_{f\in F,t\in T}\sum_{r\in R_t} g_f^r, \max_{f\in F,a\in A_f} S_{fa}\}$. For the input data related to the time slots, $\widehat{t}_s$, $\widehat{v}_s$, and $\widehat{\delta}_s$ are divided by the number of time slots $|T|$.

7.2.2 Adapt to dynamic requests

SPs have a list of type of available services $\mathscr{R}$. A user requests a service from $\mathscr{R}$ as a request. The arrival rate of type $r \in \mathscr{R}$ is λ_r. The service rate of type $r \in \mathscr{R}$ is μ_r. This work considers that the requests in R are the instances of $\mathscr{R}$. The requests in R which request the same service have the same VNFs with required processing abilities. The arrival interval of requests which are the instances of $r \in \mathscr{R}$ follows an exponential distribution with parameter λ_r. The TTL of requests which are the instances of $r \in \mathscr{R}$ follows an exponential distribution with parameter μ_r.

Request $r \in R$ arrives and releases randomly. At each time slot, the system performs the following actions: release the terminated requests; accept or reject the arriving requests in sequence; and update the network state. There are two cases between each pair of adjacent time slots. First, if new requests arrive in a time slot, the allocation of any replica instance leads to the state transition in MDP; this work calls this case *intra time slot*. Second, if there are no new requests during several continuous time slots, no action is needed; this work calls this case *inter time slot*. Fig. 7.4 shows the procedure during the moving of time slots. Each time slot includes several steps. After all time slots, one

episode is completed.

This work devise a greedy *serialization-and-backtracking (SB)* method to handle more than one arriving request in an *intra time slot*. The set of arriving requests is pre-ordered by the system according to the priorities of the requests. The allocation tries to satisfy the processing requirements of each request in sequence. Take the time slot $t \in T$ as an example. The system releases the resources taken up by the replica instances of timeout requests at first, and then obtains the newly arrival requests at time slot t as a batch $R_t^{\mathbb{N}}$. The system tries to satisfy the requirements of the current request by deciding the actions and allocating the replica instances of the requested VNFs one by one. In each step, the system decides an allocation of a replica of a VNF requested by the currently being processed request. If a replica is assigned to a node which has insufficient capacity, a punishment on the reward (a negative value) is added. If the requirements of the processing abilities of a requested VNF can be satisfied after going through all the replicas of all VNFs, the request is accepted and the reward for accepting the request is gained; otherwise, the request is rejected and the network state backtracks to the beginning of the allocations. The requests are processed in sequence until the last request in R_t is accepted or rejected. As shown in Fig 7.4, the requirement of request 1 cannot be satisfied after going through all replicas of all VNFs. Request 1 is rejected. The allocations of replica instances for request 1 are abandoned and the network state backtracks to the beginning of the allocation so that the allocations of replica instances for request 2 are not influenced by the abandoned allocations. After all requests are processed, the system implements the allocations and updates the network state.

In *inter time slots*, there is no arriving requests as the time slots from $t-1$ to t in Fig. 7.4. The system releases the timeout requests, updates the system state, and calculates the rewards for the released requests according to their resiliency levels.

The procedure to handle the dynamic requests is listed in Algorithm 7.1.

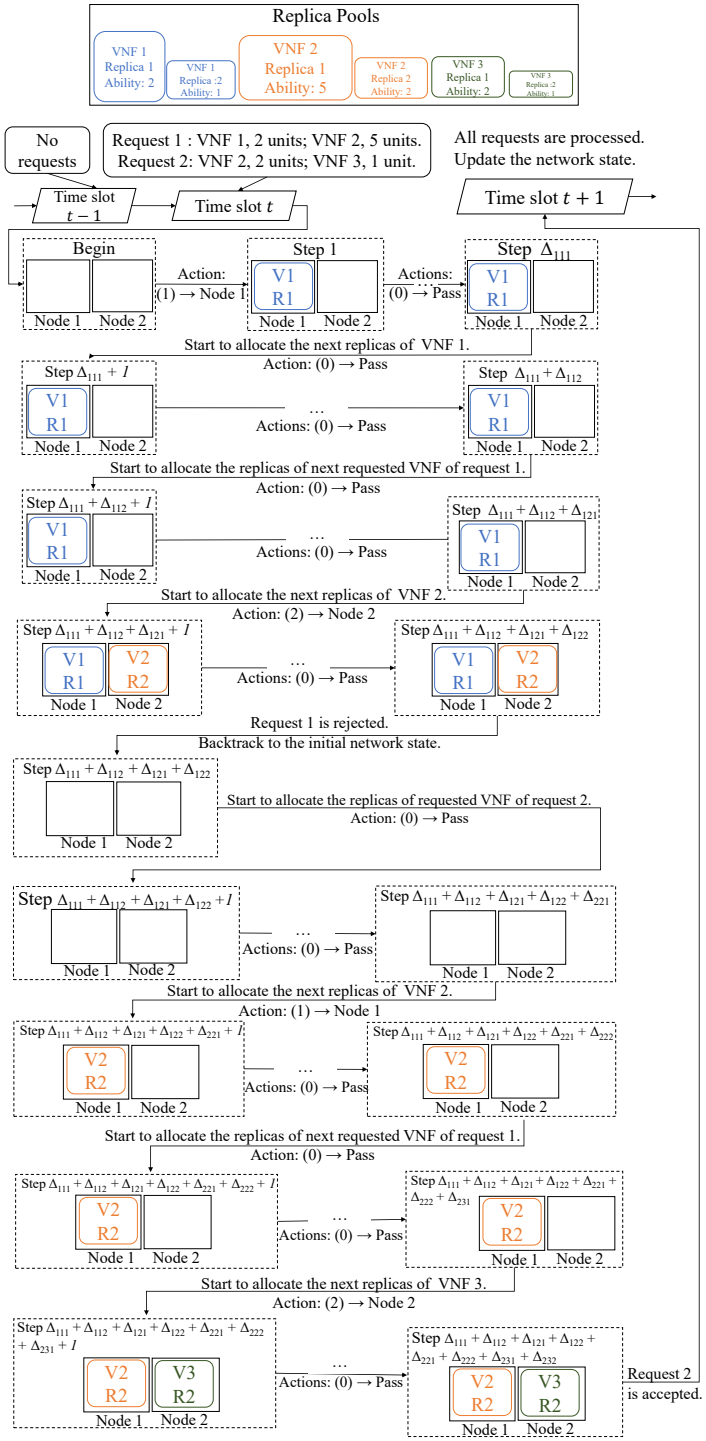


Figure 7.4: Procedure in RL-VNFA as the time slot moves on. Vx Ry represents VNF x replica y.

Algorithm 7.1 Procedure in RL-VNFA as the time slot moves

Initial time slot $t \leftarrow 1$.
while $R_t^{\mathbb{N}} = \emptyset$ **do**
    if Any request times out **then**
        Release the replica instances of the request.
    end if
    $t \leftarrow t + 1$.
end while
for Request $r \in R_t^{\mathbb{N}}$ **do**
    for VNF $f \in F_r$ **do**
        for Replica $a \in A_f$ **do**
            for Instance $i \in [1, \Delta_{rfa}]$ **do**
                Select an action α from $\mathcal{A}$
                if There is enough capacity for replica a on the selected node. **then**
                    Allocate the replica to the selected node.
                end if
            end for
        end for
    end for
    if the allocated replica instances satisfy the requirements of r and accepted requests **then**
        Request r is accepted. Reward + 1.
    else
        Request r is rejected. Backtrack the network state before the receiving of request r.
    end if
end for

7.2.3 PG-based training procedure

PG uses the potential reward of an action to increase or decrease the probabilities of the action occurrences. Our target is to obtain a policy to maximize the expected final reward after the state transitions. Let $\pi(a|s,\theta)$ denote the policy which represents the probability of choosing action $a \in \mathcal{A}$ at state $s \in \mathcal{S}$ under parameter θ. An episode in the training procedure is a sequence of MDP state transitions. With PG, the training procedure needs to find the suitable parameter θ by using gradient descent to achieve our target in an episode during the interactions between the agent and the environment. The objective function of the training procedure is given by:

$$\max \mathcal{J}(\theta) = \sum_{e \in [1,|E|]} \pi(a|s,\theta) r(s_e, a_e), \tag{7.11}$$

where E is the length of the episode, $r(s_e, a_e)$ is the reward of selecting action a at state s in the eth state of the episode. $\mathcal{J}(\theta)$ is the expected final reward of the episode.

PG is given by the gradient descent of the parameter, $\nabla_\theta J(\theta)$. θ is updated by:

$$\theta_{\text{next}} \leftarrow \theta + \varepsilon \nabla_\theta J(\theta), \tag{7.12}$$

where ε is the learning rate and is used for adjusting the convergence speed of the training procedure. A higher learning rate increases the convergence speed, but may lead to missing local minimum values.

The PG-based training procedure is shown in Algorithm 7.2. The algorithm initializes parameter θ randomly at first. In each episode, actions are selected by the current policy. The reward in each step is calculated and recorded for updating the policy at last.

7.3 Evaluation

7.3.1 Baseline models

This work gives two baseline models to compare with the proposed model introduced in Section 7.1.3. The first baseline model is called a random decision

Algorithm 7.2 PG-based training procedure

Initialize θ randomly.
for episode $\leftarrow$ [1, maximum episodes] **do**
    Initialize the network state. Let the current state be the first state, i.e., $s \leftarrow s_1$.
    for step $\leftarrow$ [1, maximum steps] **do**
        Select action α_{step} according to the policy $\pi(a_{\text{step}}|s_{\text{step}}, \theta)$.
        Calculate the reward r_{step} based on the selected action α.
        Transfer to the next state $s_{\text{step+1}}$.
        Reward for updating the policy is calculated by: $\sum_{i\in[1,\text{step}]} \gamma^{\text{step}-i} r_i$
    end for
    Updating θ by (7.12).
end for

model (RDM), which decides the allocation of replica instances belonging to each requested VNF in each request randomly. The second baseline model is called a single-slot allocation model (SAM), which tries to accept as many requests as possible at each single time slot. For time slot $t \in \boldsymbol{T}$, SAM can be expressed by:

$$\max \sum_{r\in R^{\mathbb{N}}} \rho_r \tag{7.13a}$$

$$\text{s.t. } (7.1) - (7.5), (7.7) - (7.9), \tag{7.13b}$$

$$x_{tn}^{fa} \in \{0,1\}, \forall n \in N, f \in F, a \in A^f. \tag{7.13c}$$

SAM is solved by Algorithm 7.3. The time complexity of Algorithm 7.3 is $O\Big(|R||F|\big(\sum_{a\in A_f} \Delta_{rfs}(\log \sum_{a\in A_f} \Delta_{rfs} + |N| \log |N|)\big)\Big)$.

7.3.2 Performance evaluations

The proposed and baseline models are solved by Python 3.8.8 running on the AMD Ryzen 3600 3.6GHz 6-core CPU, NVIDIA GeForce GTX1660 Super GPU, 16 GB memory.

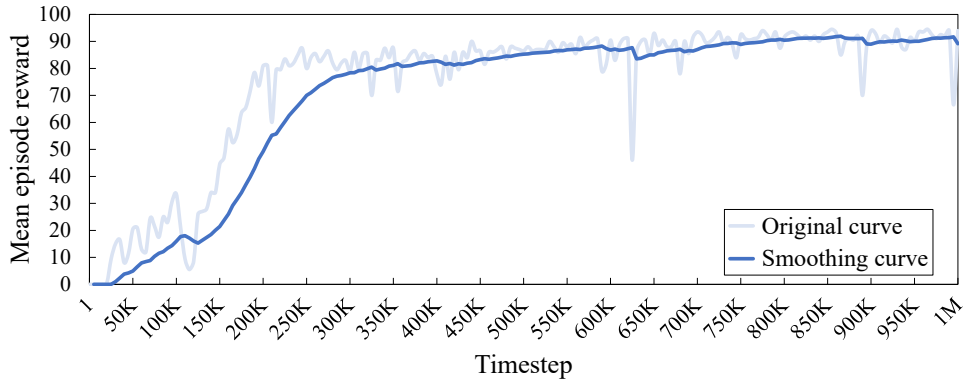
This work prepares two cases, cases 1 and 2, for the evaluation. This work evaluates the performance of the proposed model in the cases that the resources of nodes are limited for a large number of requests so that not all requests can be accepted. In case 1, the system receives 500 requests among 100 time slots and tries to allocate them to a three-node cluster. Each request selects at least

Algorithm 7.3 Obtaining the allocation of replicas at time slot $t \in T$ in SAM

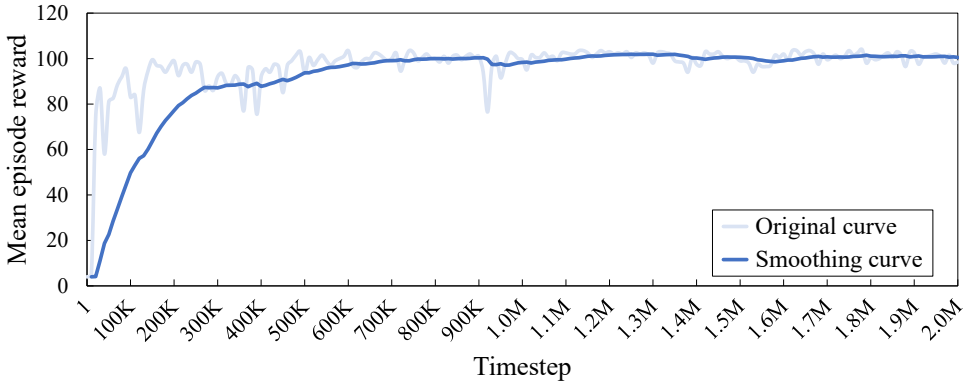
for $r \in R_t$ **do**
  for $f \in F$ **do**
    if f is satisfied by the allocated replicas **then**
      Continue to the next function.
    end if
    Sort A_f according to processing ability of each replica increasingly for the replicas whose processing ability is larger than the required processing ability. For the other replicas, sort them according to processing ability of each replica decreasingly and put them after the previous set.
    for $a \in A_f$ **do**
      Sort N according to the providable processing ability of each non-decreasingly. If there are any nodes that have the same processing ability, sort them according to the highest latency among all nodes in N_f increasingly.
      for $n \in N$ **do**
        if there is enough capacity on n for a **then**
          Allocate replica a to node n
          Break
        end if
      end for
      if f is satisfied by the allocated replicas **then**
        Continue to the next function.
      end if
    end for
  end for
end for

two VNFs and at most three VNFs from five VNFs, the required processing abilities are generated randomly between one and ten units, and the resiliency levels are generated randomly between one and two. In case 2, the system receives 1000 requests among 100 time slots and tries to allocate them to a six-node cluster. The TTL of each request in both cases is generated between 10 and 100 time slots. Each request selects at least two VNFs and at most five VNFs from five VNFs, the required processing abilities are generated randomly between one and twenty units, and the resiliency levels are generated randomly between one and three. The requests in both cases belong to five types of requests. The pools of VNFs are given arbitrarily, which include four replicas. The processing abilities of these replicas are 1, 2, 3, and 5, respectively. The required capacities of these replicas are 2, 3, 4, and 6, respectively. The capacity of each node in the clusters has ten units capacity. The interval of the arrival time between each pair of adjacent requests obeys an exponential distribution. The TTL of each request obeys exponential distribution. The priority of each request is set to one.

