Network Service Availability and Continuity Management in the Context of Network Function Virtualization

Siamak Azadiabad

A Thesis

In the Department

of

Computer Science and Software Engineering

Presented in Partial Fulfilment of the Requirements

for the Degree of

Doctor of Philosophy (Computer Science) at

Concordia University

Montreal, Quebec, Canada

September 2022

CONCORDIA UNIVERSITY

SCHOOL OF GRADUATE STUDIES

This is to certify that the thesis prepared

By: Siamak Azadiabad

Entitled: Network Service Availability and Continuity Management in the Context of Network Function Virtualization

and submitted in partial fulfillment of the requirements for the degree of

Doctor Of Philosophy (Computer Science)

complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

_____ Chair
Dr. Yong Zeng

_____ Thesis Supervisor
Dr. Ferhat Khendek

_____ Thesis Supervisor
Dr. Maria Toeroe

_____ Thesis Supervisor
Dr.

_____ Examiner
Dr. Roch Glitho

_____ Examiner
Dr. Juergen Rilling

_____ Examiner
Dr. Dhrubajyoti Goswami

_____ Examiner
Dr.

_____ External Examiner
Dr. Michel Dagenais

Approved by _____
Dr. Leila Kosseim , Graduate Program Director

November 16, 2022 _____
Dr. Mourad Debbabi , Dean
Gina Cody School of Engineering and Computer Science

Abstract

Network Service Availability and Continuity Management in the Context of Network Function Virtualization

Siamak Azadiabad, Ph.D.
Concordia University, 2022

In legacy computer systems, network functions (e.g., routers, firewalls, etc.) have been provided by specialized hardware appliances to realize Network Services (NS). In recent years, the rise of Network Function Virtualization (NFV) has changed how we realize NSs. With NFV, commercial off-the-shelf hardware and virtualization technologies are used to create Virtual Network Functions (VNF). In the context of NFV, an NS is realized by interconnecting VNFs using Virtual Links (VL).

Service availability and continuity are among the important non-functional characteristics of NSs. Availability is defined as the fraction of time the NS functionality is provided in a period. Current work on NS availability, in the NFV context, focuses on determining the appropriate number of redundant VNFs and their deployment in the virtualized environment, and the redundancy of network paths. Such solutions are necessary but insufficient because redundancy does not guarantee that the overall service outage time for an NS functionality remains below a certain threshold. Moreover, service disruption which impacts the service continuity is not addressed in the current work quantitatively. In addition, NSs and VNFs elasticity and the dynamicity of virtualized infrastructures which can impact the availability of NS functionalities, are not considered in the current state of the art.

In this thesis, we propose a framework for NS availability and continuity management, which consists of two approaches, one for design time and another for runtime adaptation. For this, we define *service disruption time* for an NS functionality as the amount of time for which the service data is lost due to service outages for a given period. We also define the *service data disruption* for an NS functionality as the maximum amount of data lost due to a service outage. The design-time approach includes analytical methods which take acceptable service disruption and availability requirements of the tenant, a designed NS, and a given infrastructure as inputs to adjust the NS design and map these requirements to constraints on low-level configuration parameters. Design-time approach guarantees the service availability and continuity requirements will be met as long as the availability characteristics of the infrastructure resources used by the NS constituents do not change at runtime. However, changes in the supporting infrastructure may happen at runtime due to multiple reasons like failover, upgrades, and aging. Therefore, we propose a runtime adaptation approach that reacts to changes at runtime and adjusts the configuration parameters accordingly to satisfy the same service availability and continuity requirements. The runtime approach uses machine learning models, which are created at design time, to determine the required adjustments at runtime.

To demonstrate the feasibility of the proposed solutions and to experiment with them, we present a proof of concept, including prototypes of our approaches and their application in a small NFV cloud environment created for validation purposes. We conduct multiple experiments for two case studies with different service availability and continuity requirements. The results from the conducted experiments show that our approaches can guarantee the fulfillment of the service availability and continuity requirements.

Acknowledgments

I would like to thank my supervisors, Dr. Ferhat Khendek and Dr. Maria Toeroe, who have helped me undertake this research. Without your guidance, support, and encouragement, I would not have been able to complete this journey.

I would like to extend my thanks to the examining committee for their advice and valuable input during the various stages of my Ph.D.

This research has been partially supported by the Natural Sciences and Engineering Research Council of Canada (NSERC), Ericsson, and Concordia University. I would like to thank them for their financial support and for providing me the opportunity to pursue my dream.

And my biggest thanks to my family for all the support they have shown me through the years. Above all, to my wife Zeynab for your love, support, understanding, and patience. Without you, I could not have come this far. And to my beloved daughter Aurora, for starting to shine in the sky of my life. We are waiting impatiently to welcome you into our family soon.

Table of Content

List of Figures

List of Tables

List of Abbreviations

ADT	Acceptable Down Time
AFR	Average Failure Rate
AM	Adaptation Module
ANN	Artificial Neural Network
ASDD	Acceptable Service Data Disruption
ASDT	Acceptable Service Disruption Time
BSS	Business Support System
BW	Bandwidth
CND	Checkpointing Network Delay
CpI	Checkpointing Interval
DB	Database
DL	Deep Learning
DNN	Deep Neural Network
EM	Element Manager
ETSI	European Telecommunications Standards Institute
FDT	Failure Detection Time
FoT	Failover Time
HI	Health-check Interval
HV	Hypervisor
HW	Hardware
IntVL	Internal VL
MANO	(NFV) Management and Orchestration
HMR	Health-check Monitoring Rate
ML	Machine Learning
MTBF	Mean Time Between Failures
MTTF	Mean Time To Failure
MTTR	Mean Time To Repair/Recover
NFP	Network Forwarding Path
NFV	Network Function Virtualization
NFVI	Network Function Virtualization Infrastructure
NFVO	Network Function Virtualization Orchestrator
NL	Network Latency
NS	Network Service
NSD	Network Service Descriptor
NsDF	Network Service Deployment Flavor
OSS	Operations Support System
OT	Outage Time
PC	Personal Computer
RA	Required Availability
PH	Physical Host
RT	Restart Time

SB	Standby
SDD	Service Data Disruption
SDT	Service Disruption Time
SRV	Server
TBCC	Time Between Consecutive Checkpoints
TBFLC	Time Between a Failure and the Latest committed Checkpoint
TDT	Total Down Time
ToT	Takeover Time
VIM	Virtualized Infrastructure Manager
VL	Virtual Link
VlEA	VL Expected Availability
VM	Virtual Machine
VNF	Virtual Network Function
VNFC	Virtual Network Function Component
VNFD	Virtual Network Function Descriptor
VnfDF	Virtual Network Function Deployment Flavor
VnfEA	Virtual Network Function Expected Availability
VNFFG	Virtual Network Function Forwarding Graph
VNFM	Virtual Network Function Manager

Chapter 1

Introduction

1.1 Context

In legacy computer systems, specialized hardware appliances have been used to provide network functions (e.g., routers, switches, firewalls, load balancers, …) and realize Network Services (NS). However, the rise of Network Function Virtualization (NFV) in recent years has changed the way we can realize and manage NSs by using general-purpose servers and virtualization technologies. With NFV, network functions are decoupled as software from hardware and can exploit virtualized resources on commercial off-the-shelf servers [1]. This way, NFV utilizes the advantages of virtualization and reduces capital and operational expenses, rapid service provisioning and deployment, scalability and elasticity, and multi-tenancy for NS [1].

The European Telecommunications Standards Institute (ETSI), a standardization organization in the Telecom industry, introduced a reference architecture for NFV [2]. An NS in the context of ETSI NFV is realized by interconnecting Virtual Network Functions (VNF) using Virtual Links (VL) [1]. The NFV framework manages the virtualization technologies to provide VNFs with virtual resources [1]. The VNF definition includes both historically network functions (e.g., firewalls, routers, etc.) and historically non-network functions, such as web servers and databases [3]. For

example, Figure 1-1 shows an NS realized by interconnecting three types of VNFs (i.e., firewall, load balancer, and web server) to provide a web service functionality. VNFs and VLs in this figure use the virtual computing, storage, and networking resources of the underlying infrastructure. In the NFV context, a VNF (or VL) instance is instantiated based on a VNF (or VL) type. Also, it is possible to have multiple instances of a VNF or a VL type in an NS. In Figure 1-1, there is one instance of the firewall VNF, one instance of the load balancer VNF, two instances of the web server VNF, and one instance of each VL. Note that a VL instance can interconnect more than two VNFs (e.g., VL3).

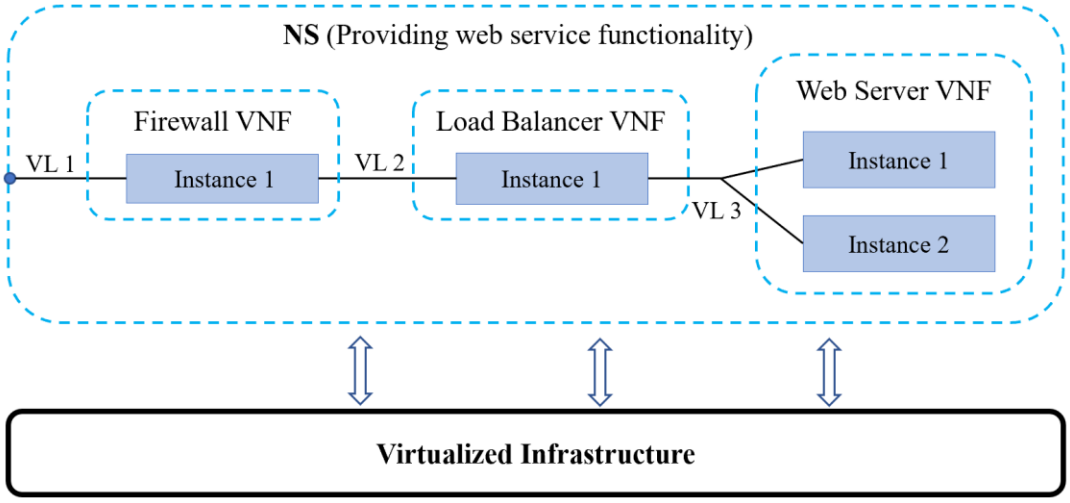


Figure 1-1: Example of NS in the context of ETSI NFV

1.2 Thesis motivations

Service availability for a system is defined as the fraction of time the system provides its service in a period [4]. It is usually expressed in terms of nines, e.g., 99.99%. Telecom NSs are expected to deliver highly available functionalities (i.e., 99.999% of availability) [5]. In addition, some use cases

of 5G require even more stringent service availability. For example, remote motion control requires ultra-high availability, which means the NS functionality should be available for at least 99.9999% of the time [6]. Thus, Telecom NSs need to be designed to meet their availability requirements, particularly when using general-purpose hardware, which is usually less reliable than specialized hardware [7]. Current work on NS availability in the NFV context only focuses on determining the appropriate number of redundant VNF instances, their placement in the virtualized environment, and their interconnections. Such solutions are necessary to fulfill the availability requirements of NSs. Still, they are insufficient since redundancy alone does not guarantee an acceptable overall service outage time for an NS functionality. Moreover, the scalability of NSs and VNFs introduces new challenges for NS availability in the NFV context since it can affect availability. This is not taken into consideration in existing solutions. For example, in the current state of the art, VNFs are considered monolithic applications. However, VNFs usually consists of internal components (VNFC) and VLs (IntVL) and may have (internal) scaling feature. Different placement policies can be applied to VNFCs which may result in different availability for the VNF. Therefore, the availability of a VNF may be different for the different number of redundant instances of VNFCs and IntVLs at different scaling levels of the VNF.

In addition, service continuity requirements are not addressed quantitatively in existing solutions, since there is no quantitative definition for service disruption in the literature. However, it is necessary to provide a quantitative definition so tenants can express the acceptable service disruptions, and NS designers/administrators can measure and guarantee it. According to the availability definition, the service is available as soon as a failed service is recovered. Therefore, in the case of a stateful service, whether a service is restarted from its initial state, or is recovered from a recently checkpointed state, the outage time (and the service availability) is the same. For example,

let us consider a video streaming NS. Assume while playing a movie, repeated failures happen every hour, and after each failure, the service is recovered in one second. If the service recovers from the last played frame before the failure, or if it recovers from the movie's beginning, the service outage time is one second per hour for both cases. Thus, the service availability is the same for both cases; however, the service disruption perceived by the end-user is different. From an end-user perspective, for the former case, the service disruption is one second per hour, and after each failure recovery, the user can continue watching the video from the last played frame eventually to its end. For the latter, the service is disrupted for one hour and one second due to each failure. Also, after the first failure, the video is restarted from the movie's beginning. As a result, the end-user will not ever finish watching the complete movie if the length of the movie is more than one hour. Therefore, when a failure happens for a stateful service, the service continuity (which is impacted by service disruption) depends also on the state recovery in addition to the service recovery.

To address all aforementioned challenges and aspects of the problem, a comprehensive solution is required to design NSs to fulfill service availability and continuity requirements. This design-time solution uses the availability characteristics of the infrastructure. For example, to find the required number of standby instances for a VNF, the availability of each VNF instance should be evaluated first, which partially depends on the availability of host types.

At runtime, the characteristics of the resources may change because of the dynamicity of virtualized infrastructure. For instance, a VNF can be migrated at runtime because of a failure and for the purpose of resource optimization, or after a failover, a standby VNF can become active using a different type of host with different availability, or the availability of resources changes over time because of aging. As a result, the number of standbys determined at design time for the VNF may

need to be increased at runtime to keep the VNF availability as required. Therefore, if an NS design satisfies certain service availability and continuity requirements at design time, it may not satisfy the same requirements at runtime if there are changes at the resource level. Thus, there is a need for a solution that reacts to these changes that affect the availability or continuity of NSs at runtime and adjust the NS configuration accordingly to guarantee that the requirements are always fulfilled throughout the lifetime of the NS.

The NS design/configuration created/refined by design-time and runtime solutions need to be cost-efficient in terms of resources the NS uses. For example, to fulfill service availability and continuity requirements, the minimum required redundancy should be imposed on VNFs to minimize the computing cost. Also, the VLs redundancy and network resource usage should be the minimum necessary. In other words, the design-time and runtime solutions should provide the optimal configuration which fulfills the service availability and continuity requirements while minimizing the resource costs.

The last challenge to be tackled for the service availability and continuity management of NSs is the inherent complexity of large NSs. NSs in the NFV environment can be large in scale and diversity of VNFs. Usually, the larger and the more complex an NS is, the more time-consuming is the design process to satisfy all the functional and non-functional requirements. For a large NS and several resource options (e.g., different networking or hosting options), finding a solution that not only satisfies the service availability and continuity requirements but also minimizes the resource cost is even more time-consuming. Therefore, the methods of both design-time and runtime solutions should have a tolerable time complexity to be applicable for large NSs.

1.3 Contributions of the Thesis

This thesis aims at devising a framework for NS availability and continuity management. Service availability already has a quantitative definition in existing work, but service disruption does not. This thesis starts by providing a quantitative definition for service disruption to enable tenants to express their service continuity (or acceptable service disruption) requirements and NS designers to measure and translate them to deployment configurations.

The main contributions of this thesis are as follows:

- **A design-time approach**: this approach takes tenant availability and acceptable service disruption requirements and a designed NS (which already satisfies the functional and performance expectations) as input and maps these requirements into low-level configuration parameters that affect the service availability and continuity and also adjusts the NS design. The deployment configuration generated by the design-time methods guarantees the fulfillment of the service availability and continuity requirements of the NS as long as, at runtime, the availability characteristics of the resources used by the NS constituents remain the same as estimated at design time. This approach also minimizes resource usage while determining the optimal configuration. In addition, if there are multiple hosting types to host VNFs and/or different networking options, this approach chooses the ones with lowest resource cost that can fulfill the service availability and continuity requirements. This solution required the following investigations and related contributions:
    - **Methods to calculate VNF instances availability and failure rate**: VNF instances availability and failure rate are needed for the design-time approach to evaluate the NS availability and service disruption. Generally, a VNF instance is composed of internal

components and VLs. In addition, a VNF instance can have internal scaling and redundancy for its components and internal VLs. In this thesis, we devise methods to calculate the guaranteed minimum availability and maximum failure rate of a VNF instance on a given infrastructure while taking into account the internal redundancy and scaling of the VNF instance and policies affecting the VNF components and internal VLs placement.

  - **A heuristic search to determine optimal configuration**: determining the optimal configuration while minimizing the cost of the resources in our design-time approach has exponential time complexity. Therefore, in this research, we propose a heuristic search that makes the approach affordable for large NSs. We show that the heuristic search result is satisfactory in terms of execution time and accuracy.

- **Runtime adaptation approach to maintain the satisfaction of the service availability and continuity requirements**: this approach reacts to the changes that can impact the fulfillment of the service availability and continuity requirements or the cost-efficiency of the configuration at runtime. This contribution consists of the following:
  - **A framework for runtime adaptation**: this framework introduces an Adaptation Module (AM) responsible for runtime adaptation management in the NFV architecture. The AM re-evaluates the availability and service disruption of the managed NSs when a change happens or if there was no change (including failures) for a long period, determines the new values for configuration parameters accordingly (if needed), and applies/requests the required reconfiguration (if needed). The proposed framework adjusts the NS if the availability or service continuity of the NS deteriorates (e.g., due to a change), or if the availability or service continuity of the NS is improved (e.g., if

there was no failure for a long period). For the latter, the adjustment reduces resource costs (e.g., reduce the number of standby instances) if possible, while maintaining the fulfillment of the service availability and continuity requirements.

- **A method to create Machine Learning (ML) models for runtime adaptation**: the AM can use the analytical methods of the design-time approach to determine the new values of the configuration parameters at runtime. However, the execution time, in this case, may not be affordable in certain situations (e.g., a large NS with stringent service availability and continuity requirements), even if the proposed heuristic search is used. Therefore, this thesis proposes a method to create ML models for NSs at design time. The ML models mimic the analytical methods, and the AM uses them at runtime to predict the required reconfiguration with constant time complexity.