In each case, this work trains a model under randomly generated environments by using the RL-VNFA approach and evaluate the performance of the trained model in two tests, which are tests 1 and 2. In test 1, this work trains one model for each case with a set of randomly generated requests. This work uses one million and two million episodes for training the models in cases 1 and 2, respectively. The training time are 23.43 [min] and 54.95 [min], respectively. The mean episode reward increases with the increase of training times, as shown in Fig. 7.5. This work compares the proposed model solved by the RL-VNFA with the two baseline models introduced in Section 7.3.1 under five randomly generated requests. For each request, the RL-VNFA and RDM use the average objective value from five decisions. This work explores the impact of two levels of randomization on the performances of the models. In the first level, the arrival time and TTL of evaluated requests are randomly generated. This work calls it partial randomization (PR). In the second level, the required VNFs, processing abilities, arrival time, and TTL of evaluated requests are randomly generated. This work calls it complete randomization (CR). The average objective values and computation times of these approaches are shown in Tables 7.1 and 7.2, respectively. This work can observe that the proposed



(a) Case 1.

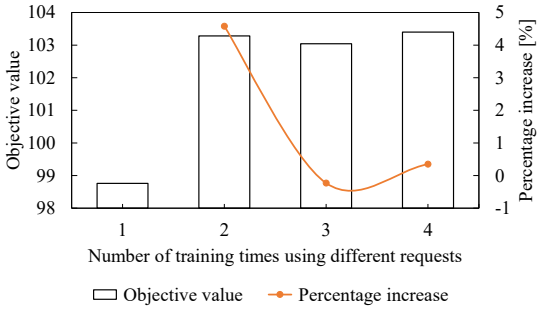


(b) Case 2.

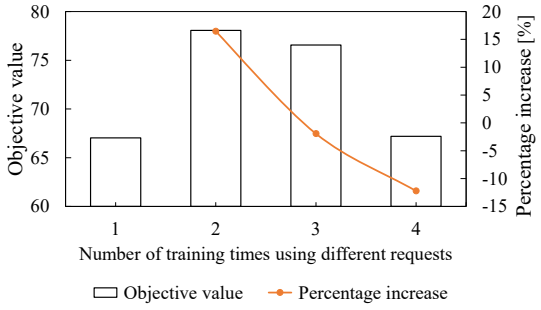
Figure 7.5: Increasing of mean episode rewards during training procedures.

model improves the number of accepted requests at the cost of longer computation time. The objective value obtained by the proposed model is 20.85 times larger than that of RDM and 1.58 times larger than that of SAM on average. Although the computation time of the proposed model is 11.59 times longer than that of RDM and 5.06 times longer than that of SAM, it is still in tens of milliseconds, which can be used in dynamic systems. Compared with the performances in PR and CR, the proposed model performs better in a limited range of random variables.

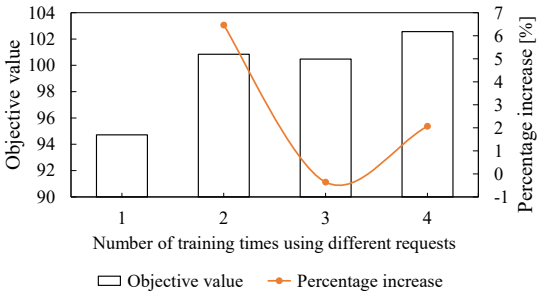
In test 2, this work evaluates the effectiveness of the additional training data on improving the performances of the trained models confronting with various environments. This work trains the models in cases 1 and 2 with one, two, and three additional randomly generated requests and evaluate them with



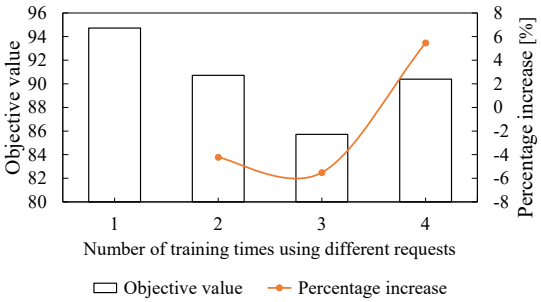
(a) Case 1 with partially randomized requests.



(b) Case 1 with completely randomized requests.



(c) Case 2 with partially randomized requests.



(d) Case 2 with completely randomized requests.

Figure 7.6: Increasing of mean episode rewards during training procedures.

Table 7.1: Objective values obtained by proposed and baseline models in test 1.

Models	Case 1		Case 2	
	PR	CR	PR	CR
RDM	1.12	0.16	8.68	7.08
SAM	61.00	42.00	89.4	33.00
Proposed	94.72	94.72	98.76	67.04

Table 7.2: Computation times [s] of proposed and baseline models per request in test 1.

Models	Case 1		Case 2	
	PR	CR	PR	CR
RDM	0.00418	0.00419	0.00595	0.00802
SAM	0.00932	0.00804	0.01447	0.01931
Proposed	0.05630	0.05405	0.06311	0.08537

five randomly generated requests. In each training procedure, this work uses 0.5 million and one million episodes for cases 1 and 2, respectively. The average training time for cases 1 and 2 are 10.93 [min] and 23.10 [min], respectively. This work evaluates the performances of the trained models in PR and CR, which are the same with those in test 1. The objective values obtained by different trained models are shown in Fig. 7.6. This work can observe that one more training data may increase the performance of the proposed model in the examined cases. However, the rate of increase of the objective values obtained by RL-VNFA decreases with the increase of the amount of additional training data. The performances even may become worse when too much training data are used. There are two reasons for this situation. Firstly, the differences between the training data set and the validation data set influence the quality of the solution. If the trained model is only trained by some similar data sets, its robustness is weak, i.e., only some specific cases can be solved. Secondly, as the number of training sessions increases, the trained model becomes over-fitting for a type of specific case if the training data sets are similar, or, the trained model becomes confused if the training data sets are different. Too

much data for training is potential for the decrease of the model performance. Suitable training data sets can let the trained model experience most of the possible requests and do not focus on any specific case.

7.4 Summary

This chapter proposed a resilient VNF allocation model for increasing the number of accepted requests with ensuring fault tolerance and considering VNF diversity in a dynamic scenario. This work developed an RL-based approach for solving the proposed model. This work designed the procedures for dealing with arrival requests in the approach. Numerical results showed that the proposed model increases the number of accepted requests compared with the two baseline models. Compared with RDM which randomly determines the allocations of replica instances, the proposed model accepts 20.85 times more requests on average in the examined cases. Compared with SAM which determines the allocations of replica instances independently at each time slot, the proposed model accepts 1.58 times more requests on average in the examined cases. Although the proposed model improves the performance at the cost of a longer computation time compared with the baseline models, it can still determine the allocation of a request in tens of milliseconds in the examined cases. This work evaluated the impact of additional training data on the performance of the proposed model. Suitable additional data can improve the performance of the proposed model. Excessive training is harmful, which leads to the performance degradation of the proposed model.

Chapter 8

Implementation of resource allocation models and service function chains

This chapter provides four demonstrations of how to implement resource allocation models and the relative algorithms, e.g., the proposed models and algorithms in Chapters 3–7, in the real network system. The current controller in Kubernetes cannot manage the custom-defined resources in the proposed models, e.g., service function chains in Chapters 3–5, backup resources in Chapter 4, and VNF diversity in Chapter 6. A current scheduler in Kubernetes cannot deploy the VNFs to nodes automatically cooperating with multiple VNF allocation models. This chapter aims to provide controller-based and scheduler-based mechanisms cooperating with VNF resource allocation models for a single resource allocation model with special customized resources and multiple resource allocation models supporting hot swapping. This chapter also aims to provide a network plugin that can manage the connections between VNFs in SFCs cooperating with the controller and scheduler implemented with the models related to the SFCs, e.g., models in Chapters 3–5. The demonstrations can implement the approaches to solve the proposed models including a optimization solver for precision solutions, e.g., CPLEX [82] and Gurobi [112], and heuristic algorithms, e.g., algorithms introduced in Sections 3.3, 4.3, 5.3, and 6.2.

Among the four demonstrations introduced in this chapter, two of them are designed for general VNFs [113, 114] and the other two implementations are designed for VNFs in SFCs with the realizations of SFCs [115, 116].

8.1 Controller-based implementation of VNF allocation model considering diversity and redundancy in Kubernetes

The controller-based implementation is designed for the users who have special requirements on resources. For example, this section considers the VNF diversity with resource pools and VNF redundancy with backup resources. In this implementation, VNFs are usually packaged in VMs or containers. Kubernetes [9] is an open-source system for automating deployment, scaling, and management of containerized applications. A Pod is the smallest deployable unit of computing that users can create and manage in Kubernetes. The deployment with a given number of the replicas of Pods is a realization of a network function in Kubernetes. The controller of a resource in Kubernetes adjusts the current state to the expected state through the control loop [117]. For example, the deployment controller maintains the desired number of Pod replicas. The providable processing ability of a network function can be required instead of the number of active replicas of the function.

VNF diversity uses a group of replicas with different processing abilities and resource requirements to replace a single VNF instance, which can fully utilize server computing resources, especially for edge computing devices. The replicas of a VNF are chosen from a pool of replica templates, which is given by the cloud provider. The service providers allocate replicas with the requirements of resources to provide certain processing abilities. VNF diversity may increase the risk of service unavailability since it allocates replicas to different servers. The unavailability of a server leads to service performance degradation. VNF redundancy which provides backups for replicas is adopted to increase the service reliability. In [10], Chapters 6 and 7 provided allocation models considering VNF diversity and redundancy jointly, which improves the service resiliency.

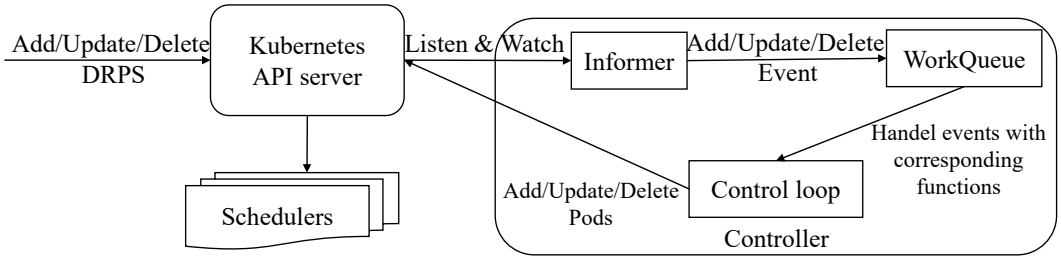


Figure 8.1: Overall structure.

However, the current deployment in Kubernetes is based on the allocation of a given number of replicas with fixed specifications [118]. The fixed specifications may not fully utilize the computing resources of servers. If the deployment can be realized by the replicas with different resource requirements and processing abilities, the deployment can be allocated flexibly to the servers with different capacities, and the computing resources of servers can be fully utilized. The automatic selection, creation, and management of diversity and redundancy resources are not considered in Kubernetes.

This section designs and implements a custom resource controller in Kubernetes based on the processing ability. The custom defined resource (CDR) jointly considers the diversity and redundancy of VNFs. This work calls it *diversity and redundancy Pod set (DRPS)*. This work uses exact and approximate methods to select suitable replicas from a pool of replica templates to satisfy the required processing ability with the minimum required number of replicas. At last, this work performs demonstrations of the controller including the function allocation and the switching from primary replicas to backups, to show the improvement of server utilization by adopting DRPS.

8.1.1 Design and Implementation

Overall structure

Fig. 8.1 overviews the structure of the reported controller and relative components. The service providers submit the configurations including adding, updating, and deleting, in a form defined in DRPS definition to the Kubernetes application programming interface (API) server. Informer listens to the

changes of the DRPS instances and pushes the corresponding events to the WorkQueue. The unterminated control loop handles the events in WorkQueue with the corresponding functions to let the current state of the DRPS instances keep pace with the desired state of the DRPS instances configured by the users. The key point is how to use the controller to cooperate with the “add", “delete", “update", and “get" of the DRPS instances in Kubernetes. The scheduler decides the locations of the created Pods with the default scheduler or allocation-model-based scheduler in Chapter 6.

Definition of DRPS instance

The definition of a DRPS instance includes three parts: *metadata, specification, and status*. *Metadata* contains the information that distinguishes different DRPS instances, e.g., name and creation timestamp. *Specification* contains the properties of a DRPS instance, e.g., required processing ability, specified solution methods, and the pool of replica templates. The pool of replica templates contains the templates of Pods, which includes the processing ability of the replica, the resource requirements, and the container image. *Status* contains the latest status of the CDR instance, e.g., the number and the names of Pods hosted by the instance. *Status* is updated periodically or updated after modification in the control loop.

Control loop

The control loop receives and handles to create, update, and delete events. When a DRPS instance is created, the Pods are created and allocated to nodes. The selection of replica templates from pools and the scheduling of the created Pods can be determined by two methods: an exact method, e.g, integer linear programming (ILP), and an approximate method, e.g., heuristic algorithms. When some primary Pods hosted by the instance fail, backup Pods are converted to primary Pods. The remaining primary Pods are deleted and a group of new backup Pods is created and allocated. Alternatively, the remaining primary Pods are converted to backup Pods and new backup Pods are created to compensate for the loss of processing ability. Fig. 8.2 shows the flow chart of the controller.

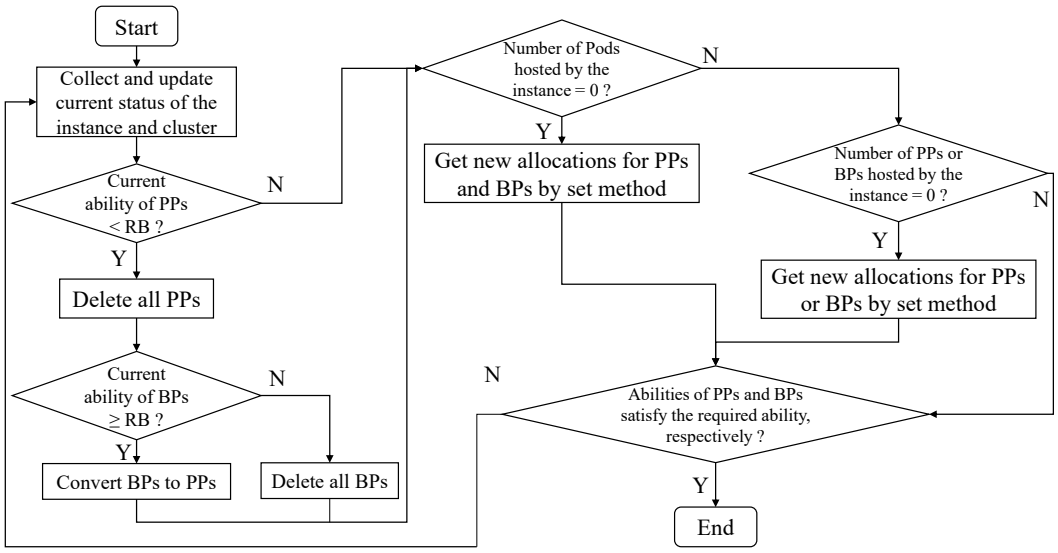


Figure 8.2: Flow chart of controller. PP: primary Pod. BP: backup Pod. RB: required ability.

8.1.2 Demonstrations

The demonstrations implement the controller by Operator SDK v1.4.2, Golang 1.15, and Python 3.7 in Kubernetes 1.20 on a five-node Kubernetes cluster (one master node and four worker nodes). The memory and central processing unit (CPU) of four nodes are up to 2.3 GB and two cores, 2.3 GB and two cores, 4.2 GB and three cores, and 4.2 GB and four cores, respectively. The demonstrations deploy the controller as a deployment on the master node. The demonstrations create a DRPS instance by applying a configuration file shown in Fig. 8.3 and the list of Pods is shown in Fig. 8.4. A primary Pod hosted by the instance fails, the backup Pods is switched to primary Pods and new backup Pods are generated, as shown in Fig. 8.5. We can observe that the backup Pod is converted to the primary Pod and a new backup Pod is started.