The design-time and runtime approaches have been implemented and experimented with for validation purposes. We created an NFV cloud environment compatible with ETSI NFV reference architecture. We developed different types of VNFs to create two different NSs for our experiments. Our experiments show that:

- Optimal deployment configurations generated by the proposed design-time approach satisfy all different service availability and continuity requirements for both case studies.
- The AM can interact with the modules in the ETSI NFV reference architecture, and the runtime adaptation framework can reconfigure the NS.
- NSs adjusted using ML models satisfy service availability and continuity requirements for both case studies and different test runs when the failure rate of resources changes randomly at runtime.

1.4 Thesis Organization

The rest of the thesis is organized as follows: In Chapter 2, we provide the background knowledge, including ETSI NFV reference architecture and its different entities. Also, concepts about NS (in the context of ETSI NFV), service availability, service continuity, and ML used throughout this thesis are explained in this chapter. In Chapter 3, we review related work. Chapter 4 introduces our service disruption definition and discusses our proposed design-time approach for service availability and continuity requirements mapping into low-level configurable parameters. Chapter 5 presents the runtime adaptation approach, including the runtime adaptation framework and the ML model creation method. Chapter 6 discusses the testbed, case studies, prototypes developed to experiment with our solutions, and the results. In Chapter 7, we conclude this thesis.

Chapter 2

Background

In this chapter, we lay out the background knowledge for the thesis by providing an overview of ETSI NFV architecture, service availability and continuity, ML, and OpenStack and Tacker.

2.1 Network Function Virtualization Framework

ETSI NFV specification group introduced a reference architecture for NFV as depicted in Figure 2-1 [1].

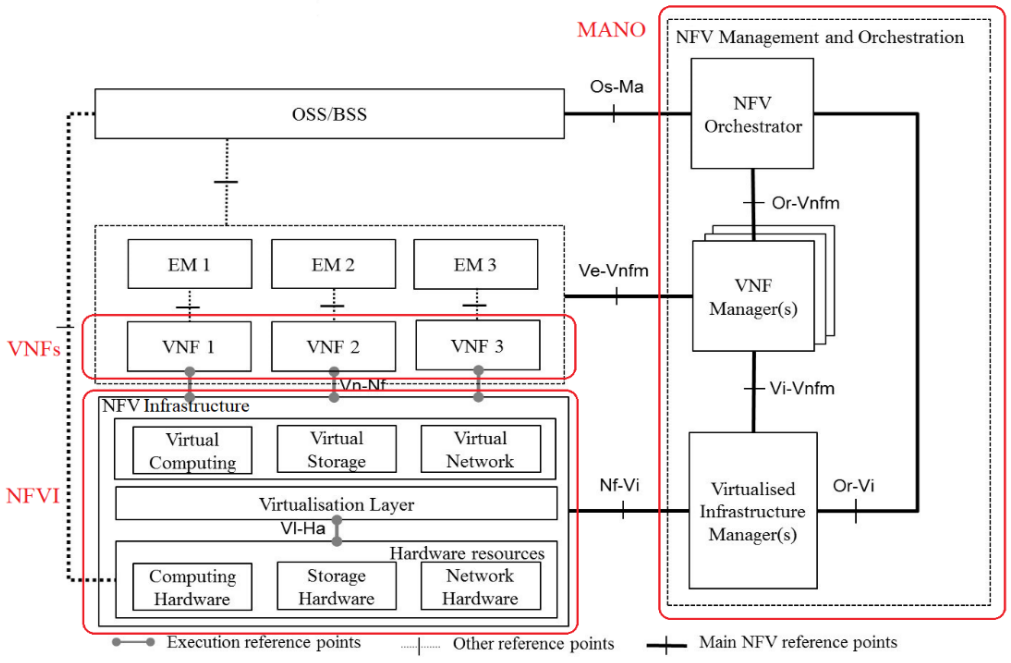


Figure 2-1: ETSI NFV reference architecture [1]

This architecture includes the following main sets of entities:

- NFV Management and Orchestration (MANO)
- NFV Infrastructure (NFVI)
- VNFs

Operations Support Systems/Business Support Systems (OSS/BSS) and Element Managers (EM) shown in Figure 2-1 are not part of the NFV architecture but interact with its entities.

2.1.1 NFV Management and Orchestration

MANO is responsible for providing NSs (i.e., their VNFs and VLs) with virtual resources and managing the lifecycle of NSs and VNFs [8]. MANO includes three functional blocks as described below.

2.1.1.1 NFV Orchestrator

NFV Orchestrator (NFVO) manages the lifecycle of NSs (e.g., instantiating, updating, and terminating of NSs) [8]. It instantiates an NS based on an NS Deployment Flavor (NsDF) [9], and each NsDF references an NS Descriptor (NSD) [9]. The NSD references the descriptors of the NS constituents, e.g., VNF Descriptors (VNFD) and VL Descriptors (VLD). NFVO receives descriptors and VNF images from the OSS and on-boards them [8]. In addition, NFVO keeps an inventory of allocated virtual resources to NSs and validates and authorizes the resource requests from VNF Managers (VNFM) [8].

2.1.1.2 VNF Manager

A VNF Manager (VNFM) is responsible for the lifecycle management of its managed VNF instances, including requesting NFVO to grant NFVI resources, and instantiating, scaling, monitoring, auto-healing, and terminating managed VNFs [8].

2.1.1.3 Virtualized Infrastructure Manager

Virtualized Infrastructure Manager (VIM) orchestrates the allocation, upgrade, release, and reclamation of NFVI resources (i.e., compute, storage, and network resources) [8]. VIM also collects performance and fault information of hardware, software, and virtual resources of the NFVI [8].

2.1.2 NFV Infrastructure

NFVI includes the hardware resources (i.e., general-purpose computing, networking, and storage hardware) at the bottom and the virtual resources at the top that can be assigned to VNFs and NSs. It also includes a virtualization layer (e.g., hypervisor) which supports the basic management of virtual resources (create, delete, or resize virtual resources) [1].

2.1.3 Virtual Network Function

A VNF is a software implementation of a network function that can run on the NFVI and consume virtual resources [1]. The VNFD of the VNF determines the VNF deployment and operational requirements [10]. A VNF profile specifies the instantiation information for a specific VNF Deployment Flavors (VnfDF) [9]. A VnfDF references a VNFD and indicates a particular deployment of the VNF [11]. VNFs instantiated based on different VnfDFs/VNF profiles of the same VNF may provide various functionalities and/or performance characteristics.

A VNF is composed of at least one VNFC and zero or more IntVLs [10, 12]. VNFCs are the actual consumers of infrastructure resources [10, 12]. A VNF vendor designs and structures the VNF with vendor specific VNFCs [12]. The VnfDF of a VNF defines the scaling levels of the deployment flavor, which indicate the number of instances for each VNFC and IntVL of the VNF at the different scaling levels [10]. The number of scaling levels is finite, and the number of instances for each scaling level may be indicated explicitly or by a *delta* [10]. In the latter case, the initial number of instances and the amount of change between consecutive scaling levels is indicated (i.e., *aspectDeltaDetails* attribute). The upper bound for the scaling steps is also set (i.e., *maxScaleLevel* attribute) to limit the number of scaling levels [10].

The VnfDF may also include anti-affinity rules for the instances of the same or different VNFCs [10]. Usually, anti-affinity rules are applied to instances of the same VNFC to place them on different physical hosts to prevent the impact of simultaneous failure of multiple/all instances due to physical host failure or upgrades [13]. Anti-affinity rules can also be specified for an IntVL to limit or prevent physical network sharing by IntVL instances. Every anti-affinity rule includes a property that determines the scope of the rule [10]. Possible scopes are *NFVI-PoP* (i.e., NFV Infrastructure Point of Presence), *ZoneGroup* (i.e., multiple zones grouped together), *Zone*, and *Node* (i.e., physical host) [10]. In this thesis, we only consider anti-affinity rules defined for VNFC/IntVL instances at the Node scope, i.e., distribution across multiple nodes inside a Zone.

2.1.4 Element Manager

In the context of ETSI NFV, MANO manages the virtualization aspect of NSs and VNFs, unaware of the functionality and application-level configuration of VNFs. However, an EM (not part of the ETSI NFV architecture) in interaction with the OSS can manage different aspects of the

application and the functionality of its managed VNFs, including fault, configuration, accounting, performance, and security [14].

2.1.5 Operation Support System/Business Support System

. In general, the OSS provides as input the descriptors of NSs and their constituents and is capable of requesting from the NFVO to onboard, instantiate, alter, or terminate NSs. The OSS can also manage VNF applications and their functionalities through EMs. The BSS includes billing and customer management systems to support business management [14].

2.2 Network Service

An NS in the context of NFV is a composition of VNFs and/or other NSs that are interconnected by VLs [11]. NSs are defined by their functional and non-functional characteristics. The deployment template of an NS is described by an NSD [9]. An NS is instantiated based on an NS deployment flavor (NsDF) [9]. Each NsDF references an NSD and specifies the deployment characteristics of the deployment flavor, like NS scaling levels [9]. The NsDF indicates a list of VNF and VL profiles used for instantiating an NS instance of the NsDF [9]. A VNF/VL profile specifies the instantiation information of a VNF/VL type [9]. Each NS scaling level indicates the number of VNF and VL instances for each VNF and VL profile of the NS. In the rest of this thesis, we refer to the VNF/VL profile as *VNF/VL*, and *VNF/VL instance* stands for an instance instantiated according to a VNF/VL profile.

The topology of an NS is described as a VNF Forwarding Graph (VNFFG) descriptor which references VNF [9]. An NS may have zero or more VNFFGs. A VNFFG contains one or more Network Forwarding Paths (NFP). An NFP defines an ordered list of connection points associated

with VNFs and VLs that form a sequence of network functions [9, 11]. Different NFPs may have some VNFs and/or VLs in common, while not all VNFs and/or VLs of the NS may be involved in every NFP.

2.3 Service Availability and Continuity

2.3.1 Service Availability

The availability is expressed as the fraction of time a system can deliver its service during a given period [4]. When a system fails, it is repaired in an average time known as Mean Time To Repair (MTTR). The average time between two consecutive failures is called Mean Time Between Failures (MTBF). If Mean Time To Failure (MTTF) denotes the average time that system is available after a repair up to the next failure, the MTBF is [4]:

$$MTBF = MTTF + MTTR \tag{1}$$

The relation between MTBF, MTTF, and MTTR in equation (1) is illustrated in Figure 2-2 [4]. In this figure, the service starts at T_{start} and fails at $T_{failure-1}$. Then, it's repaired at $T_{repaired-1}$, and fails again at $T_{failure-2}$.

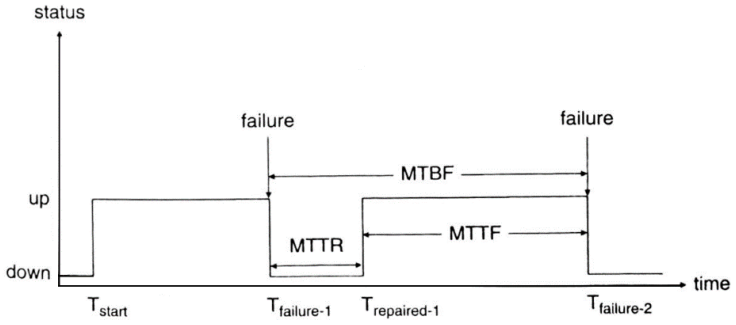


Figure 2-2: MTBF, MTTR, and MTTF [4]

According to the availability definition, the availability of a system is calculated by equation (2) [4].

$$Availability = \frac{MTTF}{MTBF} = \frac{MTTF}{MTTF + MTTR} \quad (2)$$

2.3.1.1 Fault Tolerance

According to equation (2), the availability of a system with an average failure rate in a period (i.e., in a given MTTF) can be adjusted by adjusting the MTTR. To repair the service of a software system, when a component of the system is failed, the failure should be detected first. Then, the failed component is repaired by restarting/reinitializing it. Therefore, MTTR depends on failure detection and repair time. Failure repair time is not usually adjustable since restarting a component takes the same average time. However, failure detection time can be tuned by changing the Health-check Monitoring Rate (HMR). The more frequent health-check messages are sent, the faster a failure is detected. However, increasing the HMR can burden the component with the execution of the monitoring logic and consequently decrease its performance [15]. Therefore, software components usually have a boundary for the maximum HMR. If the availability of a system with the maximum HMR needs to be improved, *fault tolerance* is a known technique that can be employed [4]. Fault tolerance relies on recovery instead of repair. To recover the failure of a component of a software system, once the failure is detected, another (standby or active) component can take over the responsibility of the failed component. Usually, failover time is less than the restart time. Therefore, the availability of a system can be improved.

2.3.1.2 Redundancy

Redundancy is the key strategy of fault tolerance. There are five different redundancy models for a component as follows [4]:

- No-redundancy: no standby instance of the component with the state information protects active instances (spares may be present).
- 2N redundancy (a.k.a., 1+1 redundancy and hot-standby redundancy): each active instance of the component is protected by one standby instance with synchronized state information.
- N+M redundancy: N active instances of the component are protected by M standby instances. Standby instances also have the state information. This model is more resource-efficient than the 2N redundancy model since, usually, N is greater than M.
- N-Way redundancy: a redundant instance may provide a service as an active instance while it is standby to protect other services.
- N-Way-Active redundancy (a.k.a., active-active redundancy): this is the load balancing redundancy model. In this model, there is no standby instance assigned to active instances. This redundancy model better suits for stateless applications.

2.3.2 Service Continuity

In addition to service availability, service continuity is another important characteristic of Telecom NSs [16]. Service continuity is defined as providing uninterrupted/low-interrupted service [17]. Thus, to improve the service continuity of a system, one way is to increase its availability (i.e., reduce the service outage time). In addition, for a stateful service (e.g., a video streaming service), the service continuity also depends on the amount of service data loss (i.e., service data disruption) when a failure happens. Checkpointing a system state is a technique that improves the state preservation after a failure repair/recovery, improving service continuity [18, 19].

2.4 Machine Learning

ML is an artificial intelligence technique to enable computer systems to learn from experiences [20]. Such a system can be used for decision making, behavior modeling, pattern recognition, etc. Generally, there are three different ML approaches [21]: *supervised*, *unsupervised*, and *reinforcement* learning. Each approach is appropriate to solve certain types of problems.

2.4.1 Supervised Machine Learning

The machine is given a training dataset for supervised ML to learn a model from [21]. Each record of the dataset has two parts: data or input and label or output. The training dataset is also called labeled data. A model is a function that predicts an output for a given input [21]. After learning the model from the training dataset, the machine can map any given input (which may not be previously present in the training dataset) to an output. Supervised ML models can solve classification problems [21]. For example, an ML model can classify patients into recovered and unrecovered groups based on their symptoms. In this case, the machine is called a *classifier*. The output of a classifier is discrete with few known values (i.e., classes). Also, supervised ML models can solve regression problems in which the output has a numerical value with natural ordering [21]. In this case, the machine is called a *regressor*. For example, a regressor can predict a land price based on its area, location, and neighborhood conditions. The output value for this example can (virtually) be any real number.

2.4.1.1 Artificial Neural Network and Deep Learning

Artificial Neural Network (ANN) is one of the most used and powerful algorithms to solve classification and regression problems. An ANN is composed of multiple layers, including input,

hidden, and output. Each layer has one or more nodes/neurons. The number of nodes for the input layer is equal to the features of the training data [22]. A node in the input layer sends through vectors the corresponding feature value to some or all the nodes in the hidden layer. Each vector applies a weight to the feature value [23]. The number of nodes for the hidden layer depends on the complexity of the model the ANN should be trained for [22]. A node in the hidden layer applies a function to the input values it receives through weighted vectors from the input layer nodes [23]. Similarly, the hidden layer nodes send their outputs to nodes in the output layer. The number of nodes for the output layer in the case of a classifier ANN is equal to the classes of the output [22]. In the case of a regressor ANN, the number of nodes for the output layer is equal to the number of different types of predictions the machine is needed to make. When an ANN is trained, the weight of each vector is adjusted so that for any given input, the ANN can predict an output with an acceptable accuracy [21]. Figure 2-3 shows an example of an ANN architecture with ten nodes in the input layer, eight in the hidden layer, and four in the output layer [23].

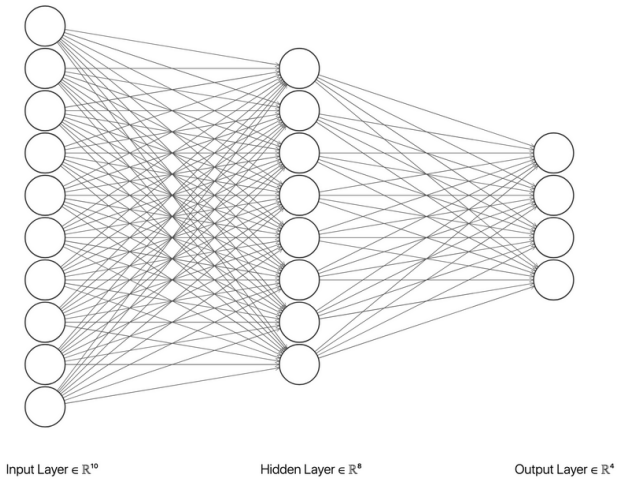


Figure 2-3: Example of an ANN architecture [23]

An ANN with one hidden layer is also called a shallow ANN [23]. To solve problems with higher level of complexity, we can create ANN models with more than one hidden layer. An ANN with more than four hidden layers is usually called a Deep Neural Network (DNN) [23]. Also, ML with a DNN is called Deep Learning (DL) [23]. Figure 2-4 shows an example of a DNN architecture with six hidden layers [23]. Note that the number of nodes for different hidden layers can be different.

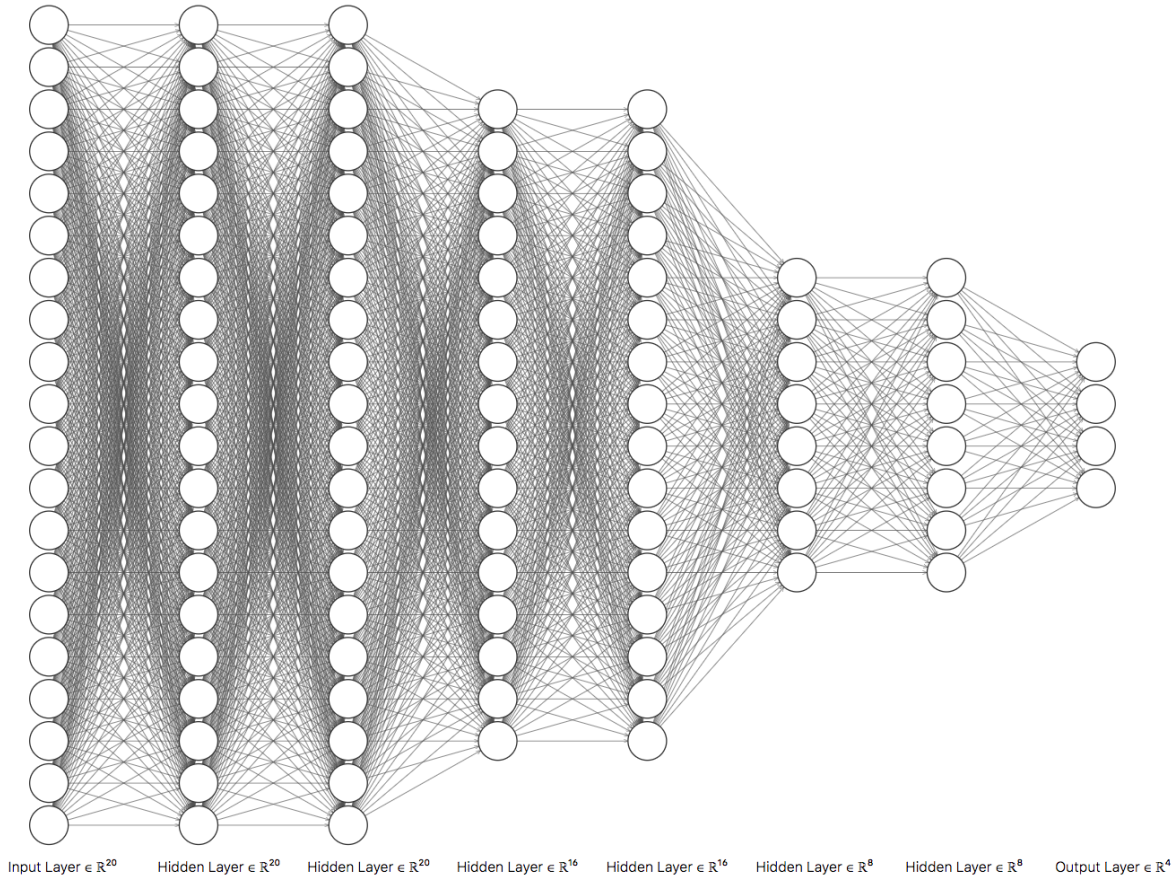


Figure 2-4: Example of a DNN architecture [23]

The following steps are necessary to create a DL model [24]:

1. Data collection
2. Data Analysis
3. Model construction
4. Model validation

Training data can be collected from a real system or generated synthetically and then labeled [25]. Data analysis includes selecting the data features that have the most effect on the target outputs and preprocessing the data [24]. Scaling and normalization are the steps of data preprocessing. Model construction includes selecting the hyper-parameters value and training the model [24]. The number of hidden layers and nodes are examples of hyper-parameters. Once the model is created, we should validate its accuracy. We can use some sample data (used or unused during the training phase) to compare the model predictions with the labels of the sample data. If the result is unsatisfactory, we may go to the first step and try to increase the model's accuracy by adding more training data, selecting better features, and/or tuning hyper-parameters.

2.4.2 Unsupervised Machine Learning

Unsupervised learning does not use labeled data. Instead, it tries to find patterns/structures inside a given set of data [21]. For example, unsupervised learning can categorize a collection of books into different groups based on their genres [21]. In this case, the machine is not trained to classify the books. Instead, it tries to find similarities between books and put similar ones into the same group. Two main unsupervised ML techniques are *clustering* and *Dimensionality reduction* [21]. The previous example of book categorization is a clustering technique. Dimensionality

reduction is used to alter data to provide a simpler view by eliminating the redundant or unimportant features of the data [21]. For instance, this technique can be used for data visualization to reduce the data dimension [21].

2.4.3 Reinforcement Machine Learning

A reinforcement machine learns how to act/react in a dynamic environment to achieve a goal by receiving feedback(s) for its actions from the environment [21]. Usually, a reinforcement machine learns in a real/near-real environment, and it takes time to learn models compared with the two other approaches. For example, a reinforcement machine can master playing chess to beat even top players. The machine only needs to know the basic chess rules in the beginning. Then, it plays and learns from the feedback received for its moves/strategies.

2.5 OpenStack and Tacker

OpenStack [26] and Tacker [27] can be used to create an NFV cloud environment compatible with the ETSI NFV architecture. OpenStack is a set of open-source cloud management software to manage and control large pools of computing, networking, and storage resources to create and orchestrate virtual machines and their interconnectivity [26]. Therefore, it can manage the NFVI and play the role of a VIM for the MANO. Tacker is a software module that adds the VNFM and the NFVO functionalities to the OpenStack controller.

Chapter 3

Related Work

In this chapter, we review existing work related to the approaches proposed in this thesis, i.e., the design-time approach and runtime adaptation framework. We also review the ETSI NFV specifications related to the reliability of VNFs/NSs. Accordingly, we organize this section into three sub-sections, two for literature review and one for ETS NFV specifications review.

3.1 Network Service Design to Fulfill Availability Requirements

In this thesis, we are interested in the service availability and continuity management of NSs in the context of NFV. However, in general, VNFs/NSs can be considered component-based/distributed systems. Therefore, in this sub-section, we start by the review of the related work in the general context of distributed systems.

3.1.1 Service Availability of Distributed Systems

Service availability has been expansively investigated for component-based/distributed systems, and it has a quantitative definition which makes it a measurable characteristic [4, 28, 29, 30]. However, related work in distributed systems focuses on component/node redundancy and protection. In the related work, parameters like failure detection and recovery times and elasticity are not addressed or they are poorly taken into account.

Fault tolerance is the key mechanism proposed/investigated in related work to improve the availability of distributed systems [31, 32, 33, 34, 35, 36, 37]. Also, different redundancy mechanisms (e.g., active-standby, active-active, etc.) for the components of distributed systems are elaborated and compared in related work in terms of their benefits and applications [31, 32, 33, 34]. But no comprehensive solution is proposed to guarantee a certain availability (i.e., a maximum affordable overall outage time in a period) which depends on the value of configuration parameters like HMR [31, 32, 33, 34, 35, 36, 37].

Authors in [38] and [39] consider adjusting parameters like timers to guarantee a certain availability. However, the solutions proposed in these articles are based on using a specific availability manager framework. In addition, parameters like the elasticity of the distributed system are not taken into consideration. The work in [38] proposes a method for the automatic generation of configurations for component-based applications to deploy on top of an availability management framework that manages the availability of the services provided by the applications. This method takes as input the configuration requirements, the deployment infrastructure, and the software description provided by the software vendor. The configuration requirements include the "workload" to be protected and the redundancy model. The method automatically selects the software components to provide the required services and organizes them according to the required redundancy model to protect these services as required. A configuration is generated automatically. The required availability is not expressed explicitly in [38] but only implicitly through the "workload" and the redundancy model. The work in [39] improves the work in [38]. It takes a required availability as input and improves the generation method with multiple configuration design patterns and availability estimate methods. The configuration patterns aim at targeting better configurations to meet the required availability, like the pattern for selecting an appropriate

redundancy model or the pattern for setting the most suitable recovery mechanism for a component. The availability estimate methods aim at evaluating the availability as early as possible in the generation process and eliminating configuration options that cannot meet the required availability. Using the configuration patterns, the approach presented in this paper sets some configuration attributes to limit the impact of a failure and therefore minimize service outage. For instance, it may set the recommended recovery attribute of a component to failover instead of restart. The approach in [39] does not determine the values for low-level configuration parameters like timers. These parameters are set according to the vendors' recommendations and are used in the availability estimate methods as defined for the middleware.

3.1.2 Network Service Availability and Continuity in the Context of NFV

Related work on designing/redesigning NSs to fulfill an availability requirement mainly focuses on protecting the VNF functionality, i.e., having enough standby instances for VNFs. There is no existing comprehensive solution in state of the art to guarantee the service outage, which may also depend on configuration parameters like HMR.

The authors in [40] propose an algorithm to determine the required number of standby instances for the least reliable VNFs of the NSs and reduce the computing resource consumption simultaneously. They show that solving this problem is NP-complete and propose a heuristic algorithm as a solution. This solution is improved in [41] by taking infrastructure availability into consideration. The work in [41] aims at minimizing the computing cost and reduces this cost by half compared to [40]. In [42], the authors take resource constraints into account. They propose two algorithms to guarantee that there is sufficient VNF redundancy to support the required availability of the NS. This paper suggests protecting only the key VNFs for more efficient resource utilization.

The distribution problem of VNF replicas is addressed in [43]. This paper approaches the NS availability problem by determining the number of required replicas for the VNFs and their placement on the existing physical nodes. However, the work does not consider the availability of computing nodes at the infrastructure layer. It only covers the application-level availability. In [44], the authors take the geo-redundancy into consideration for VNFs placement and propose a solution to find enough redundancy for VNFs while respecting hardware capacity and bandwidth constraints for their placement. The goal of the works in [40, 41, 42, 43, 44] is to provide enough standby instances for a VNF so that the probability of having healthy VNF instances is equal to or above the expected availability from the VNF. However, configuration parameters like failover/recovery time are not taken into consideration in these works.

The authors of [45] partially address the recovery time in their solution for NS availability. However, this solution does not guarantee the total service outage time for an NS satisfying an acceptable threshold. It relies on the microservice paradigm and benefits from the redundancy mechanisms available in microservice-based architectures. This work proposes to apply the 1+1 redundancy to all VNFs and calculates the networking overhead imposed by the availability mechanisms. However, assuming the 1+1 redundancy for all VNFs may not be resource-efficient and/or enough in all cases to guarantee the expected availability from the VNF. The work proposes a method of calculating the outage time during failover. All the calculations and proposed architectures in this article are application specific, and most of them cannot be generalized and used for other applications.

VNFs are building blocks of NSs, and the availability of NSs depends on the availability and failure rate of VNFs. In the literature, the availability and failure rate of VNF instances are assumed

to be known/given to design NSs, since a VNF instance is considered to have only one VNFC placed on a single host. However, this is not typically the case. A VNF may consist of more than one VNFC and zero or more IntVLs. It may also have multiple scaling levels and different affinity/anti-affinity policies for the placement of its VNFC and IntVL instances. Thus, the availability and failure rate of a VNF instance depend (partially) on its components' placement and redundancy during runtime. These are not taken into consideration in state of the art.

A set of related work address the effect of VNF redundancy and/or VNF instances placement on VNF/NS availability [41, 43, 44, 46, 47, 48, 49, 50]. However, VNF instances are not considered as (potentially) distributed applications. Article [41] proposes a method to improve the reliability of NSs and minimize the resource cost by protecting the most important VNFs with redundancy mechanisms instead of spending resources on the least reliable VNFs protection. It also suggests most reliable hosts should be selected for the most important VNFs placement to optimize the overall resource consumption. In [43], the authors tackle the problem of redundant VNFs distribution and their placement on physical nodes to create independent redundant network paths (i.e., a chain of VNFs and VLs) at the physical level to prevent single point of failures. Article [44] focuses on VNF redundancy calculation while optimizing VNF licensing cost, server utilization, and network latency. The authors in [46] aim at the problem of NS availability fulfillment by utilizing VNF redundancy and minimizing the resource cost at the same time. They propose a solution to calculate the number of required standby instances for a VNF based on the availability of each VNF instance. The solution in [47] proposes an algorithm to calculate the redundancy for VNFs with lower availability in the NS and minimize networking costs. This article also proposes an algorithm to determine the network path redundancy to improve the NS reliability. Paper [48] proposes an algorithm for the placement of VNFs of replicated network service chains to meet a required availability. [49] proposes a solution

for VNF redundancy and placement to increase NS resiliency against single-node failures and minimize network latency simultaneously. Authors in [50] address the effect of placement of router VNFs on the availability of the NS and propose a framework to create different reliable routes for enterprise data centers. This framework determines the required redundancy for (router) VNFs and their placement to increase the NS resiliency to failures and meet the required networking delay. Neither of the abovementioned articles addresses the internal components of VNFs and the impact of VNFCs and IntVLs placement policies on the availability of VNF instances since they assume each VNF is a single entity placed on a single host. Also, the VNF elasticity (i.e., VNF internal scaling) at runtime is not taken into consideration. In addition, the number of VNF instances is assumed to remain the same during runtime (i.e., NS-level scaling is not addressed).

We are not aware of any related work that quantifies service continuity. Despite the lack of a quantitative definition for service continuity in related work, state checkpointing is a well-known technique used in the literature to improve service continuity. It is used with or without redundancy [4, 51, 52, 53, 54]. However, without defining and quantifying service disruption, existing solutions cannot measure and/or provide guarantees for service continuity.

3.2 Runtime Monitoring and Adaptation of Network Services

3.2.1 Monitoring Resource Changes at Runtime

One essential requirement to enable the runtime adaptation of NSs to maintain the fulfillment of their required availability is monitoring the changes in resources assigned to the NS constituents. Some related work has proposed a framework/solution to monitor VNFs/NSs, in the context of NFV, for availability and/or performance purposes. However, some do not monitor all type of resources

and all required kinds of changes, and/or others are not fully compliant with ETST NFV reference architecture. A fault management architecture and procedures are proposed for NFV-based mobile networks in [55]. This specification proposes a procedure to notify the OSS about virtual resource failures through the VIM, VNFM, and EM. The only changes monitored by this framework are resource failures that affect VNFs. Other types of changes, including changes that affect VLs, are not considered in this specification. In [56], the authors propose a framework to monitor resources, aggregate events, notify the NFVO and the OSS about alerts, and provide a visual dashboard for 5G networks. They propose to insert a monitoring layer between the infrastructure (i.e., NFVI) and the orchestration layers. Their definition of the orchestration layer combines both the NFVO and the OSS in the same layer which is not compliant with the current ETSI NFV specifications. The proposed monitoring framework in [56] provides some advanced functionalities like visualization and aggregation. However, where the proposed layer entities are placed in the NFV reference architecture and how they interact with other NFV entities is not defined. In [57], the authors propose a monitoring framework that adds features like anomaly detection and aggregation optimization to the existing monitoring solutions of the VIM. The goal is to reduce the number of notifications sent to the upper level, especially for large systems. Although the detailed capabilities of this solution are not discussed in [57], it can be a potential candidate for adding availability constraints monitoring capability to the VIM if it is considered and implemented for the NFV reference architecture. This solution does not cover VNF application failures. Authors in [58] introduce the need for end-to-end QoS monitoring and propose a top-down approach to add to the monitoring systems of the NFV. Using the top-down approach, the OSS can monitor the end-to-end QoS of the NS and send alerts to the NFVO when a failure happens. The NFVO alerts the VNFM and/or the VIM, and they take healing actions if needed. The abovementioned monitoring solutions do not cover all changes needed

to be monitored to evaluate the availability of NSs at runtime to adjust the NS accordingly. For example, these articles do not address changes like NS scaling, resource upgrades, and VLs characteristics.

3.2.2 Runtime Adaptation of Network Services

To the best of our knowledge, there is no related work in the context of the work in this thesis addressing the runtime adaptation of NSs to fulfill service availability and continuity requirements. However, a set of related work addresses the adaptation of NSs to meet performance requirements in case of degradation at runtime. The MANO itself supports VNF and NS level scaling at runtime to adjust the number of instances according to the current workload or if it is asked for [10, 9]. The goal of [59] is to evaluate the end-to-end network delay at runtime by an analytical model and predict the optimal required resources by a reinforcement machine learning model to reduce the delay to an acceptable threshold. The work in [60] proposes an architecture for the dynamic provisioning of QoS-oriented VNFs for IoT systems. This architecture proposes an entity to manage NFV, software-defined networks, and IoT middleware together. The proposed entity monitors the resource consumption and the servers' performance and orders the NFVO and/or the SDN controller to take the necessary adaptation actions to keep the QoS of the IoT services as expected. Authors in [61] address the problem of auto-scaling during runtime to meet the required performance of the NS in a resource-efficient manner. They propose a DL solution (using DNN) to create a classifier to predict the appropriate scaling level at runtime when there is a change in the traffic load. Each class of the classifier represents a valid scaling level. To train the DNN, they generate a set of training data, i.e., corresponding labels are generated for a random set of input data. In [62], an ML approach is proposed to determine the optimal number of VNF instances at runtime to fulfill the current

workload. This work benefits from supervised ML and proposes a method to generate a dataset to train the machine. In addition, the approach in [62] determines the optimal placement for VNF instances to meet the required performance in a cost-efficient manner. Solutions in [59, 60, 61, 62] are specific to performance degradation compensation at runtime. They do not handle the case of availability constraints violation which is the goal of the solution proposed in this thesis.