This demonstration deploys the instances with the same required ability and the pool in Fig. 8.3 by using two types: traditional deployment and DRPS. The traditional deployment accepts one request of the instance and completes the allocations of a primary Pod and a backup Pod in 0.004 [s]. The DRPS instance accepts two requests and completes the allocations in 2.931 [s]. The allocation of the DRPS instance is shown in Fig. 8.6. The CPU and memory

```
apiVersion: drps.drps.example.com/v1alpha1
kind: DiversityRedundancyPodSet
metadata:
  name: drps-demo
spec:
  requiredProcessingAbility: 10
  useBackups: True
  initStrategy: ILPapproach
  updateStrategy: SAapproach
  templates:
    - processingAbility: 1
      requiredResources:
        memory: "300Mi"
        cpu: 300m
      spec:
        containers:
        - name: test1
          image: lorel/docker-stress-ng
          args: ["--vm", "1", "--vm-bytes", "300M", "--cpu", "4"]
          resources:
            limits:
              cpu: 300m
              memory: 300m
    - processingAbility: 2
      requiredResources:
        memory: "500Mi"
        cpu: 600m
      spec:
        containers:
        - name: test2
          image: lorel/docker-stress-ng
          args: ["--vm", "1", "--vm-bytes", "500M", "--cpu", "4"]
          resources:
            limits:
              cpu: 600m
              memory: "500Mi"
    - processingAbility: 5
      requiredResources:
        memory: "1100Mi"
        cpu: 1300m
      spec:
        containers:
        - name: test3
          image: lorel/docker-stress-ng
          args: ["--vm", "2", "--vm-bytes", "525M", "--cpu", "4"]
          resources:
            limits:
              cpu: 1300m
              memory: "1100Mi"
    - processingAbility: 10
      requiredResources:
        memory: "2100Mi"
        cpu: 2600m
      spec:
        containers:
        - name: test4
          image: lorel/docker-stress-ng
          args: ["--vm", "4", "--vm-bytes", "525M", "--cpu", "4"]
          resources:
            limits:
              cpu: 2600m
              memory: "2100Mi"
```

Figure 8.3: Configuration file of DRPS instance.

```
NAME                                READY   STATUS    RESTARTS   AGE   LABELS
drps-demo-template3-03768b701005    1/1     Running   0          81s   ability=10,app=drps-demo,type=primary
drps-demo-template3-4a03b9e9076e    1/1     Running   0          81s   ability=10,app=drps-demo,type=backup
```

Figure 8.4: Pod list after allocation.

```
NAME                                READY   STATUS    RESTARTS   AGE     LABELS
drps-demo-template3-3c9d13a87b6e    1/1     Running   0          2s      ability=10,app=drps-demo,type=backup
drps-demo-template3-4a03b9e9076e    1/1     Running   0          2m23s   ability=10,app=drps-demo,type=primary
```

Figure 8.5: Pod list after primary Pod fails.

utilizations are compared in Fig. 8.7. We can observe that DRPS can accept more requests and improve the system resource utilization compared with the default deployment method with the cost of longer deployment time. The main factor contributing to the increase in deployment time is solving complex deployment models. More accurate results often lead to longer solution times. Users of this demonstration need to carefully make a balance between the deployment time and resource utilization and choose the suitable algorithm

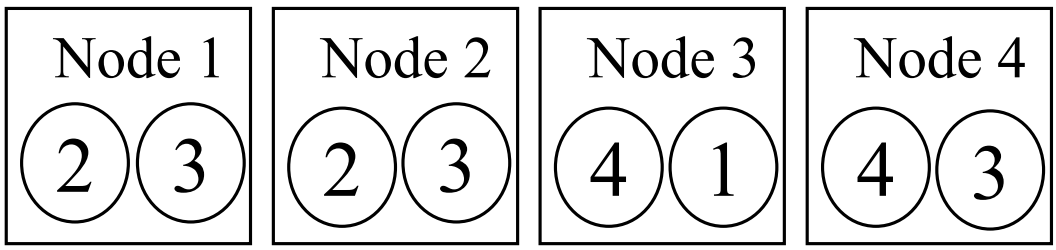


Figure 8.6: Allocation results of DRPS instance. A circular is a Pod. The number in a circle means the template which the Pod uses.

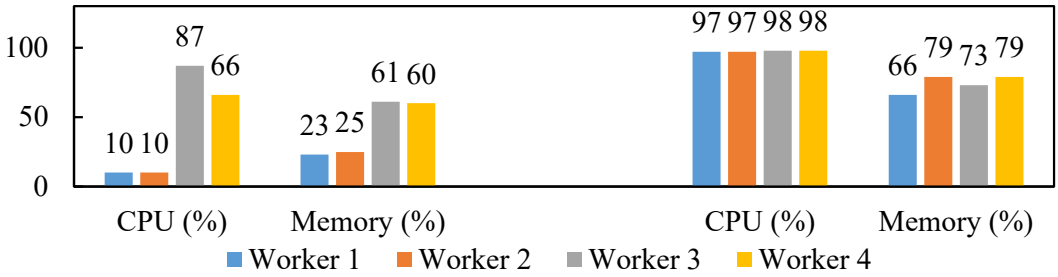


Figure 8.7: CPU and memory utilization comparison between default deployment method (left) and DRPS (right).

for solving their deployment problem.

8.2 Design of scheduler plugins for VNF allocation models in Kubernetes

The scheduler-based implementation is designed for the users who have no special requirements on resources. Compared with the implementation in Section 8.1, this section implementation the VNF allocation models in a more general way. This implementation relies on the scheduling framework in Kubernetes. The locations of Pods are decided by the schedulers in Kubernetes, which are implemented by a set of “plugin" application programming interfaces (APIs) belonging to a pluggable architecture called scheduling framework [100].

The works in Chapters 3-7 developed function allocation models, which are designed for different aspects of allocations with different objectives. The

models are solved by different optimization solvers, such as IBM® CPLEX®, and different heuristic algorithms, such as the simulated annealing algorithm and the genetic algorithm.

The scheduler must allocate the VNFs to suitable locations according to the results obtained by mathematical calculation. In addition, different users have different requests of services with different quality requirements. The expandability of scheduler is necessary to support multiple models without restarting the scheduler in case of service interruption. However, no tools in Kubernetes can be used to deploy the VNFs to nodes by using the results from multiple allocation models.

This implementation designs and implements a scheduler to allocate VNFs according to calculation results from multiple reliable function allocation models based on the scheduler framework in Kubernetes. The scheduler provides an intermediate layer to convert the results from different calculators to the pairs of VNFs and nodes in the network. New allocation models can be added to the scheduler without restarting the scheduler. At last, this implementation prepares two demonstrations of the scheduler, one for multiple allocation models, and the other one for a specific use case in a sensor network.

8.2.1 Design and Implementation

Overall structure

Fig. 8.8 overviews the structure of the reported scheduler. There are mainly two components in the structure, a scheduler Pod and several model Pods. The scheduler Pod receives the queue of the Pods corresponding to VNF requests waiting to be allocated and determines the allocation of them computed by the requested models according to the current network status. The model Pods receive the network status, e.g., the available resources of nodes and links, and compute the allocations. In addition, the model Pods convert the network status from the scheduler Pod to the recognized data of calculators and convert the calculation results from the calculators to the recognized data of the scheduler Pod.

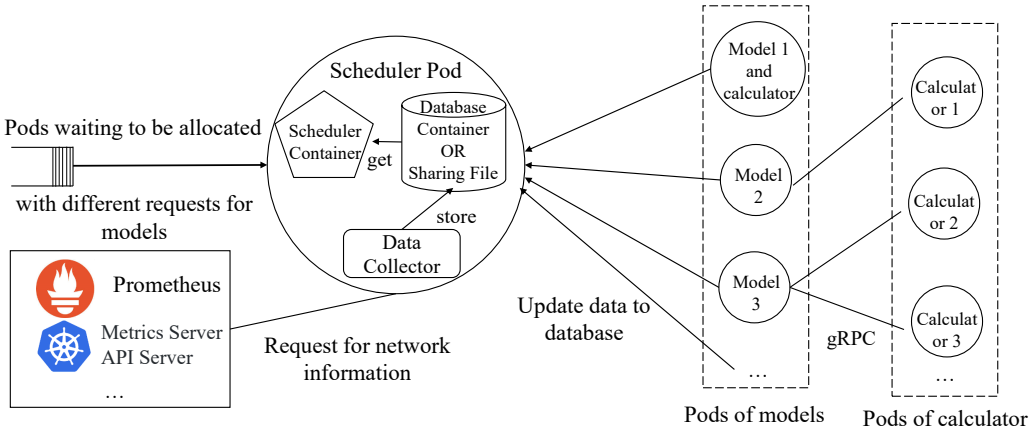


Figure 8.8: Structure of designed scheduler.

Pods design

Scheduler Pod: there are mainly three components in the scheduler Pod: a scheduler container, a data collector, and a database container or a shared volume between the above two containers, as shown in Fig. 8.8. In addition, another deployment can be used in some cases. The database container can be separated into an independent Pod considering the sharing with other applications or high stress on the network resource consumption. The other two functions communicate with the database through the Pod network.

The scheduler container implements four plugins in Kubernetes scheduling framework [100], as shown in Fig. 8.9. The Sort plugin reorders the Pods corresponding to VNF requests in the waiting queue. The scheduler lets the Pods corresponding to VNF requests and requesting the same model be scheduled together. The Prescore plugin in our scheduler marks the non-schedulable nodes and compares the existing data and current data. If the schedulable nodes are changed or the available resources vary beyond a set threshold, the scheduler Pod makes a model Pod start to compute the VNF allocations according to the updated data. In addition, if no allocation result is detected within the set timeout value, the plugin reports an error. The Score plugin gives the scores for each node in an allocation. The Pods corresponding to VNF requests are allocated to the node which has the highest score. The implementation uses the allocation results from the corresponding models to allocate the

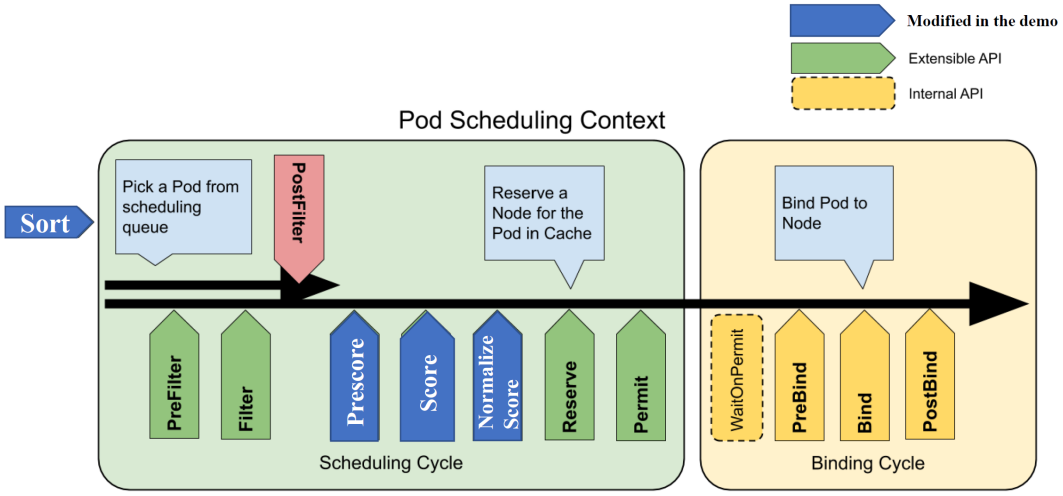


Figure 8.9: Plugins modified in scheduling framework.

Pods corresponding to VNF requests. For each Pod, the implementation gives the highest score to the node to which it should be allocated so that the Pod can be allocated following the allocation results. The Normalize Score plugin limits the final scores of nodes to a range, $[0, 100]$. This implementation also registers the plugins to Kubernetes and give the permissions to let the plugins get the required data and decide the score of each node.

Model Pods: the model Pod has two main functions: data conversion and uploading allocation results to the scheduler Pod. The model Pods may receive a list of node names, a list of VNF names, and other network parameters. They convert these data to the recognized format of the calculators. When they receive the results from the calculators, they summarize the calculation results and upload the results to the scheduler Pod in a prescribed format.

The model Pod has two types: calculator(s) in the model Pod and calculator(s) out of the model Pod. If the model Pod includes the calculators, it receives the data from the scheduler Pod, then computes the results, and finally sends them to the scheduler Pod. If the calculator Pods are independent of the model Pod, the model Pod starts its required calculator Pods and manages their lifecycles.

A model Pod and independent calculator Pods are created as Jobs. A Job in Kubernetes creates one or more Pods and ensures that a specified number

of the Pods successfully terminate [119]. The model Pod Jobs are started as Cron Jobs by the administrators, which run and upload the results on a regular interval. If the Prescore plugin requires to update the allocation results according to new network data, the model Pods are created as normal Jobs by the plugin.

Database design

The database used in this scheduler has three types of tables, table *node*, table *request*, and table *request_model*. There is only one table for *node* in the database and each request has its own table *request*. Table *node* has the fields of node name and unique identification number (UID), which are unique. The node name is the primary key in this table. Table *node* also has the node resource information, such as maximum CPU, maximum memory, available CPU, and available memory. Each request has a field that marks the availability of a node in the request. At last, each record has a timestamp which is the creation time of this record. Table *request* records the allocation results of the VNFs in a request. Each VNF has a field in the table, which is linked with the field *name* in table *node* as the foreign key. Each record has an auto-increment primary key, ID, a creation timestamp, and a name of the request model. There is only one table for *request_model* in the database, which stores the allocation result update intervals, the image names of model Pods, initialize commands, and correspondences between the models and the requests. One correspondence ensures that one request is computed by one model.

Communication between components

There are three types of communications between components mentioned in the structure: communication with Kubernetes API, communication with the database, and communication between model Pods and calculator Pods. The data collector and the scheduler container in scheduler Pod and some model Pods need to communicate with the API server or the metrics server by Kubernetes API. The communications are implemented by REST and client libraries. The model Pods, scheduler application, and network data collector

```
NAMESPACE     NAME                                READY   STATUS    RESTARTS   AGE   IP               NODE
kube-system   coredns-f9fd979d6-llc2r             1/1     Running   0          18m   10.244.0.3       master
kube-system   coredns-f9fd979d6-mc5cw             1/1     Running   0          18m   10.244.0.2       master
kube-system   etcd-master                         1/1     Running   0          18m   192.168.231.129  master
kube-system   kube-apiserver-master               1/1     Running   0          18m   192.168.231.129  master
kube-system   kube-controller-manager-master      1/1     Running   0          18m   192.168.231.129  master
kube-system   kube-flannel-ds-amd64-5gdfn         1/1     Running   0          16m   192.168.231.129  master
kube-system   kube-flannel-ds-amd64-h42m6         1/1     Running   0          16m   192.168.231.131  node1
kube-system   kube-flannel-ds-amd64-nfbzm         1/1     Running   0          16m   192.168.231.130  node3
kube-system   kube-flannel-ds-amd64-w2ljz         1/1     Running   0          16m   192.168.231.132  node2
kube-system   kube-proxy-hxlk4                    1/1     Running   0          18m   192.168.231.129  master
kube-system   kube-proxy-n8n26                    1/1     Running   0          17m   192.168.231.130  node3
kube-system   kube-proxy-vgnxz                    1/1     Running   0          17m   192.168.231.131  node1
kube-system   kube-proxy-wljms                    1/1     Running   0          17m   192.168.231.132  node2
kube-system   kube-scheduler-master               1/1     Running   0          18m   192.168.231.129  master
kube-system   metrics-server-79d68485bf-wtgj7     1/1     Running   0          16m   10.244.3.2       node3
kube-system   mysql-686d65cc97-8g6ck              1/1     Running   0          15m   10.244.2.2       node2
kube-system   scam-scheduler-7878d59856-kdcdc     2/2     Running   0          69s   10.244.1.3       node1
```

Figure 8.10: Demo 1: list of existing Pods in the beginning.

in the scheduler Pod need to communicate with the database. The Internet protocol (IP) address and port of the database are given in advance. They use transmission control protocol (TCP) connections to communicate with the database. The communication between model Pods and calculator Pods is implemented by gRPC. The model Pod remotely calls functions running in the calculator Pods and receives the results from them.