3.3 ETSI NFV Specifications and Reports

ETSI NFV has published a series of specifications/reports, known as the REL series, which address the reliability and availability of VNFs/NSs and the MANO. NFV-REL 001 lists the resiliency requirements of VNFs, defines service availability levels, and introduces a fault correlation and recovery architecture [5]. NFV-REL 002 is a report which introduces a rollback recovery architecture to support checkpointing at the virtualization level (i.e., virtual machine checkpointing) [63]. NFV-REL 003 is another report which introduces the different protection schemes for VNFs like active-active or active-standby redundancy models [7]. NFV-REL 004 is a report on an active monitoring and failure detection framework for NFV architecture [64]. This report proposes a framework and procedure for a potential monitoring agent interaction with NFV components. Another specification is NFV-REL 006, which provides procedures to maintain the availability of VNF functionality during upgrades [65]. NFV-REL 007 is a report which defines the responsibilities of MANO functional blocks to support the requirements introduced in NFV-REL 001 and indicates the appropriate protection scheme (i.e., active-active or active-standby) for the MANO components to improve the MANO availability [66]. Another REL publication is NFV-REL 010 [67]. This report covers the NFV resiliency concerns for network slice design. Also, it provides the calculations to find the overall availability of redundant VNFs. The other REL publication is

NFV-REL 011 [68]. This report introduces multiple use cases for the MANO software modification while preserving service availability and continuity. The most recently published REL report is NFV-REL 012 [69]. It introduces some use cases for monitoring and recovery actions for the MANO failures and overload.

The functional requirements of MANO, including the support for service availability at the resource level, are introduced in NFV-IFA 010 specification [70]. According to this specification, an availability requirement may be assigned to an NS, and the MANO should assign appropriate virtualized resources to the NS constituents to meet the availability expectation. NFV-IFA 010 has defined the responsibility of each functional block of the MANO to support service availability.

The NFV framework manages the virtualization aspect of VNFs and NSs, and it is unaware of their functional/application-level characteristics. Therefore, all the abovementioned ETSI NFV specifications and reports address the availability only at the resource level (e.g., hosting availability). Configuration parameters like HMR at the VNF application level are not addressed in these specifications/reports.

Chapter 4

Design-time Approach

In this chapter, first, we provide quantitative definitions for service disruption and explain the problem we aim to solve. Then, we present our design-time approach.

4.1 Service Disruption Definitions

As discussed earlier in this thesis, service availability is an important characteristic of NSs, defined as the fraction of a period that the service is provided [4]. Tenants can express the Required Availability (RA) in terms of nines for NS functionalities (i.e., at the NS service level). For example, six nines of RA (i.e., 99.9999%) for an NS functionality means that the overall outage time of the NS functionality in a year is required not to be more than 31.5 seconds.

Service continuity is also another important characteristic of Telecom NSs. Service continuity depends on service availability. However, for stateful services, service continuity also depends on the service disruption caused by failures. As mentioned in Chapter 3, there is no quantitative definition for service disruption in the current work. Therefore, to enable tenants to express their acceptable service disruption requirements and NS designers to measure them, we propose the following definitions:

- *Service Disruption Time (SDT)*: we define the SDT for an NS functionality as the amount of time for which the service state is lost due to all service outages in a period.

Tenants can express their Acceptable SDT (ASDT) for an NS functionality in terms of seconds (e.g., 31.5 seconds per year, equal to 0.000001% of a year).

- *Service Data Disruption (SDD)*: we define the SDD for an NS functionality as the maximum amount of data lost due to one failure. In other words, it is the maximum service data lost during the Time Between a Failure and the Latest committed Checkpoint (TBFLC). Tenants can express the Acceptable SDD (ASDD) for an NS functionality in terms of bits (e.g., 1024 b per failure).

4.2 Problem Statement and Analysis

4.2.1 Problem Statement

The design-time approach provides a solution that enhances an NS design by mapping service-level availability and continuity (or acceptable disruption) requirements (i.e., RA, ASDT, and/or ASDD) to constraint on low-level configuration parameters of the NS and calculate the number of required standby instances for each VNF and VL so that the requirements can be met. This solution also minimizes the cost of the resources while guaranteeing the fulfillment of the service availability and continuity requirements.

4.2.2 Assumptions

As introduced in Chapter 2, VNFFG indicates the given topology of an, and it contains one or more NFPs. If the NS provides more than one functionality, different availability and/or acceptable service disruption requirements may be requested for the different functionalities. We assume that each NS functionality is provided through a specific NFP of the NS.

We also assume availability is always part of the tenant requirements for each NS functionality. Therefore, a tenant may ask for one of the following three kinds of requirements:

- RA,
- RA together with an ASDD per failure, or
- ASDT for a given period.

It is noteworthy that since the ASDT includes the outage time, the RA is also implied.

In this thesis, VNF/VL failure refers to the simultaneous failure of all active instances of the VNF/VL (profile), causing an outage for the VNF functionality unless an instance failure is explicitly mentioned. The outage of a VNF functionality means all serving (active) instances of the VNF are terminated because of failure, or they are disconnected due to a VL failure (i.e., failure of all active instances of the VL). An outage of a VNF functionality can cause a service outage for the NS functionality in which the VNF is involved. In this thesis, we do not consider the case when only some active instances of a VNF/VL fail together, that is, when the corresponding NS functionality encounters a *service degradation*. Similarly, a VNF instance is assumed to be failed when all (redundant) instances of at least one of its VNFCs or IntVLs fail simultaneously. At the VNF functionality level, all the other cases are considered service degradation.

4.2.3 Problem Analysis

4.2.3.1 Impacting Factors

NS Scaling may change the number of VNF and/or VL instances and, as a result, alter the availability of the VNF functionality and/or the VL. Thus, the NS scalability needs to be considered

to meet the RA, the ASDT, and/or the ASDD of an NS functionality. In other words, all scaling levels of the NsDF should meet the requirements.

To meet an RA for an NS functionality, each VNF/VL in each NFP should satisfy a certain availability. I.e., for each VNF/VL, its instance(s) together should satisfy this Expected Availability (VnfEA/VlEA). Therefore, based on the requested RA of the NS functionality, an NS designer can first determine the VnfEA applicable to each VNF functionality. Then, based on the availability of a VNF instance, if the availability provided for the VNF functionality by the VNF instance(s) does not satisfy the VnfEA, providing additional redundant VNF instances is a technique suggested by [4, 7]. For example, for a stateful VNF, to protect a functionality for which N active VNF instances are needed to serve the workload, an appropriate number of standby instances can guarantee that the probability of having at least N healthy active instances at any given moment is equal to or greater than the VnfEA. Similarly, if the availability of VL does not satisfy the VlEA, increasing the redundancy of VL instances can improve the availability of the VL.

The required number of standby instances and the overall availability of the redundant VNF instances are calculated using the availability of the VNF instances [44, 46]. Also, the failure rate of a VNF (with redundant instances) depends on the failure rate of its instances. A VNF instance may consist of multiple VNFCs and IntVLs and can have internal scaling. The number of instances of VNFCs and/or IntVLs changes when the VNF instance scales in/out. Therefore, the availability and failure rate of the composite VNF instance may also change. In addition, the degree of sharing of physical hosts affects the availability and failure rate of VNFC instances since the availability of a VNFC instance is derived from the availability of the VNFC application and the availability of the underlying infrastructure. In turn, this affects the availability and the failure rate of the functionality

provided by the VNF instance. Thus, we need a solution to determine the availability and failure rate of a VNF instance, which takes into account the impact of internal redundancy, elasticity, and placement policies of the constituents of the VNF instance.

An adequate redundancy for a VNF cannot guarantee by itself that the availability of the VNF functionality is equal to or greater than the VnfEA. To ensure a VnfEA, the overall outage time of the VNF functionality should be kept below the acceptable outage time. For example, for a VNF functionality provided by one active instance, if this instance fails, the failure detection and recovery times determine the resulting outage time of the VNF functionality for one failure. Therefore, if the detection and the recovery times together, for all the failures of a VNF with a known failure rate, are longer than the acceptable outage time, the availability of the VNF functionality is less than the VnfEA even if there are enough standby instances.

The availability metrics such as failure detection and recovery times of VNFs depend on and are impacted by configuration parameters such as the HMR and the Failover Time (FoT). Whenever a failover mechanism protects a functionality of a VNF, the setting of the failover configuration parameters affects the availability of the VNF functionality and, consequently, affects the availability of the corresponding NS functionality. So, to meet the RA for an NS functionality, in addition to redundancy, it is also important to determine the appropriate values for these parameters, provided they are configurable. For example, let us assume an NS functionality provided by one stateful VNF with two instances. Figure 4-1 depicts this example. The service data rate of the NS functionality is 60 Mbps. The VNF of this NS uses 1+1 redundancy to protect its VNF functionality. To detect the failure of VNF instances, their health is monitored by health-check messages. Let also assume that the active instance checkpoints its state periodically to an external Database (DB) to

enable the redundant instance to recover the VNF functionality from the last stored state. The state of the active instance changes continuously. The active instance checkpoints its state every 80 ms, and a health-check message is sent every 40 ms. In the beginning, VNF_{ins1} is active and VNF_{ins2} is standby. VNF_{ins1} fails at 130. So, the next health-check message is not sent at 160 and VNF_{ins2} starts preparing to become active. It takes 20 ms for VNF_{ins2} to recover the state from the DB and become active. Thus, the service outage (i.e., the outage of the NS functionality) is 50 ms. However, the state recovered was checkpointed at 80, resulting in an SDT of 100 ms. In addition, the SDD, that is, the service data lost during TBFLC is 3 Mb (i.e., 50 ms * 60 Mbps).

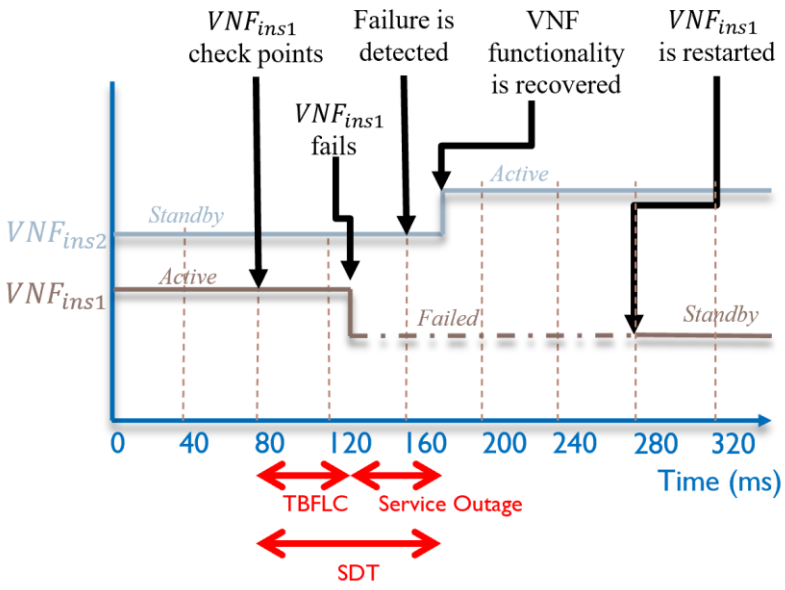


Figure 4-1: Example of outage time and service disruption for an NS

As shown in this example:

- HMR affects the service outage (and the service availability)
- HMR and TBFLC affect the SDT (and the service continuity)

- Service data rate and TBFLC affect the SDD (and the service continuity)

Therefore, if ASDT or ASDD is requested for an NS functionality, it is necessary to determine the TBFLC of the VNFs. TBFLC depends on the checkpointing method and the Checkpointing Interval (CpI) [71, 72] if the CpI is configurable for the VNF.

The TBFLC also depends on the network delay since the network delay affects the transmission time of the checkpoint data from the active VNF instance to the DB (or the peer VNF instance). We call this delay the Checkpointing Network Delay (CND). In addition, we assume that the average checkpoint preparation and commitment times are known for each VNF functionality.

4.2.3.2 Cost-efficiency and Configuration Limitations

The faster we detect failures, the faster we can react to them to reduce the overall outage time and the SDT. We can detect failures faster if we increase the HMR. If configurable, decreasing the CND and/or CpI can reduce the SDT and SDD. For example, if there are multiple networks to choose from, by selecting a network that provides lower CND, i.e., a network with lower latency and/or higher bandwidth, we can improve the TBFLC. However, increasing the HMR can burden the VNF with executing the monitoring logic and consequently decrease its performance [15]. In addition, higher HMR imposes higher networking overhead as well. Selecting a lower value for a configurable CpI to have more frequent state checkpointing has similar effects [71]. The need for guaranteeing a certain performance of the VNF instance puts constraints on the maximum acceptable HMR and, for a configurable CpI, the minimum CpI value. Thus, the NS designer must choose the HMR and CpI configuration values within these boundaries. In addition, increasing the HMR, decreasing the CpI, and selecting a network option with less CND increase the networking cost if we define the

networking cost based on the networking overhead and/or the network speed. So, there is a trade-off between improving service availability, SDT, or SDD, and the networking cost. All these factors should be considered for an appropriate set of configuration values.

To meet the service availability and continuity requirements, we may need to introduce redundancy to ensure that when a VNF fails, there is a VNF instance to failover to. Having more redundant instances (or standbys) can improve the protection of the VNF functionality. However, at the same time, they increase the cost of computing resources. Thus, there is another trade-off between improving the protection of the VNF functionality and the computing cost, which needs to be considered for a potential solution.

4.2.4 Formal Definition of the Problem

For each NS functionality, if TDT_{NFP} denotes the total downtime of the NS functionality in a period of t, SDD_{NFP} denotes the service data disruption of the NS functionality due to a failure, SDT_{NFP} denotes the service disruption time of the NS functionality in a period of t, $C_N(NFP)$ denotes the networking cost of the NFP, and $C_C(NsDF)$ denotes the computing cost of the whole NS, then:

- If the requirement for the NFP is RA, the proposed solution determines the optimal configuration values which satisfy $TDT_{NFP} \leq t * (1 - RA)$ for all NS scaling levels and minimize $C_N(NFP)$.
- If the requirement for the NFP is RA and ASDD, the proposed solution determines the optimal configuration values which satisfy $TDT_{NFP} \leq t * (1 - RA)$ and $SDD_{NFP} \leq ASDD$ for all NS scaling levels and minimize $C_N(NFP)$.

- If the requirement for the NFP is ASDT, the proposed solution determines the optimal configuration values which satisfy $SDT_{NFP} \leq ASDT$ for all scaling levels and minimize $C_N(NFP)$.

In addition, for all these three cases, our solution determines the required number of standby instances for all VNFs and VLs at all scaling levels of the NS and minimizes $C_C(NsDF)$.

4.3 Overall View of the Design-time Approach

The design-time approach aims to map an NS's availability and continuity requirements to configuration parameters and determine the required redundancy for VNFs and VLs to adjust the input NsDF. Figure 4-2 shows the overall picture of the design-time approach. This approach includes methods to calculate:

- Availability and failure rate of VNF instances
- Availability, outage time, and service disruption of VNFs
- Availability and redundancy of VLs
- Service outage, service disruption, and resource cost of NFPs

These methods feed the main method of the approach, as shown in Figure 4-2, which maps the RA, ASDT, and/or ASDD requirements to low-level configuration parameters and generates the outputs shown in this figure.

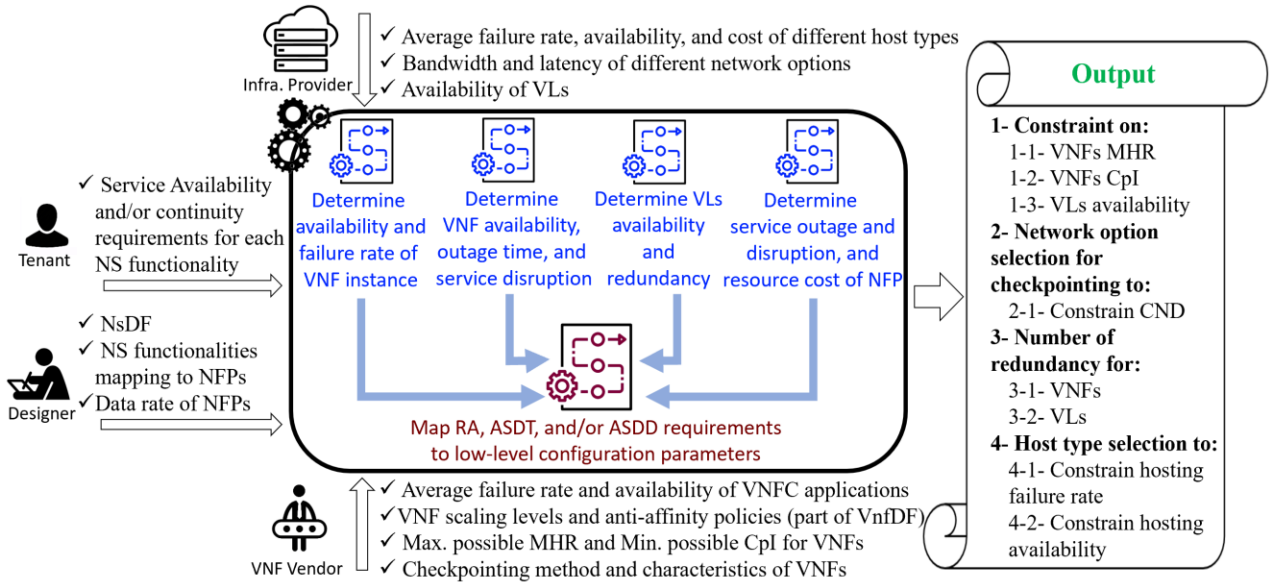


Figure 4-2: Overall picture of the design-time approach

An NS design may start by creating an NsDF that provides the requested functionalities but may or may not meet all the non-functional requirements as proposed in [73]. Our approach takes as input an NsDF meeting the functional and some non-functional requirements like capacity. But it may not satisfy the tenant's availability and service continuity requirements. The correspondence between NS functionalities and NFPs is another input required for our approach if the tenant requests different requirements for different NS functionalities. In addition, if ASDD is requested for an NS functionality, the maximum data rate of the corresponding NFP is needed.

Another set of input information of the approach is the infrastructure characteristics. The approach needs the knowledge of availability and failure rate of hosts since the availability characteristics of a host affect the availability of the guest VNFCs. Moreover, the cost of each hosting type is needed to find the optimal configuration in terms of the computing cost. In addition, the different networking options and their latency and bandwidth should be known since the CND

depends on the network latency between the sender (i.e., active VNF) and receiver (i.e., a DB or a peer VNF). Finally, the maximum possible availability of VLs that the infrastructure provider can provide (i.e., the NFVI provider) needed to be given.

The approach also requires some information about VNFs. This includes the availability and failure rate of VNFC applications, VNF scaling levels and anti-affinity policies (part of the VnfDF), the upper boundary for HMR of VNFs, the lower boundary for CpI of VNFs, and the checkpointing method and checkpointing characteristics (e.g., the average size of a checkpoint message) that each VNF supports. The VNF vendor can usually provide all this information.

To satisfy an RA or ASDT, the design-time approach determines constraints on VNFs health-check rate and VLs availability, indicates the required number of redundancies for VNFs and VLs, and selects the hosting type for each VNF to constrain the hosting availability and failure rate. To satisfy an ASDT or ASDD, methods of this approach determine the constraint on the VNFs checkpointing interval and select the networking option to constrain the checkpointing network delay.

In the following sub-sections of this chapter, we describe the proposed methods in detail.

4.4 VNF Instance Availability and Failure Rate

4.4.1 VNF Instance Availability

A VNF instance is available if at least one instance of each VNFC and IntVL is available. Therefore, if A_{VNFC} denotes the availability of a VNFC (i.e., at least one instance of the VNFC is

available) and A_{IntVL} denotes the availability of an IntVL (i.e., at least one instance of the IntVL is available), the availability of a VNF instance (A_{vnf}) with n VNFCs and m IntVLs is:

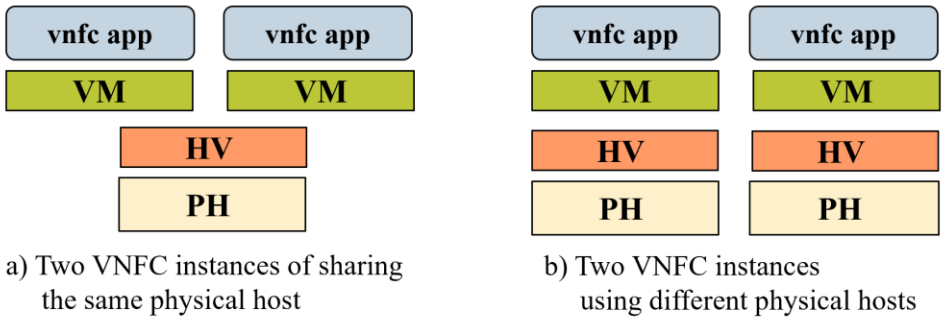
$$A_{vnf} = \Pi_{i=1}^{n} A_{VNFC_i} * \Pi_{j=1}^{m} A_{IntVL_j} \quad (3)$$

The A_{VNFC} and the A_{IntVL} depend on the number of their instances and their placement, which are different at the different scaling levels. Therefore, we first discuss how these factors affect the availability of the VNFC and the IntVL. Then, we propose a method of calculating the availability of VNF instances.

4.4.1.1 Availability of VNFCs

The A_{VNFC} is derived from the availability of the VNFC instances (A_{vnfc}). The A_{vnfc} depends on the availability of the application ($A_{vnfc-app}$) and the availability of the underlying infrastructure. With respect to the latter, if we assume one layer of virtualization (i.e., a bare-metal hypervisor), the A_{vnfc} depends on the availability of the virtual machine (A_{VM}), the availability of the underlying hypervisor (A_{HV}), and the availability of the physical host (A_{PH}).

The A_{VNFC} also depends on the placement of its instances. For example, assuming a VNFC with two instances, we have two cases of placement: in one, the VNFC instances share the same physical host (Figure 4-3-a); in the other, each VNFC instance is placed on a different physical host (Figure 4-3-b).



a) Two VNFC instances of sharing the same physical host

b) Two VNFC instances using different physical hosts

Figure 4-3: Placement of two instances of a VNFC

The availability of the VNFC for this example in case of Figure 4-3-a is:

$$A_{VNFC} = A_{PH} * A_{HV} * (1 - (1 - A_{VM} * A_{vnfc-app})^2) \quad (4)$$

In other words, the VNFC is available if the physical host (PH) and the hypervisor (HV) are both available and at least one of the VNFC application (vnfc app) instances and its corresponding virtual machine (VM) are available. In equation (4), we assume that the VMs are of the same type/flavor and have the same availability.

When each VNFC instance is placed on a different PH (Figure 4-3-b), the VNFC is available if at least one of the VNFC application instances and its corresponding VM, HV, and PH are available. Therefore, A_{VNFC} will be:

$$A_{VNFC} = 1 - (1 - A_{PH} * A_{HV} * A_{VM} * A_{vnfc-app})^2 \quad (5)$$

In equation (5), we assume that the PHs, HVs, and VMs are of the same type/flavor and have the same availability.

Thus, to calculate the availability of the VNFCs, we need to consider the placement of its instances, which is guided by the applicable anti-affinity rules. If no anti-affinity rule is applied to a VNFC, in the worst case, all the instances can be placed on the same host. The availability of a VNFC with n instances in the worst-case is calculated according to equation (6) which is a generalization of equation (4):

$$A_{VNFC} = A_{PH} * A_{HV} * (1 - (1 - A_{VM} * A_{vnfc-App})^{n}) \quad (6)$$

In the case of an anti-affinity rule, in the best case, all VNFC instances are placed on different hosts. In this case, the availability of the VNFC with n instances is a generalization of equation (5), as shown in equation (7):

$$A_{VNFC} = 1 - (1 - A_{PH} * A_{HV} * A_{VM} * A_{vnfc-App})^{n} \quad (7)$$

There are also cases where some (more than one but not all) instances of a VNFC share the same host. An anti-affinity rule may allow a degree of host sharing (i.e., d). In addition, there may be a boundary (i.e., b) for the number of collocated VMs on a host due to the capacity limitations of hosts, even if there is no anti-affinity rule. To cover these different cases, we can define k to denote the effective constraint for the number of VNFC instances that can share a host. The value of k for the different cases is:

$$k = \begin{cases} n, \text{if (no } anti\_affinity \text{ rule) and } (b \geq n) \\ b, \text{if (no } anti\_affinity \text{ rule) and } (b < n) \\ 1, \text{if (an } anti\_affinity \text{ rule) and } (d \text{ is not present}) \\ b, \text{if (an } anti\_affinity \text{ rule) and } (d \geq b) \text{ and } (b < n) \\ n, \text{if (an } anti\_affinity \text{ rule) and } (d \geq b) \text{ and } (b \geq n) \\ d, \text{if (an } anti\_affinity \text{ rule) and } (d < b) \text{ and } (d < n) \\ n, \text{if (an } anti\_affinity \text{ rule) and } (d < b) \text{ and } (d \geq n) \end{cases} \quad (8)$$

In the worst case, the number of VNFC instances that can share the same host is k. Therefore, we will have $g = \lceil n/k \rceil$ groups of VNFC instances where each group is placed on a separate host. If we assume that the instances are distributed evenly (e.g., in a round-robin manner) among g hosts, $n - g * \lfloor n/g \rfloor$ groups will have $\lceil n/g \rceil$ instances and $g * (1 + \lfloor n/g \rfloor) - n$ groups will have $\lfloor n/g \rfloor$ instances. Therefore, the availability of the VNFC is:

$$A_{VNFC} = 1 - \left(1 - A_{PH} * A_{HV} * \left(1 - \left(1 - A_{VM} * A_{vnfc-App}\right)^{\lceil n/g \rceil}\right)\right)^{n - g * \lfloor n/g \rfloor} * \left(1 - A_{PH} * A_{HV} * \left(1 - \left(1 - A_{VM} * A_{vnfc-App}\right)^{\lfloor n/g \rfloor}\right)\right)^{g * (1 + \lfloor n/g \rfloor) - n} \quad (9)$$

In other words, the VNFC is available if at least one VNFC instance (i.e., VNFC application and the VM) of one group and its corresponding HV and PH are available. Equation (9) supports all cases of k of equation (8) and generalizes Equations (6) and (7) as well (i.e., $k = n$ and $k = 1$).

4.4.1.2 Availability of Internal VLs

Like VNFCs, an IntVL may have more than one instance. The redundant instances may be used for load balancing or high availability. Redundant instances of an IntVL connect to the VNFC instances using the same virtual IP address. The virtual IP address's functionality (i.e., load balancing or high availability) determines the usage of the connected IntVL instances [10]. Whether the redundant instances of an IntVL are used for load balancing or high availability, the IntVL is assumed to fail if all active instances fail simultaneously. A VnfDF may include anti-affinity rules for IntVLs [10]. Assuming that an IntVL instance is an overlay network on top of the physical network (no nested overlay networks), the availability of one IntVL instance is equal to the

availability of the network elements (i.e., physical links and nodes) that create the virtual link [74]. If no anti-affinity rule is applied to an IntVL, in the worst case, the availability of its instances may not be independent. In fact, in the worst case, all instances of an IntVL can share partially or completely the same physical network, and when one IntVL instance fails due to a failure at the physical network layer, all instances may fail. Thus, in the worst case, where all redundant instances of an IntVL share the same physical network, the availability of the IntVL (A_{IntVL}) is equal to the availability of one of its instances (A_{intvl}).

$$A_{IntVL} = A_{intvl} \tag{10}$$

If an anti-affinity rule is defined for an IntVL, the failures of its different instances are independent since they do not share the same physical network. Therefore, for an IntVL with n (redundant) instances, we have:

$$A_{IntVL} = 1 - (1 - A_{intvl})^n \tag{11}$$

4.4.1.3 A Method for Calculating the Availability of a VNF Instance

Equations (9) and (11) show that the availability of a VNF instance varies with the number of instances, thus, according to the VNF scaling levels. Therefore, as the availability of a VNF instance, we need to consider the lowest availability ($A_{vnf-min}$) among all scaling levels, which then can be guaranteed. We propose the following method to determine the availability of a given VNF instance deployed on a given infrastructure.

Method 1: VNF instance availability calculation method

Step 1: Set $A_{vnf-min} = 1$

Step 2: For each VNF scaling level, perform steps 2-1 to 2-4

Step 2-1: Calculate the availability of each VNFC for this scaling level, using equation (9)

Step 2-2: Calculate the availability of each IntVL for this scaling level

- If no anti-affinity rule applies, use equation (10)
- Otherwise, use equation (11)

Step 2-3: Calculate the availability of the VNF instance (A_{vnf}), using equation (3).

Step 2-4: If A_{vnf} for this scaling level is lower than $A_{vnf-min}$, set $A_{vnf-min} = A_{vnf}$

In Method 1, the final value of $A_{vnf-min}$ represents the guaranteed minimum availability of the VNF instance.

4.4.2 VNF Instance Failure Rate

A VNF instance fails if all (redundant) instances of at least one of its VNFCs or IntVLs have failed at the same time. Therefore, the failure rate of a VNF instance (λ_{vnf}) with n VNFCs and m IntVLs is the summation of the failure rates of its VNFCs (λ_{VNFC}) and IntVLs (λ_{IntVL}), as shown in equation (10).

$$\lambda_{vnf} = \sum_{i=1}^{n} \lambda_{VNFC_i} + \sum_{j=1}^{m} \lambda_{IntVL_j} \tag{12}$$

Like the availability of VNFCs and IntVLs, the λ_{VNFC} and λ_{IntVL} for each VNFC and IntVL depend on the number of their instances at different scaling levels and their placement as determined by their anti-affinity rules. Thus, we explore the factors that affect the VNFC and IntVL failure rates. Then, we propose a method to calculate the guaranteed maximum for the failure rate of a VNF instance considering all scaling levels.

4.4.2.1 *Failure Rate of VNFCs*

The failure rate of a VNFC depends on the failure rate of its instances, which in turn depends on the failure rates of the VNFC application ($\lambda_{vnfc-app}$), the VM (λ_{VM}), the underlying hypervisor (λ_{HV}), and the physical host (λ_{PH}). The placement of VNFC instances also affects the failure rate of the VNFC. For example, let us consider the two cases presented in Figure 4-3 and assume that the failure rate of the VNFC application instances, the VMs, and the hypervisors is zero. Therefore, the failure rate of the VNFC depends only on the failure rate of the physical host(s). So, when both instances are placed on the same host (Figure 4-3-a), the VNFC fails if the physical host fails. However, when the two instances are hosted on different physical hosts (Figure 4-3-b), the VNFC fails only if both physical hosts fail simultaneously – which is less likely than the former case.

As mentioned earlier, if the instances of a VNFC are not anti-affine, in the worst case, all of them may share the same host. Also, there may be a boundary (i.e., b) for the number of collocated VMs on a host due to the capacity limitations of the hosts. In the case of anti-affinity, the instances do not share physical hosts, or the degree of host sharing is constrained (i.e., d).

To calculate the failure rate of a VNFC based on the failure rate of the VNFC application, VM, HV, and PH layers, and the placement of instances, we can use the relationship between reliability (R) and failure rate (λ). The reliability of a system (with exponential distribution function) is defined as the probability of having a failure-free service during a given period [75]. The reliability of a system (R_{Sys}) for a period of t is calculated using equation (13) [76].

$$R_{Sys} = \mathrm{e}^{-\lambda_{sys} * t} \tag{13}$$

So, we can first calculate the reliability of a VNFC based on the reliability of its components at the different layers and then calculate the failure rate of the VNFC using its reliability.

Fault Tree Diagrams (FTD) can be used to calculate the reliability of a software system with multiple components [77]. Different algorithms/tools have been proposed in the literature to analyze FTDs [77]. An FTD can be constructed from two basic gates, which connect input events (i.e., failures) to output events [77]:

- **AND gate**: output event happens if the input events occur together
- **OR gate**: output event happens if at least one input event occurs

For example, Figure 4-4-a depicts a system with two components (A and B) that fails if both components fail simultaneously. Figure 4-4-b depicts a system that fails if any of its two components fail. In this picture, F_A denotes the failure event of component A, and F_B denotes the failure event of component B.

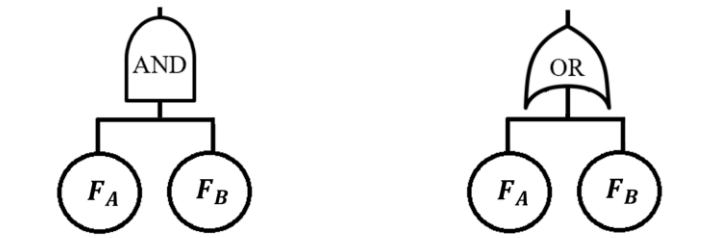


Figure 4-4: FTD of a system with two components

The reliability of a system with the FTD of Figure 4-4-a is calculated using the following equation [76]:

$$R_{Sys} = R_A + R_B - R_A * R_B \tag{14}$$

If $n > 2$ events are connected to an AND gate, we can restructure the FTD, using $n - 1$ number of AND gates and calculate the reliability of two components/subsystems with each gate. For example, if a system has three components (A, B, and C), it fails if all components fail at the same time, as depicted in Figure 4-5-a. This FTD can be reconstructed with two gates, as shown in Figure 4-5-b. If we assume that the reliability of these three components is the same ($R_{cmp} = R_A = R_B = R_C$), applying equation (14) to the FTD of Figure 4-5-b, the reliability of the system will be:

$$R_{Sys} = 3R_{cmp} - 3R_{cmp}^2 + R_{cmp}^3 \tag{15}$$

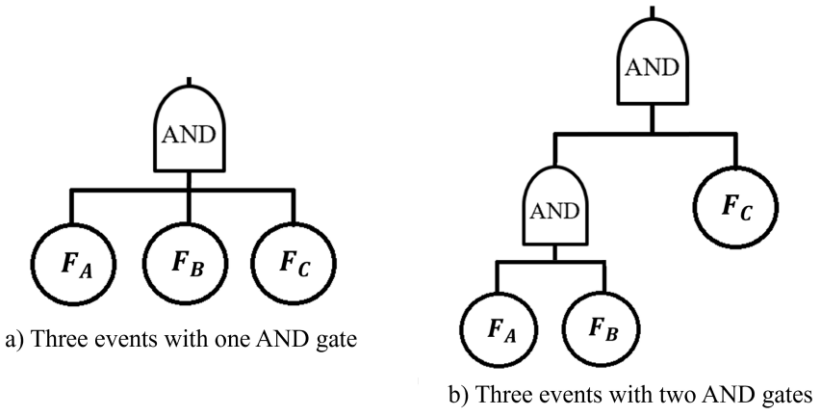


Figure 4-5: FTD of a system with three components which fails if all fail at the same time

Equation (15) can be rewritten as equation (16) and generalized to equation (17) for a system with n components that fails if all components fail at the same time.

$$R_{Sys} = \binom{3}{1} R_{cmp}^1 - \binom{3}{2} R_{cmp}^2 + \binom{3}{3} R_{cmp}^3 \tag{16}$$

$$R_{Sys} = \sum_{i=1}^{n} (-1)^{i-1} * \binom{n}{i} * R_{cmp}^{i} \quad (17)$$

For a system with FTD of Figure 4-4-b, the reliability can be calculated using the following equation [76]:

$$R_{Sys} = R_A * R_B \quad (18)$$

If $n > 2$ events are connected to an OR gate, we can reconstruct the FTD, using $n - 1$ number of OR gates and calculate the reliability of two components/subsystems with each gate. For example, if a system with three components fails, if at least one component fails, as depicted in Figure 4-6-a, the FTD can be reconstructed with two OR gates, as shown in Figure 4-6-b.