8.2.2 Demonstrations

The demonstrations implement the scheduler application by Golang 1.15 and the data collector by Python 3.9. The images of the containers are built by Docker 20.10.0. The database in the demonstrations is MySQL 8.0.22. The demonstrations run on a four-node Kubernetes cluster, one master node, and three worker nodes. The version of Kubernetes is 1.19.0.

Demonstration with multiple computing models

The demonstration prepares two requests which request models in [120] and Chapter 3. Request 1 has five VNFs. Request 2 has three VNFs. The list of Pods is shown in Fig. 8.10 and the Pods created for the scheduler are surrounded by a box. The table *node* in the database is shown in Fig. 8.11.

Then the demonstration creates the Pods in two requests by a YAML configuration file. The model Pods are created as Jobs and run as shown in

```
mysql> select * from nodes;
+--------------------------------------+--------+-----------+-----------+------------------+---------------+---------------------+----------+----------+
| uid                                  | name   | memory    | cpu       | available_memory | available_cpu | timestamp           | request1 | request2 |
+--------------------------------------+--------+-----------+-----------+------------------+---------------+---------------------+----------+----------+
| a234a7c8-fbaa-4cc2-9f8a-a9aae59dd992 | master | 8215040000 | 4000000000 |       5260510000 |    3725310000 | 2020-12-31 16:11:07 |        1 |        1 |
| a3923940-4638-4080-a270-e86f0a64b5ef | node1  | 3993400000 | 1000000000 |       1789460000 |     979371000 | 2020-12-31 16:11:00 |        1 |        1 |
| 271a4485-6b1a-42aa-9622-12b2b40a2d90 | node2  | 3993400000 | 1000000000 |       1934440000 |     977618000 | 2020-12-31 16:11:04 |        1 |        1 |
| 00e1256e-7d91-4f8b-9a20-0f86380ab5b5 | node3  | 3993400000 | 1000000000 |       2760770000 |     977245000 | 2020-12-31 16:11:06 |        1 |        1 |
+--------------------------------------+--------+-----------+-----------+------------------+---------------+---------------------+----------+----------+
4 rows in set (0.00 sec)
```

Figure 8.11: Demo 1: table *node* in database.

```
NAME      SCHEDULE      SUSPEND   ACTIVE   LAST SCHEDULE   AGE
model1    */10 * * * *   False     0        54s             47m
model2    */15 * * * *   False     0        54s             47m
```

(a) Model Jobs are created and running periodically.

```
         ID: 4
     nginx1: node2
   busybox1: node2
    debian1: node2
    radius1: node1
     mysql1: node2
  timestamp: 2020-12-31 17:20:15
 nginx1_bp1: node1
 nginx1_bp2: node3
busybox1_bp1: node1
busybox1_bp2: node3
 debian1_bq1: node1
 debian1_bp2: node3
 radius1_bp1: node2
 radius1_bp2: node3
  mysql1_bp1: node1
  mysql1_bp2: node3

         ID: 4
     nginx2: node1
   busybox2: node1
    debian2: node2
  timestamp: 2020-12-31 17:30:04
```

(b) Allocation results shown in tables *request1* and *request2*.

```
NAME            READY   STATUS    RESTARTS   AGE   IP             NODE
busybox1        1/1     Running   0          54s   10.244.3.14    node2
busybox1-bp1    1/1     Running   0          54s   10.244.1.17    node1
busybox1-bp2    1/1     Running   0          54s   10.244.2.10    node3
busybox2        1/1     Running   0          54s   10.244.1.19    node1
debian1         1/1     Running   0          54s   10.244.3.16    node2
debian1-bp1     1/1     Running   0          54s   10.244.1.16    node1
debian1-bp2     1/1     Running   0          54s   10.244.2.9     node3
debian2         1/1     Running   0          54s   10.244.3.19    node2
mysql1          1/1     Running   0          54s   10.244.3.18    node2
mysql1-bp1      1/1     Running   0          54s   10.244.1.18    node1
mysql1-bp2      1/1     Running   0          54s   10.244.2.12    node3
nginx1          1/1     Running   0          54s   10.244.3.15    node2
nginx1-bp1      1/1     Running   0          54s   10.244.1.20    node1
nginx1-bp2      1/1     Running   0          54s   10.244.2.8     node3
nginx2          1/1     Running   0          54s   10.244.1.15    node1
radius1         1/1     Running   0          54s   10.244.1.21    node1
radius1-bp1     1/1     Running   0          54s   10.244.3.17    node2
radius1-bp2     1/1     Running   0          54s   10.244.2.11    node3
```

(c) List of allocated Pods in the end.

```
I0101 14:54:50.254190       1 eventhandlers.go:173] add event for unscheduled pod default/debian2
I0101 14:54:50.261220       1 util.go:28] The request of Pod debian2 is request2.
I0101 14:54:51.261719       1 scheduler.go:452] Attempting to schedule pod: default/debian2
I0101 14:54:51.274053       1 plugin.go:244] Pod debian2 is allocated to Node node2 in Score.
I0101 14:54:51.274121       1 plugin.go:250] The score for Node node1 for Pod debian2 is 0.
I0101 14:54:51.281731       1 plugin.go:244] Pod debian2 is allocated to Node node2 in Score.
I0101 14:54:51.281755       1 plugin.go:250] The score for Node node3 for Pod debian2 is 0.
I0101 14:54:51.295169       1 plugin.go:244] Pod debian2 is allocated to Node node2 in Score.
I0101 14:54:51.295240       1 plugin.go:247] The score for Node node2 for Pod debian2 is 100.
I0101 14:54:51.300577       1 default_binder.go:51] Attempting to bind default/debian2 to node2
I0101 14:54:51.309373       1 scheduler.go:597] "Successfully bound pod to node" pod="default/debian2" node="node2" evaluatedNodes=4 feasibleNodes=3
```

(d) Log of a Pod, debian2, which is scheduled by our scheduler.

Figure 8.12: Demo 1: test results.

Fig. 8.12(a). After a few minutes, the allocation results are written into the database, as shown in Fig. 8.12(b). The list of Pods allocated by the scheduler is shown in Fig. 8.12(c). We observe that the final allocations are the same as the results from the models.

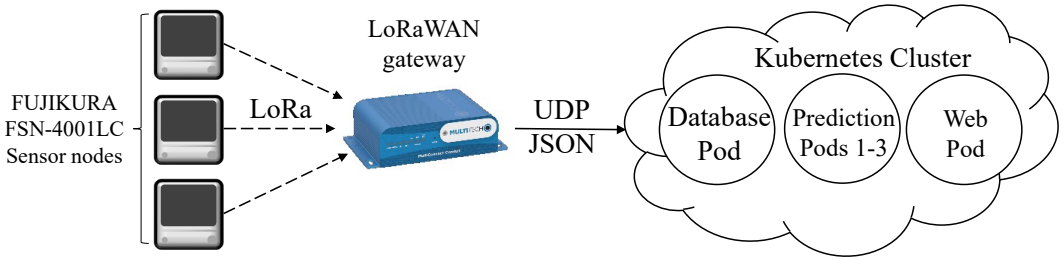


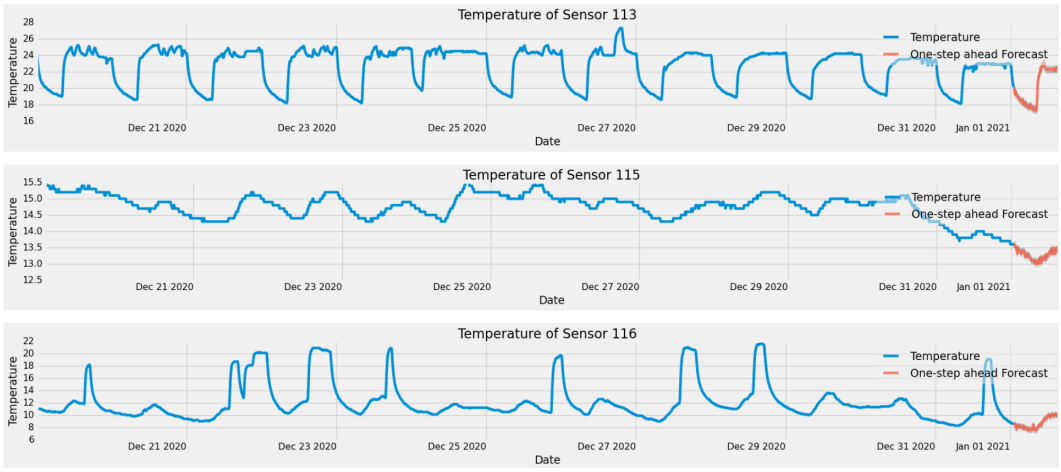
Figure 8.13: Demo 2: sensor network setting.

Demonstration with sensor network

This demonstration shows a real use case. The demonstration collects the temperatures and humidities from three rooms by sensors and predict the future temperatures and humidities through three VNFs, respectively. The data from sensors are stored in a database, which is also a VNF. Finally, a Web server displays the data. The demonstration has five VNFs in this request. The connections between components in this demonstration are shown in Fig. 8.13. The scheduler allocates the VNFs with a load balancer model.

The devices used in this demonstration are three FUJIKURA FSN-4001LC ™LoRaWAN sensor nodes with solar panel unit FSN-4001U and one ™LoRaWAN gateway. Each sensor node provides the data of temperature, humidity, atmospheric pressure, illumination, and motion detection. It does not need any external power supply because of the usage of the solar panel unit, which is environmentally friendly. ™LoRaWAN is used for low-frequency and wide-area data collection. FUJIKURA sensors have been widely used in several areas, such as warehouse temperature monitoring, heatstroke warning, and room occupancy detection.

The data are collected every ten minutes. The original and predicated data are stored in the database by using a seasonal autoregressive integrated moving average model. The data successfully displayed by the Web Pod, as shown in Fig. 8.14(a); the allocations follow the results provided by the model, as shown in Fig. 8.14(b).



(a) Data shown by Web Pod.

```
      NAME           READY   STATUS    RESTARTS   AGE   IP            NODE
      database       1/1     Running   0          45s   10.244.1.5    node1
      prediction113  1/1     Running   0          45s   10.244.1.6    node1
      prediction115  1/1     Running   0          45s   10.244.3.6    node2
      prediction116  1/1     Running   0          45s   10.244.2.4    node3
      web            1/1     Running   0          45s   10.244.3.5    node2
+----+---------------+---------------+---------------+-------+-------+---------------------+
| ID | prediction113 | prediction115 | prediction116 | db    | web   | timestamp           |
+----+---------------+---------------+---------------+-------+-------+---------------------+
|  2 | node1         | node2         | node3         | node1 | node2 | 2020-12-31 19:20:04 |
```

(b) Pod locations (upper) and results provided by the allocation model in database (lower).

Figure 8.14: Demo 2: test results.

8.3 Demonstration of network service header based service function chain application with function allocation model

Network service header (NSH) is a protocol described in [121]. NSH is the SFC encapsulation. It is designed to identify the service function path and the location of current function in an SFC. It can be easily implemented in devices.

A VNF allocation model is designed to obtain a possible allocation strategy for VNFs in SFCs. For example, the model in Chapters 3 was designed to obtain VNF allocations which maximize SFC continuous available time. Its performance in the network needs to be evaluated with different parameters of network elements in addition to the mathematical results. It is not cost

effective and convenient to perform the evaluation with real network devices. One possible solution is to simulate the whole process by software on a single computer. Computational capability required for this software should not be so high. The simulation can run at a low-end computer, such as a laptop. The result of the model is obtained by a mathematical programming approach or a heuristic algorithm. Both of them can run by writing a programming language such as Python. The functions in SFCs need to be allocated according to the computed result in a single application.

This implementation reports an NSH-based SFC application with the cooperation of the VNF allocation model in Chapters 3. This work implements an NSH-based SFC application on Ryu software-defined networking framework [35]. This work implements the functions of classifier, service function forwarder, and SFC proxy [5] on switches by the modification of flow tables which is conducted by the application. This work uses the application to simulate the VNF allocation and the traffic through these functions. The network devices are simulated in Mininet [36], which is connected to the application as a controller. It receives the registration message from allocated VNFs and instructs the actions of switches when they are necessary. We observe that the application allocates the VNFs to corresponding servers automatically and the path of each SFC is configured correctly.

8.3.1 Implementation

Overall structure

Fig. 8.15 overviews the structure of the reported application. A user registers the needed SFCs to database, as indicated by arrow *a*. The VNF allocation model with requested objective is triggered automatically or manually and sends the result to corresponding servers, as indicated by arrow *b*. The servers which receive messages from the allocation model change themselves to the corresponding functions and report their locations to the controller, as indicated by arrow *c*. When the controller receives these reports, it stores the information to database, as indicated by arrow *d*. When the user wants to start a service between two specific nodes, the description of the service is sent to the controller, as indicated by arrow *e*. The controller modifies the flow

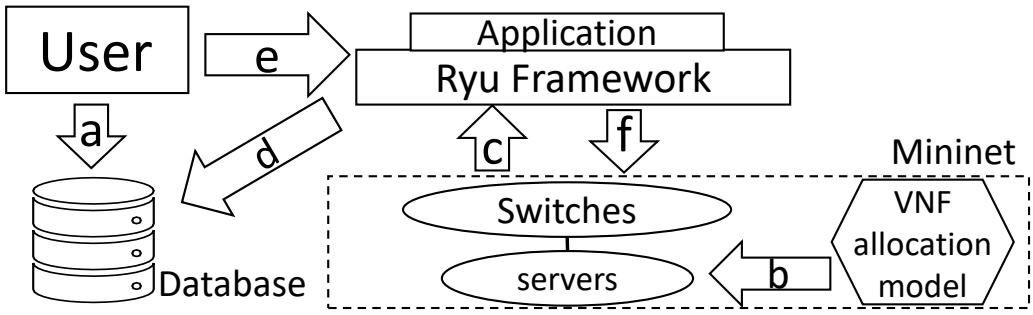


Figure 8.15: Overall structure.

table in the switches when the service traffic goes through, as indicated by arrow f.

Database structure

There are three tables in the database, which are named *flow*, *service*, and *vnf*.

In table *flow*, a user defines the filter rules including Internet protocol (IP) and medium access control (MAC) addresses, ports of transmission control protocol (TCP) and user datagram protocol (UDP), type of protocols, and input port to decide if a flow belongs to a service whose identifier (ID) is "Service_ID". A user uses "ID" to identify a flow and start or delete a flow. The application uses this value to match the detail of the service in table *service*.

Table *vnf* stores the register information from VNF. "ID" is the identification number for each VNF; "name" is a human-readable nickname for the VNF; "dpid" is the datapath ID of connected switch of the VNF; "in_port" is the port of the switch, which connects to the VNF; "Mac_addr" is the Ethernet address of the VNF output port.

Table *service* describes the order of VNFs in each SFC given by users. Each VNF in SFC has a record. "Service_ID" shows the location of the VNF and is used in NSH.

Communication between each components

Allocation results distribution and function register: a program runs in

each server for listening the configuration message from the allocation model. When the allocation results are computed, service ID, name, VNF ID, and other properties of each VNF are sent to the corresponding server. If the listening program receives the message, it chooses and starts a VNF according to the message and sends a register message to the controller through OpenFlow Packet_in message [122] which includes the information of the VNF.

Interaction between controller and switches: this implementation uses four tables in each switch named tables 0, 1, 2, and 3. In tables 0 and 1, the implementation sets a flow entry which lets all packages go to table 3 with priority 0. In table 2, the implementation sets a flow entry which lets all packages be sent to controller with priority 0. In table 3, the implementation sets a flow entry which lets all packages to be forwarded normally.