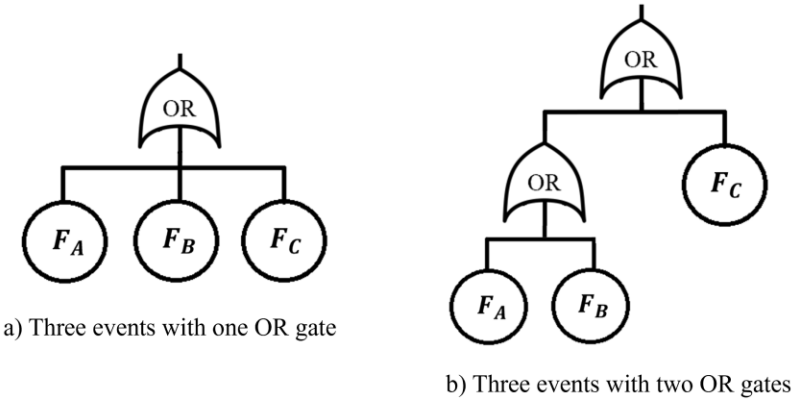


Figure 4-6: FTD of a system with three components that fails if at least one of them fails

Applying equation (18) to the FTD of Figure 4-6-b, the reliability of the system will be:

$$R_{Sys} = (R_A * R_B) * R_C \quad (19)$$

To generalize equation (19), if a system with n components fails due to one component failure, the system reliability is the product of the reliability of its components, as shown in equation (20):

$$R_{Sys} = \Pi_{i=1}^{n} R_{cmp_i} \tag{20}$$

The FTD of the example in Figure 4-3-b is shown in Figure 4-7.

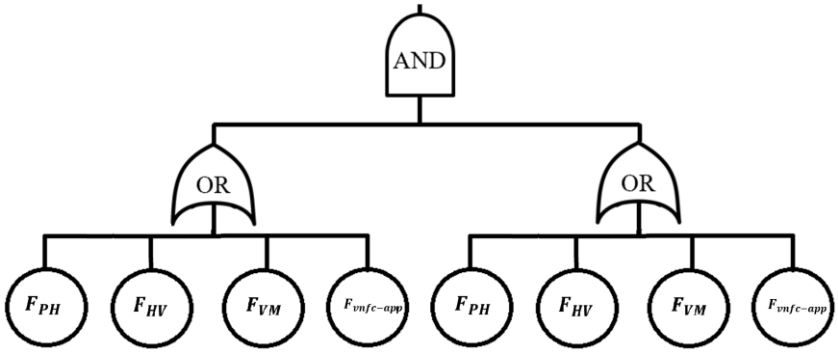


Figure 4-7: FDT of the VNFC in Figure 4-3-b

Using equations (17) and (20), the reliability of this VNFC is:

$$R_{VNFC} = \sum_{i=1}^{2} (-1)^{i-1} * \binom{2}{i} * \left(R_{PH} * R_{HV} * R_{VM} * R_{vnfc-app}\right)^{i} \tag{21}$$

In general, for a VNFC with n anti-affine instances without any degree of host sharing, the reliability will be:

$$R_{VNFC} = \sum_{i=1}^{n} (-1)^{i-1} * \binom{n}{i} * \left(R_{PH} * R_{HV} * R_{VM} * R_{vnfc-app}\right)^{i} \tag{22}$$

The FTD of the VNFC presented in Figure 4-3-a is shown in Figure 4-8. Using equations (17) and (20), the reliability of this VNFC is:

$$R_{VNFC} = R_{PH} * R_{HV} * \sum_{i=1}^{2} (-1)^{i-1} * \binom{2}{i} * \left(R_{VM} * R_{vnfc-App}\right)^{i} \tag{23}$$

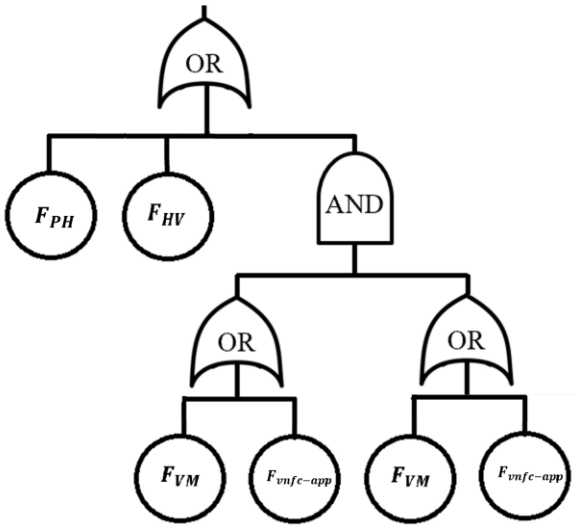


Figure 4-8: FDT of the VNFC in Figure 4-3-a

Therefore, for a VNFC with n instances that share the same host, the reliability is:

$$R_{VNFC} = R_{PH} * R_{HV} * \sum_{i=1}^{n} (-1)^{i-1} * \binom{n}{i} * \left(R_{VM} * R_{vnfc-App}\right)^{i} \quad (24)$$

As mentioned earlier, if an anti-affinity rule allows a degree of host sharing (i.e., d) or if there is a boundary (i.e., b) for the number of collocated VMs on a host, some degree of host sharing (i.e., k) may be possible between the instances of the VNFC. We have shown that the value of k can be found using equation (8). Also, in the worst case, there are $g = \lceil n/k \rceil$ groups of VNFC instances, $n - g * \lfloor n/g \rfloor$ groups with $\lceil n/g \rceil$ instances and $g * (1 + \lfloor n/g \rfloor) - n$ groups with $\lfloor n/g \rfloor$ instances for even/round-robin placement. Therefore, the reliability of a VNFC, in general, is:

$$R_{VNFC} = \left(\sum_{z=1}^{n-g*\lfloor n/g \rfloor} (-1)^{z-1} * \binom{n - g * \lfloor n/g \rfloor}{z} * \left(R_{PH} * R_{HV} * \sum_{i=1}^{\lceil n/g \rceil} (-1)^{i-1} * \binom{\lceil n/g \rceil}{i} * \left(R_{VM} * R_{vnfc-pp} \right)^{i} \right)^{z} \right)$$

$$+ \left(\sum_{y=1}^{g*(1+\lfloor n/g \rfloor)-n} (-1)^{y-1} * \binom{g * (1 + \lfloor n/g \rfloor) - n}{y} * \left(R_{PH} * R_{HV} * \sum_{i=1}^{\lceil n/g \rceil} (-1)^{i-1} * \binom{\lceil n/g \rceil}{i} * \left(R_{VM} * R_{vnfc-pp} \right)^{i} \right)^{y} \right)$$

$$- \left(\sum_{z=1}^{n-g*\lfloor n/g \rfloor} (-1)^{z-1} * \binom{n - g * \lfloor n/g \rfloor}{z} * \left(R_{PH} * R_{HV} * \sum_{i=1}^{\lceil n/g \rceil} (-1)^{i-1} * \binom{\lceil n/g \rceil}{i} * \left(R_{VM} * R_{vnfc-pp} \right)^{i} \right)^{z} \right) \quad (25)$$

$$* \left(\sum_{y=1}^{g*(1+\lfloor n/g \rfloor)-n} (-1)^{y-1} * \binom{g * (1 + \lfloor n/g \rfloor) - n}{y} * \left(R_{PH} * R_{HV} * \sum_{i=1}^{\lceil n/g \rceil} (-1)^{i-1} * \binom{\lceil n/g \rceil}{i} * \left(R_{VM} * R_{vnfc-pp} \right)^{i} \right)^{y} \right)$$

Equation (25) generalizes all the previously discussed cases for calculating the reliability of a VNFC (R_{VNFC}). Using Equation (13) and (25), we can calculate the VNFC failure rate (λ_{VNFC}), which is:

$$\lambda_{VNFC} = -\frac{\ln R_{VNFC}}{t} \quad (26)$$

4.4.2.2 Failure Rate of Internal VLs

As mentioned earlier, an IntVL is assumed to fail if all its instances fail simultaneously. Like in the case of the calculation of the failure rate of VNFCs, the failure rate of IntVLs can be calculated based on their reliability. Assuming that an IntVL instance is an overlay network on top of the physical network (not nested overlay networks), the reliability of one IntVL instance is equal to the

reliability of its undelaying physical network (i.e., physical links and nodes). Therefore, if no anti-affinity rule is applied to an IntVL, in the worst case, any physical layer failure causing a failure for an IntVL instance may lead to the failure of all IntVL instances that are sharing partially or completely the same physical network. Thus, in the worst case, the reliability of the IntVL (R_{IntVL}) is equal to the reliability of one of its instances (R_{intvl}).

$$R_{IntVL} = R_{intvl} \tag{27}$$

If an anti-affinity rule is defined for an IntVL, the reliability of its instances is independent. Therefore, for an IntVL with n redundant instances:

$$R_{IntVL} = \sum_{i=1}^{n} (-1)^{i-1} * \binom{n}{i} * R_{intvl}^{i} \tag{28}$$

In equations (27) and (28), we assume the same reliability for all instances of the IntVL.

Therefore, the failure rate of an IntVL (λ_{IntVL}) is calculated as follows:

$$\lambda_{IntVL} = -\frac{\ln R_{IntVL}}{t} \tag{29}$$

4.4.2.3 A Method for Calculating the Failure Rate of a VNF Instance

The failure rate of a VNF instance at one scaling level can be calculated using equations (12), (26), and (29). However, the number of VNFC and/or IntVL instances changes for the different VNF scaling levels. As a result, the reliability of the VNF instance changes according to the VNF scaling. Therefore, we define the failure rate of a VNF instance ($\lambda_{vnf-max}$) as the maximum among the failure rates of all its scaling levels, which then can be guaranteed.

We propose the following method to determine the failure rate of a given VNF instance. We assume the reliability or failure rate of VNFC applications, VMs, HV, PH, and IntVLs are given.

Method 2: VNF instance failure rate calculation method

Step 1: Set $\lambda_{vnf-max} = 0$

Step 2: For each VNF scaling level, perform steps 2-1 to 2-6

Step 2-1: Calculate the reliability of each VNFC for this scaling level using equation (25)

Step 2-2: Calculate the failure rate of each VNFC for this scaling level using equation (26)

Step 2-3: Calculate the reliability of each IntVL for this scaling level

- If no anti-affinity rule is applied, use equation (27)
- Otherwise, use equation (28)

Step 2-4: Calculate the failure rate of each IntVL for this scaling level using equation (29)

Step 2-5: Calculate the failure rate of the VNF instance (λ_{vnf}) for this scaling level using equation (12)

Step 2-6: If λ_{vnf} for this scaling level is higher than $\lambda_{vnf-max}$, set $\lambda_{vnf-max} = \lambda_{vnf}$

In Method 2, the final value of $\lambda_{vnf-max}$ represents the guaranteed maximum failure rate of the VNF instance.

4.5 VNF Availability, Outage Time, and Service Disruption

4.5.1 VNF Availability

The guaranteed minimum availability of a VNF instance ($A_{vnf-min}$) on a given host is calculated using Method 1 (see Sub-Section 4.4.1.3). The availability of a VNF is derived from the availability of its instances. Assuming a VNF with three active instances (i.e., N=3) and one standby instance (i.e., M=1), where the availability of all instances is the same (i.e., all the instances use the same hosting type):

$$A_{vnf-min_1} = A_{vnf-min_2} = A_{vnf-min_3} = A_{vnf-min_4} = A_{vnf-min} \quad (30)$$

The availability goal is met if no more than one instance fails for this VNF. As a result, the availability of this VNF is:

$$A_{VNF} = \left(A_{vnf_1} * A_{vnf_2} * A_{vnf_3} * \left(1 - A_{vnf_4}\right)\right) + \left(A_{vnf_1} * A_{vnf_2} * \left(1 - A_{vnf_3}\right) * A_{vnf_4}\right) + \left(A_{vnf_1} * \left(1 - A_{vnf_2}\right) * A_{vnf_3} * A_{vnf_4}\right) + \left(\left(1 - A_{vnf_1}\right) * A_{vnf_2} * A_{vnf_3} * A_{vnf_4}\right) + \left(A_{vnf_1} * A_{vnf_2} * A_{vnf_3} * A_{vnf_4}\right) \quad (31)$$

Therefore, considering equation (30), A_{VNF} is:

$$A_{VNF} = 4 * A_{vnf}^3 * \left(1 - A_{vnf}\right) + A_{vnf}^4 \quad (32)$$

Equation (32) can be re-written as:

$$A_{VNF} = \binom{4}{3} A_{vnf}^3 * \left(1 - A_{vnf}\right)^1 + \binom{4}{4} A_{vnf}^4 * \left(1 - A_{vnf}\right)^0 \quad (33)$$

. We can generalize equation (33) as equation (34) for a VNF with N active and M standby instances, where any standby can replace any active instance of the VNF.

$$A_{VNF} = \sum_{k=0}^{M} \binom{N+M}{N+k} A_{vnf-min}^{N+k} * \left(1 - A_{vnf-min}\right)^{M-k} \quad (34)$$

$$where\ N \geq 1\ and\ M \geq 0$$

4.5.2 VNF Outage Time

Redundancy is a key requirement for fault tolerance. However, even if there are enough standby VNF instances, the outage time of the VNF functionality may be unacceptable with respect to the expected availability if the failure detection and/or recovery times are too long. To meet the expected availability, the outage time of the functionality needs to be not more than the acceptable outage time. If all active instances of a VNF (N) fail at the same time, the VNF functionality is not provided,

and there is a service outage at the NS functionality level. If the maximum failure rate of the VNF is λ_{VNF}, and the $MTTR_{vnf}$ denotes the mean time to repair/recover a VNF instance, then the Outage Time of the VNF (OT_{VNF}) in a given period is:

$$OT_{VNF} = \lambda_{VNF} * MTTR_{vnf} \tag{35}$$

The λ_{VNF} is derived from the failure rate of VNF instances. The guaranteed maximum failure rate of a VNF instance ($\lambda_{vnf-max}$) for a given infrastructure is calculated using Method 2 (see Sub-Section 4.4.2.3). Using equation (13), the guaranteed reliability of the VNF instance for a period of t is [76]:

$$R_{vnf}(t) = e^{-\lambda_{vnf-max}*t} \tag{36}$$

Then, the reliability of N (active) instances ($R_{VNF}(t)$) is calculated using equation (37) [78].

$$R_{VNF}(t) = 1 - \left(1 - R_{vnf}(t)\right)^N \tag{37}$$

Therefore, the λ_{VNF} for a given period of t would be:

$$\lambda_{VNF} = -\frac{ln\left(1 - \left(1 - e^{-\lambda_{vnf-max}*t}\right)^N\right)}{t} \tag{38}$$

In equation (35), we have no control over the failure rate of the VNF. Thus, in case we need to adjust the OT_{VNF}, only the $MTTR_{vnf}$ can be adjusted. If the recovery mechanism for a VNF is failover and the active instances checkpoint to a peer, the $MTTR_{vnf}$ is calculated as the summation of its Failure Detection Time (FDT_{vnf}), the Failover Time (FoT_{vnf}), and the Takeover Time (ToT_{vnf}). FoT is the time needed to assign active role to a standby instance. When the active role is assigned to an instance it takes ToT to prepare itself to start serving.

$$MTTR_{vnf} = FDT_{vnf} + FoT_{vnf} + ToT_{vnf} \quad (39)$$

For the restart recovery mechanism, if the active instances checkpoint to their peers, the $MTTR_{vnf}$ is calculated as the summation of its FDT_{vnf}, Restart Time (RT_{vnf}), and ToT_{vnf}. Note that when an instance restarts and has the active role, it takes ToT for the instance to get ready to serve.

$$MTTR_{vnf} = FDT_{vnf} + RT_{vnf} + ToT_{vnf} \quad (40)$$

For a VNF that checkpoints to a DB, the recovery time also depends on the time to retrieve a checkpoint from the DB. We assume the network delay (i.e., CND) in transmitting a checkpoint is equal when the checkpoint is sent to the DB or received from it. Therefore, for the failover mechanism, the $MTTR_{vnf}$ is:

$$MTTR_{vnf} = FDT_{vnf} + FoT_{vnf} + ToT_{vnf} + CND_{vnf} \quad (41)$$

For the restart recovery, the $MTTR_{vnf}$ is:

$$MTTR_{vnf} = FDT_{vnf} + RT_{vnf} + ToT_{vnf} + CND_{vnf} \quad (42)$$

In equations (39) to (42), we can assume that for every VNF, the average failover, takeover, restart times are known. The CND_{vnf} is adjustable, and later we show how it is calculated. The FDT_{vnf} is also adjustable by configuring the HMR. In the worst case, the FDT is the summation of the Health-check Interval ($HI = \frac{1}{HMR}$) and a timeout [79]. So, FDT_{vnf} in the worst case would be:

$$FDT_{vnf} = HI_{vnf} + timeout_{vnf} \quad (43)$$

The goal of the *timeout* is to reduce false-positive failure detections. If the timeout is configurable, it should be greater than the network delay between the monitoring agent/peer and the monitored application/node.

4.5.3 VNF Disruption Time

According to the definition of SDT, the disruption time of a VNF due to a single failure is the summation of the $MTTR_{vnf}$ and the $TBFLC_{vnf}$. Therefore, the SDT of a VNF for all failures in a given period is:

$$SDT_{VNF} = \lambda_{VNF} * (MTTR_{vnf} + TBFLC_{vnf}) \quad (44)$$

In equation (44), the $TBFLC$ may be adjustable for some VNFs, if there are multiple networking options to adjust the CND and/or if the CpI is configurable. The TBFLC for a VNF is calculated differently for different kinds of CpI and checkpointing methods, as described below:

- **Constant CpI**: the active VNF instance checkpoints at fixed intervals. However, the CpI value may or may not be configurable. If the interval is configurable, the CpI can usually be chosen from a predefined set of discrete values.
- **Variable CpI**: the active VNF instance creates a checkpoint whenever its state changes. Therefore, the CpI is not configurable.
- **Synchronous checkpointing**: there is no state change until the checkpoint is committed. As a result, the next checkpoint preparation cannot start until the previous one is committed [80, 81].