The OpenFlow connection is established when a switch is connected to controller. A new flow entry with priority 10 in table 0 is added to the new switch. In this flow entry, it matches the VNF register message with preset IP and MAC addresses. The action is to output the message to controller.

When a user starts a new service flow, a flow entry is configured to every switch. It matches the filter conditions of this flow and sends a message with the service ID, input port and the datapath ID to controller. It deletes all preset flow entries related to this flow.

When the controller receives the packages from this service, it looks up table *service* and finds the VNFs which belongs to this service by service ID. For the switch which receives the first frame of the flow, the controller asks the switch to encapsulate the Ethernet frame with NSH. In NSH, service path identification (SPI) is set to service ID, and service index (SI) is set to the length of SFC. An Ethernet frame whose destination is the address of the next VNF is encapsulated out of NSH. For the switch which is connected by the last VNF, it decapsulates NSH. For the other switches which are connected by VNFs, they decapsulate NSH and send packages to VNF. When they receive the packages from VNF, they encapsulate NSH with the same SPI and decrease SI by 1. An Ethernet frame whose destination is the address of the next VNF is encapsulated out of NSH.

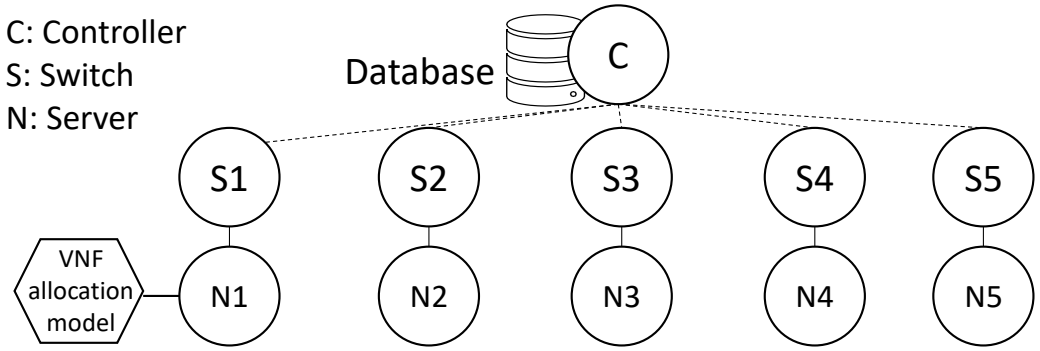


Figure 8.16: Device connection diagram of demonstration.

8.3.2 Demonstration

The demonstration uses Ryu 4.32 (added support for encap() and decap ()), Mininet 2.3.0d6, Open vSwitch 2.12.0, Python 3.6.9, and Sqlite 2.8.17 for the demonstration. The demonstration connects the devices as shown in Fig. 8.16 in Mininet and enable the IP forwarding function for each service node. The demonstration runs allocation model on node N1 simulated by Mininet. The objective of cooperated allocation model is to maximize the continuous available time of SFCs. We observe that the VNFs register themselves to database successfully as shown in Fig. 8.17. The demonstration uses the tracepath command in Mininet to see the path of this flow. We observe that the data is successfully encapsulated by NSH as shown in Fig. 8.18. We observe that the path of a two-function chain is correctly configured from the result shown in Fig. 8.19(a) compared with the old path shown in Fig. 8.19(b).

8.4 Implementation of service function chain deployment with allocation models in Kubernetes

An SFC allocation model obtains the allocations of functions in chains, whose results can be calculated by a mathematical programming approach, a heuristic approach, or a machine learning approach.

	id	name	dpid	in_port	mac_addr
	Filter	Filter	Filter	Filter	Filter
1	1	forward1	1	1	00:00:00:00:00:01
2	2	forward2	2	1	00:00:00:00:00:02
3	3	forward3	4	1	00:00:00:00:00:04
4	4	forward4	5	1	00:00:00:00:00:05
5	5	forward5	3	1	00:00:00:00:00:03

Figure 8.17: Register information in database.

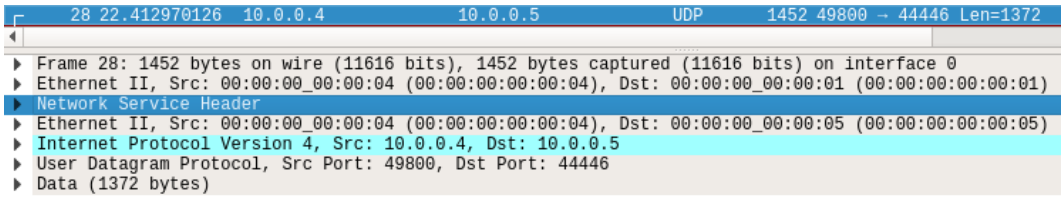


Figure 8.18: Ethernet frame is encapsulated by NSH and another Ethernet frame.

```
mininet> h4 tracepath h5 -l 1400
 1:  10.0.0.1                                              2.139ms
 2:  10.0.0.2                                              1.422ms asymm  1
 3:  10.0.0.5                                              2.177ms reached
     Resume: pmtu 1400 hops 3 back 1
```

(a) Before allocation.

```
mininet> h4 tracepath h5 -l 1400
 1:  10.0.0.5                                              1.685ms reached
     Resume: pmtu 1400 hops 1 back 1
```

(b) After allocation.

Figure 8.19: Result of tracepath.

Kubernetes [9] is a system for automating management of containerized applications. A resource is an endpoint which stores the application interface objectives of a certain kind in Kubernetes. A controller of a resource in Kubernetes adjusts the current state to the expected state. However, no resource defines an SFC with its allocation strategy so that the functions can be

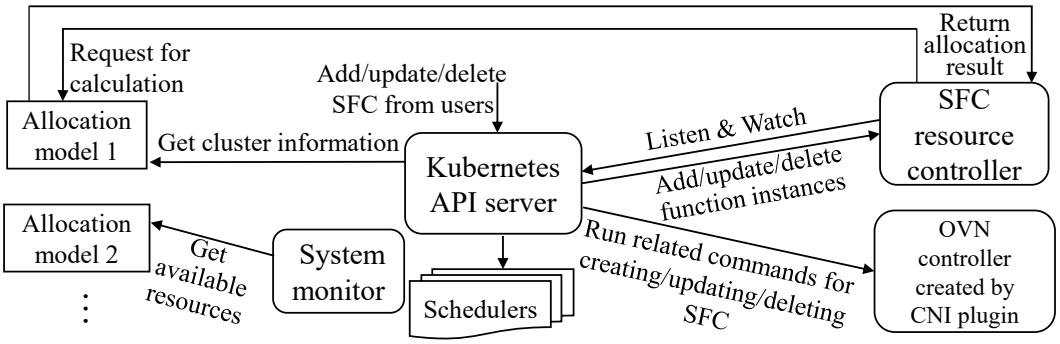


Figure 8.20: Overall structure.

automatically allocated in Kubernetes.

This work reports an implementation of SFCs with the cooperation of the SFC allocation models in Kubernetes. The implementation modifies the open virtual network (OVN) [37] so that it can support the routing for SFC flow, e.g., classifier, service function forwarder, and load balancer for multiple ports. The implementation applies the customized OVN to Kubernetes as a container network interface (CNI) plugin based on Kube-OVN [38]. The implementation adds the definition of the SFC resource and its controller in Kubernetes. The implementation prepares a demonstration to show that the functions in SFCs are allocated to corresponding nodes automatically and the path of each SFC is configured correctly.

8.4.1 Implementation

Overall structure

Fig. 8.20 reports the overall structure. The service providers submit the request of adding, updating, or deleting an SFC instance to the Kubernetes through our defined application interface (API). The SFC resource controller detects the new requirement or changes of existing SFCs. If necessary, new function instances are created and scheduled according to the allocation results. New function instances are created and scheduled according to the allocation results if necessary. At last, the commands for adding, updating, or deleting SFCs are run in the OVN controller requested by the SFC resource controller.

Modification on OVN modules

Compared with the official OVN version, our OVN can route the traffic that meets the matching rules to pass through the specified functions in order to form an SFC in a logical switch for transparent service functions [123]. The modified OVN modules are based on [124]. The implementation adds the following functions: weighted load balancing support for multiple input and output ports of a VNF; flexible matching rules; and the removals of port-security on VNF ports.

Command definitions: compared with the commands in the official OVN version, there are four new commands for adding an SFC. Firstly, a pair of one input port and one output port is given for a service function instance, which is called a port pair (PP). Secondly, if there are multiple PPs for a service function, a port pair group (PPG) is defined. Different weights can be set for different PPs in a PPG. Scenarios that require multiple PPs include: one function instance with multiple ports for inputting or outputting data stream, and multiple function instances that form a logical service function. Thirdly, an SFC is defined by giving ordered PPGs. Finally, the SFC is installed by defining a classifier. The classifier of the SFC sets the flow entries in flow and group tables to route the traffic, which matches the given rules through the functions in order from the source endpoints to the destination endpoints.

Flow table design: two extra flow tables are added to OVN for ingress and egress pipelines, respectively. When a classifier is set, the corresponding flow entries are given. There are four types of flow entries: default entries, inward entries, outward entries, and internal entries. The default entries skip the current flow table for service chaining if there is no match. The inward entries direct the flow that meets the matching conditions to the input port(s) of the first VNF in the chain. The outward entries direct the flow that meets the matching conditions from the output port(s) of the last VNF in the chain to its destination. The internal entries direct the flow that meets the matching conditions from the output port(s) of the previous VNFs in the chain to the input port(s) of the next VNFs in the chain. The matching conditions are decided by the given input port, output port, and other matching conditions. The action is decided by the number of output ports. If there are multiple

input ports for the next hop, a group action is created for load balancing among the multiple input ports.

Design of SFC in Kubernetes

The definition of an SFC instance includes three parts: metadata, specification, and status. Metadata contains the information that distinguishes different SFC instances, e.g., name and creation timestamp. The specification contains the properties of an SFC instance, e.g., the matching conditions, specified allocation methods for creating new instances, and the orders of VNFs in the chain. A specified allocation method contains the container image, communication interfaces, and other related requirements of the required allocation model. The VNFs contain the resource requirements, the container image, and the order of the VNF in the chain. Status contains the latest status of the SFC instance, including the number and the names of containers hosted by the instance; PPs, PPGs, SFC, and classifier created by the OVN controller. The status is updated periodically or after modification.

8.4.2 Demonstrations

There are two demonstrations in this section. Demonstration 1 shows that the created SFC connects existing functions deployed on different physical nodes in a cluster. The matched traffic flows are routed as requested. Demonstration 2 shows that the controller creates the requested SFC and new functions whose location is calculated by an allocation model which aims to reduce the latency from the source to the destination. The demonstration compares the latencies from the source to the destination through the requested SFC, which are deployed by the default controller and our controller, respectively.

The demonstrations implement the design on a six-node Kubernetes cluster including one master node and five worker nodes.

The SFC used for demonstration 1 includes three Pods, which are created in advance along with two Pods for source and destination, respectively. The demonstration uses *traceroute* to show the path between the source and destination. Since the Pods cannot give an Internet control message protocol (ICMP) reply for time-to-live (TTL) expired, the demonstration deploys a

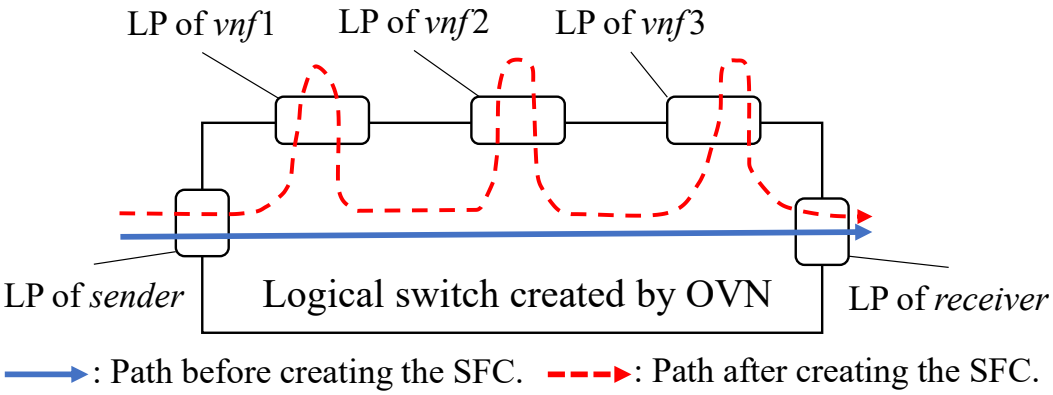
```
ubuntu@master:~$ kubectl get no -o wide
NAME      STATUS   ROLES                  AGE   VERSION   INTERNAL-IP
master    Ready    control-plane,master   52m   v1.22.4   192.168.231.131
worker1   Ready    <none>                 50m   v1.22.4   192.168.231.129
worker2   Ready    <none>                 50m   v1.22.4   192.168.231.132
worker3   Ready    <none>                 50m   v1.22.4   192.168.231.133
worker4   Ready    <none>                 49m   v1.22.4   192.168.231.130
worker5   Ready    <none>                 49m   v1.22.4   192.168.231.128
ubuntu@master:~$ kubectl get pods -o wide
NAME       READY   STATUS    RESTARTS   AGE   IP             NODE
receiver   1/1     Running   0          53s   10.16.0.19     worker1
sender     1/1     Running   0          53s   10.16.0.18     worker3
vnf1       1/1     Running   0          53s   10.16.0.20     worker5
vnf2       1/1     Running   0          53s   10.16.0.21     worker2
vnf3       1/1     Running   0          53s   10.16.0.22     worker3
```

(a) Nodes and Pods lists.

```
root@sender:/traceroute_test# traceroute 10.16.0.19 -z 1
traceroute to 10.16.0.19 (10.16.0.19), 30 hops max, 60 byte packets
 1  10.16.0.19 (10.16.0.19)  2.118 ms  0.587 ms  0.589 ms
```

```
root@sender:/traceroute_test# traceroute 10.16.0.19 -z 1
traceroute to 10.16.0.19 (10.16.0.19), 30 hops max, 60 byte packets
 1  10.16.0.20 (10.16.0.20)  31.221 ms  39.177 ms  39.473 ms
 2  10.16.0.21 (10.16.0.21)  72.157 ms  94.267 ms  78.995 ms
 3  10.16.0.22 (10.16.0.22)  112.349 ms  125.067 ms  101.756 ms
 4  10.16.0.19 (10.16.0.19)  123.105 ms  113.561 ms  121.367 ms
```

(b) *Traceroute* results. (Upper: before creating SFC, lower: after creating SFC.)



(c) Schematic diagram of demonstration 1. (LP: logical port.)