- **Asynchronous checkpointing**: the checkpointing operations are performed independently from each other in this case, which means that the preparation of the next checkpoint can start before the previous one has been committed. [80, 81].

In the case of variable CpI and synchronous checkpointing, the worst case happens when there is a failure during checkpointing just before committing the checkpoint. Hence, the TBFLC is:

$$TBFLC_{vnf} in\ the\ Worst\ Case = Checkpoint\_Preparation\_Time + CND\ + Checkpoint\_Commitment\_Time \quad (45)$$

For synchronous checkpointing with constant CpI, the worst case is also when the failure happens just before committing a checkpoint. Thus, the state is recovered from the previous checkpoint, which was prepared at the beginning of the previous interval. Therefore, the TBFLC is:

$$TBFLC_{vnf} in\ the\ Worst\ Case = 2 * Checkpoit\_Preparation\_Time + 2 * CND\ + 2 * Checkpoint\_Commitment\_Time\ + CpI \quad (46)$$

For asynchronous checkpointing with constant CpI, the worst case is when the latest checkpoint, and maybe some other previously sent checkpoints still in transit, have not been committed yet when the failure happens. Therefore, the state recovered is the state at the beginning of preparing the latest committed checkpoint. Thus, the TBFLC is:

$$TBFLC_{vnf} in\ Worst\ Case = \ 2 * Checkpoit\_Preparation\_Time + CND\ + Checkpoint\_Commitment\_Time\ + CpI \quad (47)$$

For asynchronous checkpointing with variable CpI, the VNF instance is stateful during the checkpoint preparation, transfer, and committing time. Thus, the TBFLC in the worst case is:

$$TBFLC_{vnf} in\ the\ Worst\ Case = Checkpoint\_Preparation\_Time + CND\ + Checkpoint\_Commitment\_Time \quad (48)$$

Equations (45) to (48) provide the calculation of TBFLC for one VNF instance. For a failed VNF (i.e., all active instances fail simultaneously), in the worst case, the TBFLC of each instance is the worst possible TBFLC. Therefore, in the worst case, the TBFLC of a VNF is equal to the worst-case TBFLC of one VNF instance.

The networking delay in sending a message from a source to a destination is the summation of the transmission delay and the propagation delay [82]. So, the CND is calculated by equation (49).

$$CND = Transmission\ Delay + Propagation\ Delay \tag{49}$$

In our case, the checkpointing source is the active VNF instance, the message is the checkpoint data, and the destination is a DB or a peer VNF instance. At recovery, if the checkpoint is stored in a DB, it becomes the source and the standby VNF instance the destination. The transmission delay is derived from the checkpoint data size divided by the network bandwidth. The propagation delay (also referred to as networking latency) depends on the distance between the source and the destination, and the transmission speed of the network.

$$Transmission\ Delay = \frac{Checkpointing\ Data\ Size}{Bandwidth} \tag{50}$$

$$Propagation\ Delay = \frac{Distance}{Transmission\ Speed} \tag{51}$$

According to [9], the bandwidth and propagation delay of VLs can be indicated for NSs. So, for a known average checkpoint size, we can determine an appropriate bandwidth and/or ask for an appropriate networking latency/propagation delay to adjust the CND. We also assume that the failures of the VLs used only for checkpointing do not affect the functionalities of the NS. However, when we determine the required redundancy for VLs (the calculation method will be presented later

in this chapter), we expect the same redundancy for checkpointing VLs to control the checkpoint data loss and service disruption.

4.5.4 VNF Service Data Disruption

SDD of a VNF (SDD_{VNF}) due to a failure, in the worst case, is the product of the VNF data rate and the TBFLC of a VNF instance.

$$SDD_{VNF} = (Data\_Rate_{VNF}) * (TBFLC_{vnf}) \quad (52)$$

In other words, after recovering from the last checkpoint, the data that was sent from the time of this checkpoint up until the failure should be resent.

4.6 VL Availability and Redundancy

4.6.1 Expected Availability of VLs

The availability of the functionality delivered via an NFP (A_{NFP}) is the product of the availability of the VNFs, and the VLs should be in the NFP.

$$A_{NFP} = (VNFs\ availability) * (VLs\ availability) \quad (53)$$

If an RA is requested for the functionality provided through an NFP, the product of the availability of the VNFs and the VLs should not be less than the required availability.

$$(VNFs\ availability) * (VLs\ availability) \geq RA \quad (54)$$

Assuming VNFs and VLs contribute equally to fulfill the RA, the availability of VNFs should satisfy inequation (55).

$$VNFs\ availability \geq \sqrt{RA} \quad (55)$$

And the availability of VLs should satisfy inequation (56).

$$VLs\ availability \geq \sqrt{RA} \tag{56}$$

If the NFP includes X types of VLs, the VlEA for each VL (expected availability of each VL) should satisfy inequation (57).

$$VlEA \geq \sqrt[2*X]{RA} \tag{57}$$

If an ASDT in the period of t is requested for the functionality provided through an NFP, the SDT of the NFP is the summation of the outage times caused by VLs failures and the SDTs of its VNFs. To meet this ASDT, if we assume equal contribution for VNFs and VLs to fulfill the ASDT, then inequation (58) should be satisfied for VLs:

$$VLs\ availability \geq \sqrt{\frac{t - \frac{ASDT_{NFP}}{2}}{t}} \tag{58}$$

Therefore, the VlEA should satisfy the following inequation:

$$VlEA \geq \sqrt[2*X]{\frac{t - \frac{ASDT_{NFP}}{2}}{t}} \tag{59}$$

4.6.2 VL Redundancy

The maximum availability of VL instances (A_{vl-max}) provided by the NFVI is an input for the design-time approach. The availability of VLs for an NS can be requested according to [9] in the same way as the networking latency. If the A_{vl-max} is not less than the VlEA for a VL, one VL instance is enough for the VL to satisfy its expected availability, and VlEA is the constraint for the

VL availability. Otherwise, the VL requires redundancy. The availability of a VL (A_{VL}) with n redundant instances is:

$$A_{VL} = 1 - (1 - A_{vl-max})^n \quad (60)$$

Therefore, inequation (61) can be used to determine the number of instances (m) to keep the availability of the redundant VL instances (A_{VL}) greater than or equal to the $VlEA$:

$$m \geq log_{(1-A_{vl-max})}(1 - VlEA) \quad (61)$$

4.7 NFP Service Outage, Service Disruption, and Resource Cost

4.7.1 NFP Outage Time

To meet the RA for an NS functionality, we can calculate the Acceptable Downtime of the NFP (ADT_{NFP}), which is:

$$ADT_{NFP} = (time\ period) * (1 - RA) \quad (62)$$

Then, according to inequations (55) and (56), the Outage Time of all VNFs together (OT_{VNFs}) should be less than $\frac{ADT_{NFP}}{2}$. As discussed earlier, the outage time of a VNF is calculated using equation (35). Therefore, for an NFP with Y different VNFs, the outage time of the NFP due to VNFs failures ($OT_{NFP_{VNFs}}$) is:

$$OT_{NFP_{VNFs}} = OT_{VNFs} = \sum_{i=1}^{Y} OT_i \quad (63)$$

Therefore, to meet the RA for an NS functionality, we should calculate the ADT_{NFP} and adjust the $MTTR_{vnf}$ of the different VNFs to keep the $OT_{NFP_{VNFs}}$ less than or equal to the $\frac{ADT_{NFP}}{2}$. Also, all VLs should meet their VlEA.

4.7.2 NFP Service Disruption Time

To meet an $ASDT_{NFP}$, if we assume that VNFs and VLs contribute equally, then the SDTs of VNFs should not be more than $\frac{ASDT_{NFP}}{2}$. If an NFP includes Y different VNFs, using equation (44), the overall SDT of the functionality provided through the NFP due to VNFs failures ($SDT_{NFP_{VNFs}}$) is:

$$SDT_{NFP_{VNFs}} = \sum_{i=1}^{Y} SDT_i \tag{64}$$

For each VNF, depending on the checkpointing method, the worst-case TBFLC is used in equation (44) to calculate the worst-case scenario for the NFP by equation (64). When the tenant asks for the ASDT of an NS functionality, we should adjust the $MTTR_{vnf}$ and $TBFLC_{vnf}$ of the different VNFs so that the $SDT_{NFP_{VNFs}}$ is less than or equal to the $\frac{ASDT_{NFP}}{2}$. Also, all VLs of the NFP should meet the VlEA.

4.7.3 NFP Service Data Disruption

For an NFP with one or more VNFs, there is a ratio between each VNF data rate and the NFP data rate [73].

$$Data\_Rate_{NFP} = Data\_Rate_{VNF} * Ratio_{VNF} \tag{65}$$

Therefore, the SDD at the NFP level due to the failure of one VNF is:

$$SDD_{NFP} = (SDD_{VNF}) * (Ratio_{VNF}) \quad (66)$$

Thus, to satisfy the ASDD for the functionality provided through an NFP (i.e., to satisfy $SDD_{NFP} \leq ASDD$) with Y different VNFs, we should adjust the $TBFLC_i$ of each VNF_i of the NFP to satisfy inequation (67).

$$TBFLC_i \leq \frac{ASDD}{(Data\_Rate_i)*(Ratio_i)} \text{ , where } 1 \leq i \leq Y \quad (67)$$

We expect the $Data\_Rate_{VNF}$ and the $Ratio_{VNF}$ as input for all VNFs for each NFP.

4.7.4 Cost Function

When the number of required standby instances for a VNF is calculated, the computing cost should also be addressed. So, we should determine the minimum number of required instances for each VNF at each scaling level that satisfies the VnfEA to avoid overprovisioning computing resources for the VNF. It is possible to have multiple hosting types to choose from for VNF placement. Different hosting types may have different availability (A_{PH}) and result in different availability of the VNF instance (A_{vnf}). Therefore, for different hosting types, we may end up with a different required number of standby instances for a VNF. Also, different hosting types may have a different cost. For example, placing a VNF on one hosting type with better availability may be twice as expensive as placing the same VNF on a host with lower availability.

Therefore, we define a cost function to be able to choose a hosting option, which results in a lower computing cost for the VNFs. We assume that all VNFs of the given NsDF will be placed on the same hosting type. If C_h shows the cost for hosting one VNFC of an instance of VNF_i, the hosting cost of one VNF instance for the average number of VNFC instances ($\bar{n}$) for all the VNF scaling levels is:

$$C_h(vnf_i) = \bar{n} * C_h \tag{68}$$

Then, the computing cost of each VNF_i (i.e., VNF profile) at the j^{th} scaling level of the NsDF with N active and M standby instances is calculated by equation (69).

$$C_C(VNF_{i,j}) = (N_{i,j} + M_{i,j}) * C_h(vnf_i) \tag{69}$$

Similarly, we can use the average number of VNF instances to calculate the overall computing cost ($C_C(VNF_i)$) for each VNF_i for all NS scaling levels.

$$C_C(VNF_i) = (\overline{N_\iota + M_\iota}) * C_h(vnf_i) \tag{70}$$

Therefore, for an NsDF with Y different VNFs, the overall computing cost would be:

$$C_C(NsDF) = \sum_{i=1}^{Y} C_C(VNF_i) \tag{71}$$

To minimize the networking cost, we define a cost function for NFPs, which is calculated differently depending on the protection mechanisms configured (e.g., health-check monitoring, checkpointing), which in turn depend on the tenant's requirements. We consider the other portion of the networking cost constant and out of our control.

The networking cost for VNF_i of an NFP which has N_j active instances at the j^{th} scaling level is:

- If the requirement is to satisfy an RA

$$C_N(VNF_{i,j}) = N_{i,j} * (MHR_{i,j}) \tag{72}$$

- If the tenant asks for an ASDT or ASDD

$$C_N(VNF_{i,j}) = N_{i,j} * (\alpha * MHR_{i,j} + \beta * \frac{1}{CpI_{i,j}} + \gamma * \frac{1}{CNd_{i,j}}) \tag{73}$$

According to the cost function (73), regardless of whether the HMR increases, the CpI decreases, or a faster network is selected, the networking cost for the VNF increases. To be able to adjust the importance of these three configuration parameters, we use coefficients (i.e., α, β, and γ) in the equation.

Accordingly, the total cost for the NFP at scaling level j is:

$$C_N(NFP_j) = \sum_{i=1}^{y} C_N(VNF_{i,j}) \tag{74}$$

4.8 Network Service Availability and Continuity Requirements Mapping

4.8.1 Mapping Method

A tenant may ask for different kinds of requirements for different NS functionalities. Also, the required values may differ for functionalities with the same requirements. So, for each NS functionality, there is a specific kind of requirement with a specific value to be met. The goal of our mapping method is to satisfy the corresponding requirement for each NS functionality and minimize the networking cost simultaneously; then to select the hosting option with the lowest computing cost considering all the VNFs of the NsDF. Thus, first, we find for the VNFs the optimal HMR and CpI values and networking options. Then, we determine the minimum required number of standby instances for each VNF/VL that keeps the probability of having enough active VNF/VL instances higher than VnfEA/VlEA. To minimize the computing resource cost, we repeat the whole process using different hosting types and select the hosting option with the minimum total cost for the NsDF.

For some VNFs, the VNF instance availability may be enough to satisfy the VnfEA without any standby instance(s) for some/all scaling levels. In such a case, the recovery method for the VNF

is restart recovery. Therefore, we need to use the MTTR of the restart in equations (39) to (42) for each VNF and then apply the method of finding the optimal values/options for the HMR, the CpI, and networking. However, in the beginning, we do not know yet whether the VNF instance availability will satisfy the applicable VnfEA. One way to solve this problem is to perform the method at each scaling level for equations (39)/(41) and (40)/(42) for all VNFs and choose the solution which results in the highest number of VNFs without any standby instance. If an NFP has Y different VNFs, the time complexity of examining all combinations of MTTRs for all VNFs is:

$$\text{Time Complexity} = O(2^Y) \tag{75}$$

This exponential time complexity is not acceptable for a large Y, i.e., a large number of VNFs. In the context of our work, this problem can be avoided as follows: first, we calculate the VnfEA for each VNF, assuming that that failover mechanism is used for the VNFs. Then, we calculate the availability of each VNF with zero standby and compare it with the VnfEA. If the VNF without any standby can satisfy the VnfEA, we mark the VNF. At the end of this process, we have some marked and some unmarked VNFs, and we can determine the appropriate MTTR equation for each VNF; equation (39) or (41) should be used for unmarked VNFs, and equation (40) or (42) is the appropriate one for marked VNFs. We can do so because the method adjusts other configuration values (e.g., for HMR) to the difference between the two MTTRs. Next, we can calculate the number of standby instances for the unmarked VNFs, and based on them, we can tackle the calculation of the computing cost for the NsDF.

The steps of our proposed mapping method can be summarized as follows: steps 1, 2, and 3 create loops for different hosting types, for different scaling levels, and for the requested NS functionalities. Steps 4 to 6 check whether the service availability and continuity requirements can

be met for the NS. The goal of performing steps 7 to 11 is to mark VNFs and set the appropriate recovery method for each VNF at each scaling level for each hosting type. Step 12 creates a loop for the requested NS functionalities, and steps 13 and 14 are performed in this loop for marked and unmarked VNFs. Executing steps 12 to 15, we will find the optimal configuration values/options for each marked and unmarked VNF of each NFP to satisfy the requirements. Steps 16 and 17 find the required number of standbys for each unmarked VNF of the NsDF for each hosting type. Step 18 determines the required redundancy for VLs. Steps 19 and 20 find the hosting type with minimum computing cost for all VNFs of the NsDF.

Figure 4-9 depicts the flowchart of the requirement mapping method.