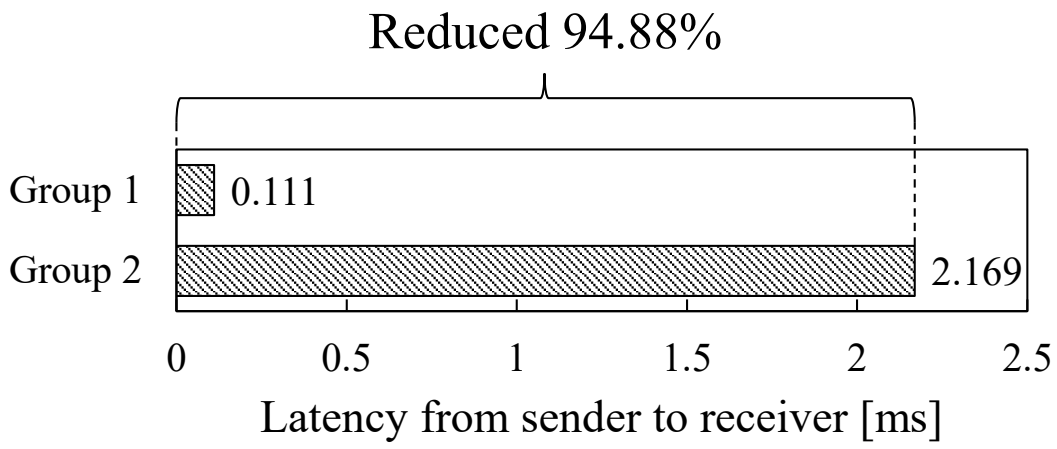
Figure 8.21: Results and setting in demonstration 1.

```
ubuntu@master:~$ kubectl get pods -o wide
NAME       READY   STATUS    RESTARTS   AGE   IP             NODE
receiver   1/1     Running   0          12m   10.16.0.29     worker1
sender     1/1     Running   0          12m   10.16.0.28     worker1
vnf1       1/1     Running   0          12m   10.16.0.30     worker1
vnf2       1/1     Running   0          12m   10.16.0.31     worker1
vnf3       1/1     Running   0          12m   10.16.0.32     worker1
ubuntu@master:~$ kubectl get pods -o wide
NAME       READY   STATUS    RESTARTS   AGE   IP             NODE
receiver   1/1     Running   0          93s   10.16.0.34     worker3
sender     1/1     Running   0          93s   10.16.0.33     worker1
vnf1       1/1     Running   0          93s   10.16.0.35     worker5
vnf2       1/1     Running   0          93s   10.16.0.36     worker2
vnf3       1/1     Running   0          93s   10.16.0.37     worker1
```

(a) Pod allocations of two groups. (Upper: group 1, lower: group 2.)

```
sender:   13:33:29.896    sender:   13:42:30.395
vnf1:     13:33:29.918    vnf1:     13:42:31.011
vnf2:     13:33:29.957    vnf2:     13:42:31.777
vnf3:     13:33:29.989    vnf3:     13:42:32.142
receiver: 13:33:30.007    receiver: 13:42:32.564
```

(b) File received in two groups. (Left: group 1, right: group 2.)



(c) Latency comparison between groups 1 and 2.

Figure 8.22: Results and setting in demonstration 2.

program by using Python and Scapy, which can send an ICMP reply for TTL expired and forward the UDP packet if TTL is not expired. The results of demonstration 2 are shown in Fig. 8.21.

In demonstration 2, the demonstration sets the incoming and outgoing network latency of five worker nodes to 100, 200, 300, 400, and 500 [ms], respectively. The source sends a text file to the destination. The function instances in the chain add their names and timestamp to the file. In group 1, three Pods are allocated by the model, and an SFC is created to connect them simultaneously. In group 2, three Pods are created in advance, and then an SFC is created to connect them in order. The results of demonstration 2 are shown in Fig. 8.22. We can observe that the latency in the SFC is reduced with the allocation model.

8.5 Summary

This chapter introduced four demonstrations on the cooperation between resource allocation models and containerized networks and SDN networks.

The first implementation designed and implemented a custom resource and the corresponding controller in Kubernetes to manage the VNF diversity and redundancy jointly. The demonstration of this implementation validated that the controller automatically manages the resources correctly, improves the resource utilization, and increases the number of acceptable requests.

The second implementation designed and implemented a Pod scheduler in Kubernetes. The implementation introduced the structure and the components in the scheduler in details. The implementation provided the two demonstrations of the introduced scheduler, which confirmed that the effectiveness of the scheduler cooperating with the examined allocation models.

The third implementation designed and implemented a network service header based service function chain application which can be cooperated with the resource allocation models for SFCs in the SDN environment. Demonstration validates that the application allocates functions by using the allocation from the model automatically and runs service function chains correctly.

The forth implementation designed and implemented an OVN based SFC-compatible network plugin for Kubernetes. The implementation implements

two controllers for creating SFCs among existing functions and SFC deployments without existing functions which can be cooperated with allocation models. The plugin allocates the functions in chains according to the given models and connects each function in chains by setting suitable flow entries in Kubernetes. The demonstrations of this implementation validate the implementation at last.

Chapter 9

Conclusions

Along with the large-scale deployment of VNFs in the network, failures of hardware and software occur from time to time. How to improve the fault tolerance of the network is an important issue that must be addressed in network virtualization. This thesis classifies failures into three types and studies five specific problems about the resource allocations in NFV with relative implementations in containerized networks and SDN.

Firstly, this thesis proposed a VNF allocation model for improving the continuous available time of service function chains assuming the existence of known availability schedules. This work formulated the proposed model as an ILP problem that maximizes the minimum number of the longest continuous available time slots in each SFC. This work proved that the decision version of the VNF allocation problem (VNFA) in the proposed model is NP-complete. Numerical results showed that the proposed model improves the continuous available time of SFCs, compared with the persistent allocation model, the single-slot allocation model, and the double-slot allocation model. This work observed that the proposed model reduces the path lengths of SFCs as it computes VNF allocation and SFC routes at the same time. This work developed a heuristic algorithm to yield practical the solution time. In the cases examined, the developed heuristic algorithm reduces the average computation time with some penalty in performance compared with the ILP approach.

Secondly, this thesis proposed a primary and backup VNF placement model for improving the continuous available time of service function chains by avoid-

ing the interruptions caused by unavailable nodes and function reallocations. This work formulated the proposed model as an ILP problem that maximizes the minimum number of the longest continuous available time slots in each SFC by considering deterministic availability schedules. This work extended the proposed model to a network-aware one with considering a routing problem. Evaluation results showed that the proposed model improves the continuous available time of SFCs, compared with the baseline models in the examined cases. The network-aware proposed model reduces the latencies of services with keeping the same values of SCATs of services compared with the non-network-aware proposed model. This work introduced an algorithm that estimates the number of key unavailabilities at each time slot, which indicates the number of unavailable nodes that are required to be eliminated priorly. The number of key unavailabilities helps SPs to increase the service continuous available time with the least cost of failure recovery. This work analyzed the impact of different types of availability schedules on the proposed model. In the examined test cases, the proposed model provides a lower improvement on SSCATs if the positions of unavailabilities are more compact, and provides a higher improvement on SSCATs if the positions of unavailabilities are more sparse. This work developed and analyzed a heuristic algorithm to speed up the computation for the case that the problem size increases. The heuristic algorithm reduces 66.75% computation time with a 1.57% performance degradation in terms of the objective value on average in the examined test cases. This work provided the discussion on dealing with multiple replicas of a function and their backups.

Thirdly, this thesis proposed a robust VNF allocation model for improving the continuous available time of service function chains with considering the uncertain availability schedule. This work formulated the proposed model as an MILP problem. Numerical results showed that the proposed model improves the continuous available time of SFCs, compared with the persistent allocation model, the single-slot allocation model, and the double-slot allocation model in both deterministic and uncertain availability schedules. In the cases examined, the proposed model can provide longer continuous available time slots compared with the three baseline models under different levels of the robustness of uncertain availability schedule. The developed heuristic al-

gorithm reduces the computation time by 99.85% compared with the MILP approach with a limited performance penalty by 3.37% in our evaluations. We evaluated the relationship between availability schedules and objective values. The size of the uncertainty set can be reduced according to our observations. This work gave three discussions of the proposed model: for maintenance ability, for multiple unavailability periods on each node, and for unpredictable unavailabilities. In addition, this work provided three directions to extend the proposed model.

Fourthly, this thesis proposed a k-resilient VNF allocation model for reducing the E2E latency during recovery migration with VNF diversity and redundancy. This work formulated the proposed model as an MILP problem. This work proved that the subproblem of the VNF allocation problem in the proposed model is NP-complete. Numerical results showed that the proposed model reduces the E2E latencies between the primary replicas and the backup replicas, compared with five baseline models. This work developed and analyzed two approximate algorithms to solve the proposed model in a shorter computation time on average at the cost of accuracy compared with the MILP approach. In the examined cases, the greedy approach reduces 99.98% computation time compared with the MILP approach at the cost of 31.96% performance loss on average. The greedy and PH approaches become valuable to be applied as the problem size increases. The performance loss of the PH approach is lower than that of the greedy approach, which is 23.76% on average in the examined cases. This work evaluated the impact of replica pool design and gave the suggestions to SPs for better allocations. This work investigated the relationship between the given conditions and resiliency level. This work derived the theorems to give the upper and lower bounds of the resiliency level. This work showed the practical methods to manage the replicas with the required amount of processing ability and recover the data after the failures.

Fifthly, this thesis proposed a resilient VNF allocation model for increasing the number of accepted requests with ensuring fault tolerance and considering VNF diversity in a dynamic scenario. This work developed an RL-based approach for solving the proposed model. This work designed the procedures for dealing with arrival requests in the approach. Numerical results showed

that the proposed model increases the number of accepted requests compared with the two baseline models. Compared with RDM which randomly determines the allocations of replica instances, the proposed model accepts 20.85 times more requests on average in the examined cases. Compared with SAM which determines the allocations of replica instances independently at each time slot, the proposed model accepts 1.58 times more requests on average in the examined cases. Although the proposed model improves the performance at the cost of a longer computation time compared with the baseline models, it can still determine the allocation of a request in tens of milliseconds in the examined cases. This work evaluated the impact of additional training data on the performance of the proposed model. Suitable additional data can improve the performance of the proposed model. Excessive training is harmful, which leads to the performance degradation of the proposed model.

Sixthly, this thesis introduced four demonstrations on the cooperation between resource allocation models and containerized networks and SDN networks. The first implementation designed and implemented a custom resource and the corresponding controller in Kubernetes to manage the VNF diversity and redundancy jointly. The demonstration of this implementation validated that the controller automatically manages the resources correctly, improves the resource utilization, and increases the number of acceptable requests. The second implementation designed and implemented a Pod scheduler in Kubernetes. The implementation introduced the structure and the components in the scheduler in details. The implementation provided the two demonstrations of the introduced scheduler, which confirmed that the effectiveness of the scheduler cooperating with the examined allocation models.The third implementation designed and implemented a network service header based service function chain application which can be cooperated with the resource allocation models for SFCs in the SDN environment. Demonstration validates that the application allocates functions by using the allocation from the model automatically and runs service function chains correctly. The forth implementation designed and implemented an OVN based SFC-compatible network plugin for Kubernetes. The implementation implements two controllers for creating SFCs among existing functions and SFC deployments without existing functions which can be cooperated with allocation models. The plugin

allocates the functions in chains according to the given models and connects each function in chains by setting suitable flow entries in Kubernetes. The demonstrations of this implementation validate the implementation at last.

The five proposed models studied five typical application scenarios of fault-tolerant resource allocations in network virtualization with considering the deterministic, uncertain, and unknown failures. This work provided different approaches with theoretical analyses in each model. The implementations of the proposed model can be cooperated with the five models. SPs can select appropriate models with suitable implementation methods according to the specific requirements to achieve a reliable and economical network virtualization environment.

For future works, the proposed models can be extended to consider resource migration against network update with considering the update frequency to guarantee consistency properties for the configured forwarding rules and paths (e.g., to avoid forwarding loops which can quickly deplete switch buffers and harm the availability and connectivity provided by a network). Previous researches have never considered resource allocation models for VNFs placement incorporating network updates; the introduced network resource allocation models for VNF placement are designed under the assumption that the network states and topologies are static. Against the network update, optimizing VNF resource allocation and steering its traffic iteratively may lead to a disruptive network that will never converge. Since more intricate processes need more calculation time, they are not appropriate for real-time decision-making in a flexible data center network. Additionally, topology modification and traffic shaping have costs. A change that occurs too often can disrupt routing and transport protocols and lower network usage because the network is interrupted during the transition phase. The future works are expected to handle the balance in resource migration against network update: the higher frequency of updates will obviously improve the adaptability of resource allocation against the updated network (this can be reflected in the service delay of post-migration and consistency of network); but higher frequency will bring troubles in consistency of network state, unavailability time against service interruptions, and extra bandwidth and latency due to synchronization against resource migration.

Appendix A

Linearization of proposed models

This thesis introduces the following linearization process for the proposed models.

$$x = \max_i\{y_i\} \Leftrightarrow \begin{cases} x \le y_i + (1-\delta_i)\cdot B, \forall i \in Y \\ x \ge y_i - (1-\delta_i)\cdot B, \forall i \in Y \\ y_i \ge (\delta_i - 1)\cdot B + y_j, \forall i \in Y, j \in Y \backslash \{i\} \\ \sum_{i\in Y} \delta_i = 1 \\ x \ge y_i, \forall i \in Y \\ \delta_i \in \{0,1\}, \forall i \in Y \end{cases} \tag{A.1}$$

B is sufficiently large to ensure that its value is larger than $y_i, i \in Y$.

$x = \min_i\{y_i\}$ can be expressed in linear form by:

$$x = \min_i\{y_i\} \Leftrightarrow \begin{cases} x \le y_i + (1-\delta_i)\cdot B, \forall i \in Y \\ x \ge y_i - (1-\delta_i)\cdot B, \forall i \in Y \\ y_i \le (1-\delta_i)\cdot B + y_j, \forall i \in Y, j \in Y \backslash \{i\} \\ \sum_{i\in Y} \delta_i = 1 \\ x \le y_i, \forall i \in Y \\ \delta_i \in \{0,1\}, \forall i \in Y, \end{cases} \tag{A.2}$$

where B is sufficiently large to ensure that its value is larger than $y_i, i \in Y$.

For binary decision variables $y_i \in Y$, the operation $x = \vee_{i \in Y} y_i = y_1 \vee y_2 \vee \cdots \vee y_{|Y|}$ can be expressed in linear form as follows:

$$x = \vee_{i \in Y} y_i \Leftrightarrow \begin{cases} x \geq \frac{1}{|Y|} \cdot \sum_{i \in Y} y_i \\ x \leq \sum_{i \in Y} y_i \\ x, y_i \in \{0,1\}, \forall i \in Y. \end{cases} \tag{A.3}$$

The relationship

$$\begin{aligned} &\textbf{If} \quad a \leq x \leq b \quad \textbf{then} \\ &\qquad e = 1 \\ &\textbf{Else} \\ &\qquad e = 0 \end{aligned} \tag{A.4a}$$

$$e \in [0,1] \tag{A.4b}$$

$$x \in \mathbb{Z}. \tag{A.4c}$$

can be linearized by using the following equations:

$$x - a + \epsilon \leq \alpha \cdot B \tag{A.5a}$$

$$x - a + \epsilon \geq (\alpha - 1) \cdot B \tag{A.5b}$$

$$b - x + \epsilon \leq \beta \cdot B \tag{A.5c}$$

$$b - x + \epsilon \geq (\beta - 1) \cdot B \tag{A.5d}$$

$$e \leq \alpha \tag{A.5e}$$

$$e \leq \beta \tag{A.5f}$$

$$e \geq \alpha + \beta - 1 \tag{A.5g}$$

$$e, \alpha, \beta \in [0,1] \tag{A.5h}$$

$$x \in \mathbb{Z}. \tag{A.5i}$$

a, b are given integer parameters. ϵ is a given positive parameter: $0 < \epsilon < 1$. B is a number which is larger than $x - a + \epsilon$ and $b - x + \epsilon$.

Exclusive nor operation ($\odot$) can be expressed in linear form.

$$x = a \odot b \Leftrightarrow \begin{cases} x = 1 - a - b + 2 \cdot h \\ h \leq a \\ h \leq b \\ h \geq a + b - 1 \\ x, z, a, h \in \{0, 1\} \end{cases} \tag{A.6}$$

For binary variables $y_i \in Y$, the operation $x = \prod_{i \in Y} y_i = y_1 \wedge y_2 \wedge \cdots \wedge y_{|Y|}$ can be expressed in linear form.

$$x = \prod_{i \in Y} y_i \Leftrightarrow \begin{cases} x \leq y_i, \forall i \in Y \\ x \geq \sum_{i \in Y} y_i - |Y| + 1 \\ x, y_i \in \{0, 1\}, \forall i \in Y \end{cases} \tag{A.7}$$

Bibliography

[1] A. Lombardo, A. Manzalini, V. Riccobene, and G. Schembra, "An analytical tool for performance evaluation of software defined networking services," in *2014 IEEE Netw. Oper. and Manag. Symp.*, 2014, pp. 1–7.

[2] M.-T. Thai, Y.-D. Lin, and Y.-C. Lai, "A joint network and server load balancing algorithm for chaining virtualized network functions," in *2016 IEEE Int. Conf. on Commun.* IEEE, 2016, pp. 1–6.

[3] J. G. Herrera and J. F. Botero, "Resource allocation in NFV: A comprehensive survey," *IEEE Trans. Netw. Service Manag.*, vol. 13, no. 3, pp. 518–532, 2016.

[4] J. Fan, Z. Ye, C. Guan, X. Gao, K. Ren, and C. Qiao, "Grep: Guaranteeing reliability with enhanced protection in nfv," in *Proceedings of the 2015 ACM SIGCOMM Workshop on Hot Topics in Middleboxes and Network Function Virtualization.* Association for Computing Machinery, 2015, p. 13–18. [Online]. Available: https://doi.org/10.1145/2785989.2786000

[5] E. J. Halpern and E. C. Pignataro, "Service function chaining (SFC) architecture," RFC 2481, Oct. 2015.

[6] ETSI, *Network Functions Virtualisation (NFV); Management and Orchestration*, ETSI Std. GS NFV-MAN 001, Rev. V1.1.1, Jan. 2014. [Online]. Available: https://www.etsi.org/deliver/etsi_gs/nfv-man/001_099/001/01.01.01_60/gs_nfv-man001v010101p.pdf

[7] OpenInfra Foundation, "Open source cloud computing infrastructure - openstack," https://www.openstack.org/, accessed Dec. 15, 2022.

[8] ——, "Welcome to the heat documentation! — openstack-heat 19.1.0.dev14 documentation," https://docs.openstack.org/heat/latest/, accessed Dec. 15, 2022.

[9] The Kubernetes Authors, "Kubernetes," https://kubernetes.io/, accessed Dec. 15, 2022.

[10] A. Alleg, T. Ahmed, M. Mosbah, and R. Boutaba, "Joint diversity and redundancy for resilient service chain provisioning," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 7, pp. 1490–1504, 2020.

[11] M. T. Beck, J. F. Botero, and K. Samelin, "Resilient allocation of service function chains," in *2016 IEEE Conf. on Netw. Function Virtualization and Software Defined Netw.* IEEE, 2016, pp. 128–133.

[12] R. Wen, G. Feng, J. Tang, T. Q. Quek, G. Wang, W. Tan, and S. Qin, "On robustness of network slicing for next-generation mobile networks," *IEEE Trans. Commun.*, vol. 67, no. 1, pp. 430–444, 2018.

[13] M. Wessler, *Oracle DBA on Unix and Linux.* USA: Sams, 2001.

[14] T. Sato, F. He, E. Oki, T. Kurimoto, and S. Urushidani, "Implementation and testing of failure recovery based on backup resource sharing model for distributed cloud comp. system," in *2018 IEEE 7th Int. Conf. on Cloud Netw.* IEEE, 2018, pp. 1–3.

[15] L. Sun, J. An, Y. Yang, and M. Zeng, "Recovery strategies for service composition in dynamic network," in *2011 Int. Conf. on Cloud and Service Comput.* IEEE, 2011, pp. 60–64.

[16] H. Abid and N. Samaan, "A novel scheme for node failure recovery in virtualized networks," in *2013 IFIP/IEEE Int. Symp. on Integr. Netw. Manag.* IEEE, 2013, pp. 1154–1160.

[17] M. Raza, V. Samineni, and W. Robertson, "Physical and logical topology slicing through SDN," in *2016 IEEE Canadian Conf. on Elect. and Comput. Eng.* IEEE, 2016, pp. 1–4.

[18] M. Zhu, F. He, and E. Oki, "Optimal multiple backup resource allocation with workload-dependent failure probability," in *2020 IEEE Global Commun. Conf.*, 2020, pp. 1–6.

[19] ETSI, *Network Functions Virtualisation (NFV); Virtual Network Functions Architecture*, ETSI Std. GS NFV-SWA 001, Rev. V1.1.1, Dec. 2014. [Online]. Available: https://www.etsi.org/deliver/etsi_gs/NFV-SWA/001_099/001/01.01.01_60/gs_NFV-SWA001v010101p.pdf

[20] J. P. Sterbenz, D. Hutchison, E. K. Çetinkaya, A. Jabbar, J. P. Rohrer, M. Schöller, and P. Smith, "Resilience and survivability in communication networks: Strategies, principles, and survey of disciplines," *Comput. Networks*, vol. 54, no. 8, pp. 1245–1265, 2010.

[21] F. Machida, Masahiro Kawato, and Y. Maeno, "Redundant virtual machine placement for fault-tolerant consolidated server clusters," in *2010 IEEE Netw. Oper. and Manage. Symp. (NOMS)*, 2010, pp. 32–39.

[22] S. G. Kulkarni, K. K. Ramakrishnan, and T. Wood, "Managing state for failure resiliency in network function virtualization," in *2020 IEEE Int. Symp. on Local and Metropolitan Area Netw.*, Jul. 2020, pp. 1–6.

[23] A. Mouaci, E. Gourdin, I. LjubiC, and N. Perrot, "Virtual network functions placement and routing problem: Path formulation," in *2020 IFIP Netw. Conf.*, Jun. 2020, pp. 55–63.

[24] B. Yi, X. Wang, K. Li, S. k. Das, and M. Huang, "A comprehensive survey of network function virtualization," *Comput. Networks*, vol. 133, pp. 212–262, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128618300306

[25] S. Khebbache, M. Hadji, and D. Zeghlache, "Scalable and cost-efficient algorithms for VNF chaining and placement problem," in *2017 20th Conf. on Innovations in Clouds, Internet and Netw.*, 2017, pp. 92–99.

[26] M. Mechtri, C. Ghribi, and D. Zeghlache, "VNF placement and chaining in distributed cloud," in *2016 IEEE 9th Int. Conf. on Cloud Comput.*, 2016, pp. 376–383.

[27] Y. Sang, B. Ji, G. R. Gupta, X. Du, and L. Ye, "Provably efficient algorithms for joint placement and allocation of virtual network functions," in *2017 IEEE Conf. on Comput. Commun.*, 2017, pp. 1–9.

[28] Q. Zhang, Y. Xiao, F. Liu, J. C. Lui, J. Guo, and T. Wang, "Joint optimization of chain placement and request scheduling for network function virtualization," in *2017 IEEE 37th Int. Conf. on Distrib. Comput. Syst.*, 2017, pp. 731–741.

[29] Y. Xiao, Q. Zhang, F. Liu, J. Wang, M. Zhao, Z. Zhang, and J. Zhang, "NFVdeep: Adaptive online service function chain deployment with deep reinforcement learning," in *2019 IEEE/ACM 27th International Symposium on Quality of Service (IWQoS)*, 2019, pp. 1–10.

[30] L. Wang, W. Mao, J. Zhao, and Y. Xu, "DDQP: A double deep Q-learning approach to online fault-tolerant SFC placement," *IEEE Trans. Netw. Service Manag.*, vol. 18, no. 1, pp. 118–132, 2021.

[31] S. I. Kim and H. S. Kim, "Method for VNF placement for service function chaining optimized in the NFV environment," in *2019 11th Int. Conf. on Ubiquitous and Future Netw.*, 2019, pp. 721–724.

[32] M. Johnston, H.-W. Lee, and E. Modiano, "A robust optimization approach to backup network design with random failures," *IEEE/ACM Trans. Netw.*, vol. 23, no. 4, pp. 1216–1228, 2015.

[33] F. He, T. Sato, and E. Oki, "Backup resource allocation model for virtual networks with probabilistic protection against multiple facility node failures," in *2019 15th Int. Conf. on the Des. of Reliable Commun. Netw.* IEEE, 2019, pp. 37–42.

[34] M. Ito, F. He, and E. Oki, "Robust optimization model for probabilistic protection under uncertain virtual machine capacity in cloud," in *2020 16th Int. Conf. on the Des. of Reliable Commun. Netw.*, 2020, pp. 1–8.

[35] Ryu SDN Framework Community, "Ryu SDN framework," https://osrg.github.io/ryu/index.html, accessed Dec. 21, 2022.

[36] Mininet Team, "Mininet: An instant virtual network on your laptop (or other pc)," mininet.org/, accessed Dec. 21, 2022.

[37] The OVN community, "OVN, Open Virtual Network :: OVN project documentation website," https://www.ovn.org/en/, accessed Oct. 12, 2021.

[38] The Kube-OVN community, "Kube-OVN | the most advanced kubernetes network fabric for enterprises," https://www.kube-ovn.io/, accessed Oct. 12, 2021.

[39] A. Zamani and S. Sharifian, "A novel approach for service function chain (SFC) mapping with multiple SFC instances in a fog-to-cloud computing system," in *2018 4th Iranian Conf. on Signal Process. and Intell. Syst.* IEEE, 2018, pp. 48–52.

[40] N. Hyodo, T. Sato, R. Shinkuma, and E. Oki, "Virtual network function placement for service chaining by relaxing visit order and non-loop constraints," *IEEE Access*, vol. 7, pp. 165 399–165 410, 2019.

[41] A. Alleg, T. Ahmed, M. Mosbah, R. Riggio, and R. Boutaba, "Delay-aware VNF placement and chaining based on a flexible resource allocation approach," in *2017 13th Int. Conf. on Netw. and Service Manag.*, Nov 2017, pp. 1–7.

[42] L. Qu, M. Khabbaz, and C. Assi, "Reliability-aware service chaining in carrier-grade softwarized networks," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 3, pp. 558–573, March 2018.

[43] P. T. A. Quang, A. Bradai, K. D. Singh, G. Picard, and R. Riggio, "Single and multi-domain adaptive allocation algorithms for VNF forwarding graph embedding," *IEEE Trans. Netw. Service Manag.*, vol. 6, no. 1, pp. 98–112, 2019.

[44] F. Carpio, A. Jukan, and R. Pries, "Balancing the migration of virtual network functions with replications in data centers," in *2018 IEEE/IFIP Netw. Oper. and Manag. Symp.*, 2018, pp. 1–8.

[45] K. A. Noghani, A. Kassler, and J. Taheri, "On the cost-optimality trade-off for service function chain reconfiguration," in *2019 IEEE 8th Int. Conf. on Cloud Netw.*, 2019, pp. 1–6.

[46] A. Engelmann and A. Jukan, "A reliability study of parallelized vnf chaining," in *2018 IEEE Int. Conf. on Commun.*, 2018, pp. 1–6.

[47] A. Engelmann, A. Jukan, and R. Pries, "On coding for reliable vnf chaining in dcns," in *2019 15th Int. Conf. on the Des. of Reliable Commun. Netw.*, 2019, pp. 83–90.

[48] L. Zhang, Y. Wang, X. Qiu, and H. Guo, "Redundancy mechanism of service function chain with node-ranking algorithm," in *2019 IFIP/IEEE Symp. on Integr. Netw. and Service Manag.*, 2019, pp. 586–589.

[49] M. Wang, B. Cheng, and J. Chen, "Joint availability guarantee and resource optimization of virtual network function placement in data center networks," *IEEE Trans. Netw. Service Manag.*, vol. 17, no. 2, pp. 821–834, 2020.

[50] J. Fan, M. Jiang, and C. Qiao, "Carrier-grade availability-aware mapping of service function chains with on-site backups," in *2017 IEEE/ACM 25th Int. Symp. on Qual. of Service (IWQoS)*, June 2017, pp. 1–10.

[51] J. Fan, M. Jiang, O. Rottenstreich, Y. Zhao, T. Guan, R. Ramesh, S. Das, and C. Qiao, "A framework for provisioning availability of NFV in data center networks," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 10, pp. 2246–2259, Oct 2018.

[52] J. Kong, I. Kim, X. Wang, Q. Zhang, H. C. Cankaya, W. Xie, T. Ikeuchi, and J. P. Jue, "Guaranteed-availability network function virtualization with network protection and vnf replication," in *2017 IEEE Global Commun. Conf.*, 2017, pp. 1–6.

[53] A. L. Soyster, "Convex programming with set-inclusive constraints and applications to inexact linear programming," *Operations research*, vol. 21, no. 5, pp. 1154–1157, 1973.

[54] A. Ben-Tal, L. El Ghaoui, and A. Nemirovski, *Robust Optimization*. Princeton University Press, 2009.

[55] D. Bertsimas and M. Sim, "The price of robustness," *Operations research*, vol. 52, no. 1, pp. 35–53, 2004.

[56] ——, "Robust discrete optimization and network flows," *Mathematical programming*, vol. 98, no. 1-3, pp. 49–71, 2003.

[57] F. He, T. Sato, B. C. Chatterjee, T. Kurimoto, S. Urushidani, and E. Oki, "Robust optimization model for backup resource allocation in cloud provider," in *2018 IEEE Int. Conf. on Commun.* IEEE, 2018, pp. 1–6.

[58] E. Bin, O. Biran, O. Boni, E. Hadad, E. K. Kolodner, Y. Moatti, and D. H. Lorenz, "Guaranteeing high availability goals for virtual machine placement," in *2011 31st Int. Conf. on Distrib. Comput. Syst.*, Jun. 2011, pp. 700–709.

[59] A. Zhou, S. Wang, B. Cheng, Z. Zheng, F. Yang, R. N. Chang, M. R. Lyu, and R. Buyya, "Cloud service reliability enhancement via virtual machine placement optimization," *IEEE Trans. Serv. Comput.*, vol. 10, no. 6, pp. 902–913, 2017.

[60] A. Hmaity, M. Savi, F. Musumeci, M. Tornatore, and A. Pattavina, "Virtual network function placement for resilient service chain provisioning," in *2016 8th Int. Workshop on Resilient Netw. Des. and Model.*, Sep. 2016, pp. 245–252.

[61] S. Yang, F. Li, R. Yahyapour, and X. Fu, "Delay-sensitive and availability-aware virtual network function scheduling for nfv," *IEEE Trans. Serv. Comput.*, vol. 15, no. 1, pp. 188–201, 2022.

[62] M. Schöller, M. Stiemerling, A. Ripke, and R. Bless, "Resilient deployment of virtual network functions," in *2013 5th Int. Congr. on Ultra Modern Telecommun. and Control Syst. and Workshops*, Sep. 2013, pp. 208–214.

[63] L. Qu, C. Assi, K. Shaban, and M. J. Khabbaz, "A reliability-aware network service chain provisioning with delay guarantees in NFV-enabled enterprise datacenter networks," *IEEE Trans. Netw. Service Manag.*, vol. 14, no. 3, pp. 554–568, Sep. 2017.

[64] J. Liu, W. Lu, F. Zhou, P. Lu, and Z. Zhu, "On dynamic service function chain deployment and readjustment," *IEEE Trans. Netw. Service Manag.*, vol. 14, no. 3, pp. 543–553, 2017.

[65] G. Wang, G. Feng, T. Q. S. Quek, S. Qin, R. Wen, and W. Tan, "Reconfiguration in network slicing—optimizing the profit and performance," *IEEE Trans. Netw. Service Manag.*, vol. 16, no. 2, pp. 591–605, 2019.

[66] P. M. Mohan and M. Gurusamy, "Resilient VNF placement for service chain embedding in diversified 5G network slices," in *2019 IEEE Global Commun. Conf.*, Dec. 2019, pp. 1–6.

[67] N. Hyodo, T. Sato, R. Shinkuma, and E. Oki, "Resilient virtual network function placement model based on recovery time objectives," in *2020 IEEE 21st Int. Conf. on High Perform. Switching and Routing*, May 2020, pp. 1–7.

[68] S. Lin, W. Liang, and J. Li, "Reliability-aware service function chain provisioning in mobile edge-cloud networks," in *2020 29th Int. Conf. on Comput. Commun. and Netw.*, Aug. 2020, pp. 1–9.

[69] Y. Chen and J. Wu, "Service function chain deployment with guaranteed resilience," in *2020 IEEE 17th Int. Conf. on Mobile Ad Hoc and Sensor Syst.*, Dec. 2020, pp. 585–593.

[70] W. Xia, P. Zhao, Y. Wen, and H. Xie, "A survey on data center networking (DCN): Infrastructure and operations," *IEEE Commun. Surveys Tut.*, vol. 19, no. 1, pp. 640–656, 2017.

[71] ETSI, *Network Functions Virtualisation (NFV); Resiliency Requirements*, ETSI Std. GS NFV-REL 001, Rev. V1.1.1, Jan. 2015. [Online]. Available: https://www.etsi.org/deliver/etsi_gs/NFV-REL/001_099/001/01.01.01_60/gs_NFV-REL001v010101p.pdf

[72] M. Akbari, M. R. Abedi, R. Joda, M. Pourghasemian, N. Mokari, and M. Erol-Kantarci, "Age of information aware VNF scheduling in industrial IoT using deep reinforcement learning," *IEEE J. Sel. Areas Commun.*, vol. 39, no. 8, pp. 2487–2500, 2021.

[73] R. Kang, F. He, T. Sato, and E. Oki, "Virtual network function allocation to maximize continuous available time of service function chains," in *2019 IEEE 8th Int. Conf. on Cloud Netw. (CloudNet)*. IEEE, Nov 2019, pp. 1–6.

[74] ——, "Virtual network function allocation to maximize continuous available time of service function chains with availability schedule," *IEEE Trans. Netw. Service Manag.*, vol. 18, no. 2, pp. 1556–1570, 2020.

[75] P. T. A. Quang, A. Bradai, K. DeepSingh, and R. Riggio, "A^2VF: Adaptive allocation for virtual network functions in wireless access networks," in *2018 IEEE 19th Int. Symp. on a World of Wireless, Mobile and Multimedia Netw.* IEEE, 2018, pp. 1–9.

[76] J. F. G. Fernández and A. C. Márquez, *Maintenance Management in Network Utilities: Framework and Practical Implementation*. Springer Science & Business Media, 2012.

[77] E. T. Jon Kleinberg, *Algorithm Design*. Pearson Education, Inc., 2006.

[78] D. S. J. Michael R. Garey, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[79] R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of computer computations*. Springer, 1972, pp. 85–103.

[80] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[81] M. Mitchell, *An Introduction to Genetic Algorithms*. MIT press, 1998.

[82] IBM Knowledge Center, "Introducing IBM ILOG CPLEX optimization studio v12.9.0.0," https://www.ibm.com/support/knowledgecenter/

SSSA5P_12.9.0/ilog.odms.studio.help/Optimization_Studio/topics/COS_home.html, accessed May 8, 2020.

[83] R. Kang, F. He, and E. Oki, "Optimal virtual network function placement in chains using backups with availability schedule," in *2020 IEEE 9th Int. Conf. on Cloud Netw.* IEEE, 2020, pp. 1–6.

[84] ——, "Virtual network function allocation in service function chains using backups with availability schedule," *IEEE Trans. Netw. Service Manag.*, vol. 18, no. 4, pp. 4294–4310, 2021.

[85] S. Herker, X. An, W. Kiess, S. Beker, and A. Kirstaedter, "Data-center architecture impacts on virtualized network functions service chain embedding with high availability requirements," in *2015 IEEE Globecom Workshops*, 2015, pp. 1–7.

[86] M. Pióro, M. Mycek, and A. Tomaszewski, "Network protection against node attacks based on probabilistic availability measures," *IEEE Trans. Netw. Service Manag.*, vol. 18, no. 3, pp. 2742–2763, 2021.

[87] T. Kimura, K. Ishibashi, T. Mori, H. Sawada, T. Toyono, K. Nishimatsu, A. Watanabe, A. Shimoda, and K. Shiomoto, "Spatio-temporal factorization of log data for understanding network events," in *2014 IEEE Conf. on Comput. Commun.*, 2014, pp. 610–618.

[88] D. Li, P. Hong, K. Xue, and J. Pei, "Availability aware vnf deployment in datacenter through shared redundancy and multi-tenancy," *IEEE Trans. Netw. Service Manag.*, vol. 16, no. 4, pp. 1651–1664, 2019.

[89] E. S. Kumar, E. M. Tufail, E. S. Majee, E. C. Captari, and S. Homma, "Service function chaining use cases in data centers," https://tools.ietf.org/html/draft-ietf-sfc-dc-use-cases-06, Feb. 2017.

[90] L. Deng, H. Zheng, X.-Y. Liu, X. Feng, and Z. D. Chen, "Network latency estimation with leverage sampling for personal devices: An adaptive tensor completion approach," *IEEE/ACM Trans. Netw.*, vol. 28, no. 6, pp. 2797–2808, 2020.

[91] W. Rankothge, F. Le, A. Russo, and J. Lobo, "Optimizing resource allocation for virtualized network functions in a cloud center using genetic algorithms," *IEEE Trans. Netw. Service Manag.*, vol. 14, no. 2, pp. 343–356, 2017.

[92] ——, "Experimental results on the use of genetic algorithms for scaling virtualized network functions," in *2015 IEEE Conf. on Netw. Function Virtualization and Software Defined Netw. (NFV-SDN)*, 2015, pp. 47–53.

[93] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu, "BCube: A high performance, server-centric network architecture for modular data centers," *SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 4, p. 63–74, Aug. 2009. [Online]. Available: https://doi.org/10.1145/1594977.1592577

[94] J. Zhao, D. Han, X. Jiang, T. Li, M. Zhang, and J. He, "Design and implementation of a topology discovery mechanism for bidirectional VLC networking," in *2020 12th Int. Symp. on Commun. Syst., Netw. and Digital Signal Process.*, 2020, pp. 1–6.

[95] Microsoft, "Azure AI notebooks for predictive maintenance," https://azuremlsampleexperiments.blob.core.windows.net/datasets/PdM_maint.csv, accessed Apr. 7, 2021.

[96] R. Kang, F. He, and E. Oki, "Robust virtual network function allocation in service function chains with uncertain availability schedule," *IEEE Trans. Netw. Service Manag.*, vol. 18, no. 3, pp. 2987–3005, 2021.

[97] H. Maleki and Y. Yang, "An uncertain programming model for preventive maintenance scheduling," *Grey Systems: Theory and Application*, 2017.

[98] W. Luo, T. E. Cheng, and M. Ji, "Single-machine scheduling with a variable maintenance activity," *Comput. Ind. Eng.*, vol. 79, pp. 168–174, 2015.

[99] D. Xu, L. Wan, A. Liu, and D.-L. Yang, "Single machine total completion time scheduling problem with workload-dependent maintenance duration," *Omega*, vol. 52, pp. 101–106, 2015.

[100] The Kubernetes Authors, "Scheduling framework," https://kubernetes.io/docs/concepts/scheduling-eviction/scheduling-framework/, accessed Dec. 20, 2022.

[101] J. R. Cheng and M. Gen, "Accelerating genetic algorithms with GPU computing: A selective overview," *Comput. Ind. Eng.*, vol. 128, pp. 514–525, Feb. 2019.

[102] R. Kang, F. He, and E. Oki, "Resilient resource allocation model in service function chains with diversity and redundancy," in *2021 IEEE Int. Conf. on Commun.* IEEE, 2021, pp. 1–6.

[103] ——, "Fault-tolerant resource allocation model for service function chains with joint diversity and redundancy," *Comput. Networks*, vol. 217, no. 9, Nov. 2022.

[104] M. Zhu, R. Kang, F. He, and E. Oki, "Implementation of backup resource management controller for reliable function allocation in kubernetes," in *2021 IEEE 7th Int. Conf. on Netw. Softwarization.* Tokyo, Japan: IEEE, Jun. 2021, pp. 360–362.

[105] ETSI, *Network Functions Virtualisation (NFV) Release 3; Reliability; Maintaining Service Availability and Continuity Upon Software Modification*, ETSI Std. GS NFV-REL 006, Rev. V3.1.1, Feb. 2018. [Online]. Available: https://www.etsi.org/deliver/etsi_gs/NFV-REL/001_099/006/03.01.01_60/gs_nfv-rel006v030101p.pdf

[106] T. Sato, F. He, E. Oki, T. Kurimoto, and S. Urushidani, "Experiment and availability analytical model of cloud computing system based on backup resource sharing and probabilistic protection guarantee," *IEEE Open J. of the Commun. Soc.*, vol. 1, pp. 700–712, 2020.

[107] D. Bertsimas and J. Tsitsiklis, "Simulated annealing," *Statistical Sci.*, vol. 8, no. 1, pp. 10–15, 1993.

[108] R. Kang, F. He, and E. Oki, "Resilient virtual network function allocation with diversity and fault tolerance considering dynamic requests," in *2022 IEEE/IFIP Netw. Oper. and Manag. Symp.* IEEE, 2022, pp. 1–9.

[109] L. Deng, G. Hinton, and B. Kingsbury, "New types of deep neural network learning for speech recognition and related applications: an overview," in *2013 IEEE Int. Conf. Acoust., Speech, Signal Process.*, 2013, pp. 8599–8603.

[110] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[111] J. Sola and J. Sevilla, "Importance of input data normalization for the application of neural networks to complex industrial problems," *IEEE Trans. Nucl. Sci.*, vol. 44, no. 3, pp. 1464–1468, 1997.

[112] Gurobi Optimization, LLC, "Gurobi Optimizer Reference Manual," https://www.gurobi.com, 2023, accessed July 20, 2023.

[113] R. Kang, M. Zhu, F. He, T. Sato, and E. Oki, "Design of scheduler plugins for reliable function allocation in kubernetes," in *2021 17th Int. Conf. on the Des. of Reliable Commun. Netw.* IEEE, 2021, pp. 1–3.

[114] R. Kang, M. Zhu, F. He, and E. Oki, "Implementation of virtual network function allocation with diversity and redundancy in kubernetes," in *2021 IFIP Netw. Conf.* IEEE, 2021, pp. 1–2.

[115] R. Kang, F. He, T. Sato, and E. Oki, "Demonstration of network service header based service function chain application with function allocation model," in *2020 IEEE/IFIP Netw. Oper. and Manage. Symp. (NOMS)*. IEEE, April 2020, pp. 1–2.

[116] R. Kang, M. Zhu, and E. Oki, "Implementation of service function chain deployment with allocation models in kubernetes," in *2022 IEEE Conf. on Comput. Commun. Workshops.* IEEE, 2022, pp. 1–2.

[117] The Kubernetes Authors, "kube-controller-manager | kubernetes," https://kubernetes.io/docs/reference/command-line-tools-reference/kube-controller-manager/, accessed Aug. 23, 2023.

[118] ——, "Deployments | Kubernetes," https://kubernetes.io/docs/concepts/workloads/controllers/deployment/, accessed Aug. 23, 2023.

[119] ——, "Jobs | Kubernetes," https://kubernetes.io/docs/concepts/workloads/controllers/job/, accessed Aug. 23, 2023.

[120] M. Zhu, F. He, and E. Oki, "Load balancing model under multiple failures with workload-dependent failure probability," in *Conf. Design of Reliable Commun. Netw.* IEEE, 2021.

[121] P. Quinn, U. Elzur, and C. Pignataro, "Network service header (NSH)," RFC 8300, Jan. 2018.

[122] Open Networking Foundation, "OpenFlow switch specification version 1.3.0 (wire protocol 0x04)," Jun. 2012.

[123] SFC working group, "SFC header mapping for legacy SF," https://www.ietf.org/archive/id/draft-song-sfc-legacy-sf-mapping-08.txt, accessed Dec. 21, 2022.

[124] John McDowall, "doonhammer/ovs at sfc.v30," https://github.com/doonhammer/ovs/tree/sfc.v30, accessed Dec. 21, 2022.

Publication List

Journal Papers

1. **R. Kang**, F. He, and E. Oki, "Fault-Tolerant Resource Allocation Model for Service Function Chains with Joint Diversity and Redundancy," *Comput. Networks*, vol. 217, no. 9, Nov. 2022.

2. **R. Kang**, F. He, and E. Oki, "Virtual Network Function Allocation in Service Function Chains Using Backups with Availability Schedule," *IEEE Trans. Netw. Service Manag.*, vol. 18, no. 4, pp. 4294 - 4310, Dec. 2021.

3. **R. Kang**, F. He, and E. Oki, "Robust Virtual Network Function Allocation in Service Function Chains with Uncertain Availability Schedule," *IEEE Trans. Netw. Service Manag.*, vol. 18, no. 3, pp. 2987-3005, Sep. 2021.

4. **R. Kang**, F. He, T. Sato, and E. Oki, "Virtual Network Function Allocation to Maximize Continuous Available Time of Service Function Chains with Availability Schedule," *IEEE Trans. Netw. Service Manag.*, vol. 18, no. 2, pp. 1556-1570, Jun. 2021.

International Conference Papers

1. **R. Kang**, M. Zhu, and E. Oki, "Implementation of Service Function Chain Deployment with Allocation Models in Kubernetes," *2022 IEEE Conf. on Comput. Commun. Workshops*, May 2022.

2. **R. Kang**, F. He, and E. Oki, "Resilient Virtual Network Function Allocation with Diversity and Fault Tolerance Considering Dynamic Requests," *2022 IEEE/IFIP Netw. Oper. and Manag. Symp.*, Apr. 2022.

3. M. Zhu, **R. Kang**, and E. Oki, "Implementation of Real-time Function Deployment with Resource Migration in Kubernetes," *2022 IEEE/IFIP Netw. Oper. and Manag. Symp.*, Apr. 2022.

4. **R. Kang**, M. Zhu, F. He, and E. Oki, "Implementation of Virtual Network Function Allocation with Diversity and Redundancy in Kubernetes," *2021 IFIP Netw. Conf.*, Jun. 2021.

5. M. Zhu, **R. Kang**, F. He, and E. Oki, "Implementation of Backup Resource Management Controller for Reliable Function Allocation in Kubernetes," *7th IEEE Int. Conf. on Netw. Softwarization*, Jun. 2021.

6. **R. Kang**, F. He, and E. Oki, "Resilient Resource Allocation model in Service Function Chains with Diversity and Redundancy," *2021 IEEE Int. Conf. on Commun.*, Jun. 2021.

7. **R. Kang**, M. Zhu, F. He, T. Sato, and E. Oki, "Design of Scheduler Plugins for Reliable Function Allocation in Kubernetes," *17th Int. Conf. on the Des. of Reliable Commun. Netw.*, Apr. 2021.

8. **R. Kang**, F. He, and E. Oki, "Optimal Virtual Network Function Placement in Chains Using Backups with Availability Schedule," *2020 IEEE 9th Int. Conf. on Cloud Netw.*, Nov. 2020.

9. **R. Kang**, F. He, T. Sato, and E. Oki, "Demonstration of Network Service Header Based Service Function Chain Application with Function Allocation Model," *2020 IEEE/IFIP Netw. Oper. and Manag. Symp.*, Apr. 2020.

10. **R. Kang**, F. He, T. Sato, and E. Oki, "Virtual Network Function Allocation to Maximize Continuous Available Time of Service Function Chains," *2019 IEEE 8th Int. Conf. on Cloud Netw.*, Nov. 2019.

Awards

1. Japan Society for the Promotion of Science Research Fellowships for Young Scientists (DC1) from 2021 to 2024.
2. Travel grant at ACM SIGCOMM in 2022.
3. IEEE ComSoc Student Grant at IEEE International Conference on Computer Communications (INFOCOM) in 2022.
4. IEEE ComSoc Student Grant at IEEE International Conference on Communications (ICC) in 2021.
5. IEEE ComSoc Student Grant at IEEE International Conference on Cloud Networking (CloudNet) in 2020.
6. Japanese Government (Monbukagakusho: MEXT) Scholarship in 2019.