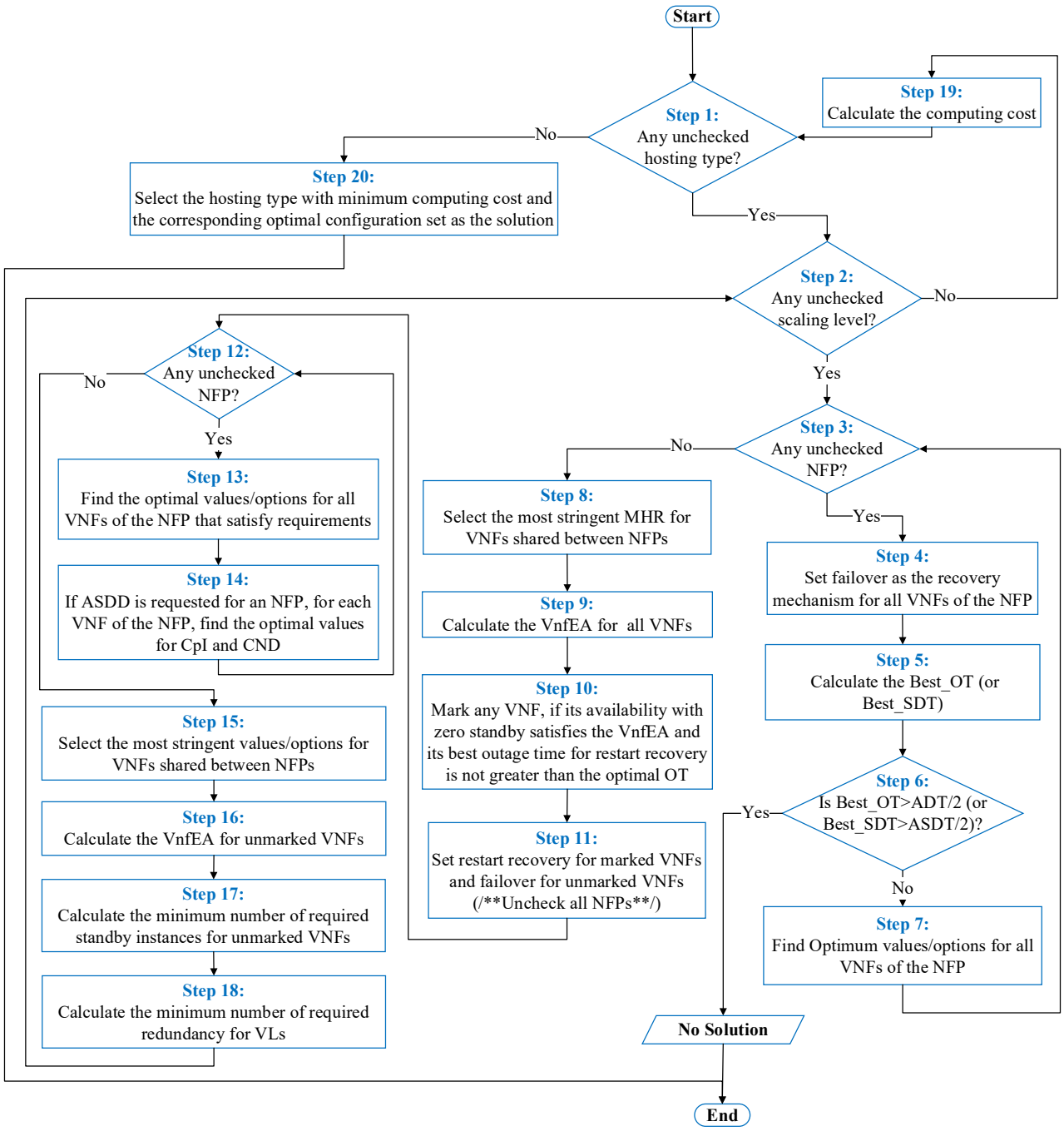


Figure 4-9: Flowchart of the requirements mapping method

The followings are the explanation of the steps in the flowchart of Figure 4-9:

- **Step 1**: For each hosting type, perform steps 2 to 19
- **Step 2**: For each scaling level, perform steps 3 to 18.
- **Step 3**: For each NFP (i.e., NS functionality), perform steps 4 to 7.
- **Step 4**: Set failover as the recovery mechanism for all VNFs of the NFP to use equation (39) or (41) for all VNFs.
- **Step 5**: For an NFP, either the RA or the ASDT is requested. If the RA is requested, calculate the $\frac{ADT_{NFP}}{2}$ using equation (62). Then, calculate the best possible $OT_{NFP_{VNFs}}$ ($Best\_OT$) for the NFP based on equation (63). The $Best\_OT$ can be found using the maximum allowed value for MHR_i of each VNF_i of the NFP. If the ASDT is requested, calculate the best possible $SDT_{NFP_{VNFs}}$ ($Best\_SDT$) for the NFP according to equation (64). The $Best\_SDT$ can be found using the maximum allowed value for MHR_i, the minimum allowed value for CpI_i, and choosing the best available networking option for all VNF_i of the NFP.
- **Step 6**: If the RA is requested, compare the $Best\_OT$ with the $\frac{ADT_{NFP}}{2}$. If $Best\_OT > \frac{ADT_{NFP}}{2}$, the RA for this NFP cannot be achieved with the given VNFs, and there is no solution. If an ASDT is requested, compare the $Best\_SDT$ with the $ASDT$. If $Best\_SDT > \frac{ASDT_{NFP}}{2}$, then the ASDT for this NFP cannot be achieved with the given VNFs. Otherwise, go to the next step.
- **Step 7**: If the RA is requested and $Best\_\mathrm{OT} = \frac{ADT_{NFP}}{2}$, the best value of MHR_i for each VNF_i is the only acceptable configuration. If $Best\_\mathrm{OT} < \frac{ADT_{NFP}}{2}$, there may be multiple values of MHR_i that can satisfy the requirement. Then, find the optimal values for the

MHR_i for all VNF_i of the NFP that minimize the cost for the scaling level (equation (74)) and satisfies the requirement. If an ASDT is requested and $Best\_SDT = \frac{ASDT_{NFP}}{2}$, the best values/options for MHR_i, CpI_i, and networking for all VNF_i are the only acceptable configuration. If $Best\_SDT < \frac{ASDT_{NFP}}{2}$, there may be multiple MHR_i and CpI_i values, and networking options that can satisfy the requirement. Then, find the optimal values/options for the MHR_i, CpI_i and networking for all VNF_i of the NFP that minimize the cost for the scaling level (equation (74)) and satisfies the requirement. To find the optimal configuration to satisfy the RA, we calculate equations (63) and (74) for all possible MHR_i of all VNF_i of the NFP, and we choose the configuration values that result in the minimum cost while satisfying the $\frac{ADT_{NFP}}{2}$. To find the optimal configuration to satisfy an ASDT, we calculate equations (64) and (74) for all possible combinations of the MHR_i, CpI_i values and the networking options for all VNF_i of the NFP and we choose the configuration values that result in the lowest cost while satisfying the $\frac{ASDT_{NFP}}{2}$. Thus, to find the optimal configuration in this step, we examine all possible combinations, that is, we perform a *complete search*.

- **Step 8**: For a VNF shared between multiple NFPs, we select the most stringent MHR_i among the solutions found in step 7 for the different NFPs it is used.
- **Step 9**: Calculate for each VNF_i of the NsDF the $VnfEA_i$ using equation (76) based on the optimal value for its MHR_i selected in steps 7 and 8:

$$VnfEA_i = \frac{Uptime}{Uptime+OT_i} = \frac{1\,year - OT_i}{1\,year} \quad (76)$$

With the optimal value for MHR_i and its relation to FDT_i (i.e., equation (43)), we can calculate the OT_i using equation (35). Since RA is defined for one year, the $Uptime$ is calculated as "$one\ year$" minus OT_i.

- **Step 10**: Calculate the availability of each VNF_i using equation (34) with zero standby instance ($A_{VNF_{i-0}}$) and the best outage time of each VNF_i for restart recovery mechanism (OT_{i-best}), using the best MHR_i. If $A_{VNF_{i-0}} \geq VnfEA_i$ and $OT_{i-best} \leq OT_i$, mark the VNF. If OT_{i-best} is greater than the optimal OT_i found in step 9, the VNF_i cannot meet the same $VnfEA_i$ for restart recovery mechanism.
- **Step 11**: Set restart recovery (i.e., equation (40) or (42)) for marked VNFs and failover (i.e., equation (39) or (41)) for unmarked VNFs.
- **Step 12**: For each NFP, perform steps 13 and 14.
- **Step 13**: Find the optimal values/options for the MHR_i, and – if ASDT is requested – for the CpI_i and networking for all VNF_i of the NFP, with the selected recovery mechanism in step 11 for each VNF_i. (The same process as step 7)
- **Step 14**: If ASDD is requested for the NS functionality, for each VNF_i of the NFP, find the optimal value for the CpI_i and networking option that satisfy inequation (67). To do so, we generate all possible TBFLCs for each VNF_i using all possible CpI_i values and networking options for the VNF. Then we sort the TBFLCs of the VNF in descending order. The first TBFLC which satisfies the inequation (67) is the optimal one, and the corresponding CpI_i and networking option is selected as optimal value and option for the VNF.

- **Step 15**: For a VNF shared between multiple NFPs, we select the most stringent configuration values/options among the solutions found in steps 13 and 14 for the different NFPs.
- **Step 16**: Calculate for each unmarked VNF_i of the NsDF the $VnfEA_i$ using equation (76) based on the optimal value for its MHR_i selected in steps 13 and 15.
- **Step 17**: Find for every unmarked VNF_i using equation (34), the minimum number of standbys (i.e., M) that satisfies $A_{VNF_i} \geq \mathrm{Vnf}EA_i$.
- **Step 18**: Calculate the minimum number of VL redundancy for each VL of the NFP that satisfies inequation (61).
- **Step 19**: Assuming we are given the computing cost of each hosting type, calculate the computing cost of the NsDF using equation (71).
- **Step 20**: Select the hosting type with the minimum computing cost for the NsDF. Once we select the hosting type, we adjust the given NsDF by adding to the number of active instances the calculated number of standbys instances for the different scaling levels. In addition, we store the corresponding set of configuration values/options found in steps 13 to 15 as the optimal configuration for the NsDF.

Steps 7, 13, 17, and 18 are the main steps of the method. The goal of steps 7 and 13 is to find the optimal configuration for VNFs and minimize the networking cost if RA or ASDT is requested for an NS functionality. The goal of steps 17 and 18 is to find the minimum required redundancy which guarantees the required protection level for each unmarked VNF and VL of the NsDF at each scaling level.

The values for the MHR_i and the CpI_i are discrete values, and there is a limited number of available networking options for checkpointing. To find the optimal configuration in step 7 (and respectively in step 13), we need to examine all possible combinations of the configuration values/options for all VNFs of the NFP and select the ones that satisfy the requirement and minimize the cost function of equation (74). First, we generate all possible OT_{VNF_i} (SDT_{VNF_i}) for each VNF_i of the NFP using equation (35) (and respectively equation (44)). Then, we sort the OT_{VNF_i} (or the SDT_{VNF_i}) and determine their acceptable lower and upper bounds for each VNF_i as follows. The lower bound for VNF_i is determined by using the best configuration values in equation (35) (or in (44)). The upper bound for VNF_i can be found by using the best configuration values for all other VNFs in equation (63) (or in (64)). That is, the upper bond is the maximum acceptable OT_{VNF_i} (SDT_{VNF_i}) for the VNF_i satisfying $OT_{NFP_{VNFs}} \leq \frac{ADT_{NFP}}{2}$ (or $SDT_{NFP_{VNFs}} \leq \frac{ASDT_{NFP}}{2}$), when the OT_{VNF} (or SDT_{VNF}) of all other VNFs are at their minimum. By this, we have the possible range of OT_{VNF_i} (or SDT_{VNF_i}) for each VNF_i of the NFP. Every OT_{VNF_i} (or SDT_{VNF_i}) of each VNF_i represents a combination of configuration values (i.e., MHR_i, CpI_i, and CND_i) for the VNF_i. So, to find the optimal configuration, we examine all the possible OT_{VNF_i} (or SDT_{VNF_i}) of all VNFs of the NFP to find a combination for different VNFs that satisfies $OT_{NFP_{VNFs}} \leq \frac{ADT_{NFP}}{2}$ or $SDT_{NFP_{VNFs}} \leq \frac{ASDT_{NFP}}{2}$ and minimizes the overall cost.

In step 17, we determine the number of standbys using equations (34) and (76), starting with one (i.e., M = 1). We start from M=1 since, for unmarked VNFs, we should add at least one standby instance. We increment the number of standbys until we satisfy $A_{VNF_i} \geq \mathrm{Vnf}EA_i$.

4.8.2 Time Complexity Analysis

Steps 7 and 13 are the most time-consuming steps of our method. If the tenant asks for an ASDT, three parameters (i.e., HMR (or HI), CpI, and CND) should be optimized. The HMR (or HI) is usually configurable for VNFs, while the CpI may not be configurable for a VNF. Also, there may be one or multiple networking options for checkpointing. We assume that the acceptable intervals for the HI and the (configurable) CpI are integer values expressed in milliseconds. The upper bound for the HI and the (configurable) CpI of VNFs are determined when we find the upper bound of the SDT_{VNF_i} for each VNF, as explained earlier. Therefore, there is a limited set of possible HIs, one or a limited number of possible CpIs, and one or a limited number of networking options for each VNF. Let us assume that, on average, for a VNF there are:

- HRN number of possible configuration values for the health-check rate
- CPN number of possible configuration values for the checkpointing interval
- NON number of possible networking options for remote checkpointing

If there are Y number of different VNFs in an NFP, the time complexity for examining all possible combinations (i.e., complete search) of configuration values in step 7 (and in step 13) would be:

$$\text{Time Complexity} = O((HRN * CPN * NON)^Y) \tag{77}$$

If the tenant asks for an RA, the HI is the only parameter that should be optimized for all VNFs. So, the time complexity for a complete search for this case is:

$$\text{Time Complexity} = O(HRN^Y) \tag{78}$$

Therefore, the time complexity of the complete search at step 7 (and also at step 13) is exponential in terms of the number of VNFs as in equations (77) and. Thus, for a large number of VNFs, we may not be able to examine all possible combinations. Therefore next, we propose a heuristic search, which finds a near-optimal configuration in a timely manner.

4.8.3 Heuristic Algorithm

We have implemented the proposed method using the complete search and applied it to a few sample NSs to find the respective optimal configuration values. From these samples, we have observed that the optimal configuration values for an NFP (i.e., NS functionality) always result in a $OT_{NFP_{VNFs}}$ (and $SDT_{NFP_{VNFs}}$) very close to the $\frac{ADT_{NFP}}{2}$ (and to the $\frac{ASDT_{NFP}}{2}$). This was expected since when the mapping method finds the configuration, it should keep the $OT_{NFP_{VNFs}}$ (and $SDT_{NFP_{VNFs}}$) as close as possible to the $\frac{ADT_{NFP}}{2}$ (and to the $\frac{ASDT_{NFP}}{2}$) to minimize the networking cost function.

Based on this observation, to satisfy the RA, instead of examining all possible HMR values for each VNF, our heuristic algorithm examines only those values that satisfy the following condition:

$$0 \leq \frac{ADT_{NFP}}{2} - OT_{NFP_{VNFs}} \leq Search\_Window \tag{79}$$

If the ASDT is requested for an NS functionality, instead of examining all possible configuration values/options, our heuristic algorithm examines only those values/options that satisfy the following condition:

$$0 \leq \frac{ASDT_{NFP}}{2} - SDT_{NFP_{VNFs}} \leq Search\_Window \tag{80}$$

We show later that the configuration found by this heuristic search can be a near-optimal configuration.

Whether the RA or the ASDT is requested for an NFP, an appropriate $Search\_Window$ needs to be chosen by the NS designer for our heuristic. Selecting a value closer to zero results in less execution time, at the price of sacrificing the thoroughness of the search. The found solution always satisfies the condition of inequation (79) for the RA (and (80) for the ASDT), but the $OT_{NFP_{VNFs}}$ (or the $SDT_{NFP_{VNFS}}$) of this configuration and its total cost may not be very close to the $OT_{NFP_{VNFs}}$ (or the $SDT_{NFP_{VNFS}}$) and the total cost of the optimal configuration. The reason is that with a smaller search window, we reduce the number of combinations that step 7 (and 13) examines. A bigger $Search\_Window$ results in a more thorough search but requires more execution time. So, the NS designer can decide on the value of the $Search\_Window$ to adjust the execution time.

In the rest of this section, we present the heuristic search for the ASDT. The heuristic search for the RA is almost the same. The only difference is that to meet the RA, we only examine the HMR values of each VNF and do not consider the CpI values and the networking options.

For the heuristic search, we find the lower and upper bounds of the SDT_{VNF_i} for each VNF_i of the NFP the same way as we do for the complete search. However, once the boundaries of the SDT_{VNF_i} are found for each VNF_i, we do not explore all combinations of the SDT_{VNF} values for the different VNFs of the NFP. Instead, we consider only the combinations of SDT_{VNF} values that fulfill the condition of inequation (80).

The pseudo-code of the heuristic search is shown in Alg. 4-1. It examines recursively all the combinations of SDT_{VNF} values for all VNFs of an NFP, which means that it starts with a SDT_{VNF}

value of one VNF and adds the SDT values of other VNFs one by one. At each level of the recursion, the set of possible SDT_{VNF} (i.e., the $Reduced\_SDT_i$) is reduced according to inequation (80). To find the reduced set of possible SDT_{VNF}, we use a binary search as its execution time complexity is low. The output of the algorithm is the near-optimal configuration for the NFP, which includes a near-optimal configuration for each VNF of the NFP. The time complexity of the heuristic algorithm is $O(Y^2)$ since the main function of the algorithm at line 9 runs for *Y* times and at each of the iterations, the function is called again recursively at line 13 or 19 for *Y-i* times (i.e., in total, *findConfiguration* function is called for $\frac{Y*(Y-1)}{2}$ times).

Alg. 4-1: Heuristic Algorithm Pseudo-code

```
 1: for i ∈ {1, ... , Y} do      // Y is the number of VNFs
 2:    SDT_i ← ø;
 3: end for
 4: Generate all the possible SDT_VNF for each VNF_i separately;
 5: Sort SDT_VNF for all VNFs in ascending order;
 6: Find the lower and upper bound of SDT_VNF for each VNF_i;
 7: Create a set of acceptable SDT_VNF for each VNF_i separately (SDT_i);
 8: i ← 1, optimalCost ← ∞, configuration ← ø;
 9: function findConfiguration(i, ASDT_NFP / 2)
10:   if (i < Y) then
11:       Reduced_SDT_i ← All SDT_VNF of SDT_i that satisfy (SDT_NFP_VNFs ≤ ASDT_NFP / 2);
12:       for SDT_VNF ∈ Reduced_SDT_i do
13:           findConfiguration(i + 1, ASDT_NFP / 2 - SDT_VNF);
14:       end for
15:   end if
16:   if (i == Y) then
17:       Reduced_SDT_i ← All SDT_VNF of SDT_i that satisfy (0 ≤ ASDT_NFP / 2 − SDT_NFP_VNFs ≤ Search_Window);
18:       for SDT_VNF ∈ Reduced_SDT_i do
19:           findConfiguration(i + 1, ASDT_NFP / 2 - SDT_VNF);
20:       end for
21:   end if
22:   if (i > Y) then
23:       cost ← cost of the selected configuration values;
24:       if (cost < optimalCost) then
25:           optimalCost ← cost;
26:           configuration ← selected configuration values for all VNFs;
27:       end if
28:    end if
29: end function
```

4.8.4 Experiments and Evaluation

Here, we present the experiments conducted for the proposed method implemented with both search strategies to compare their accuracies and execution times. The first implementation performs a complete search by exploring all possible combinations of the configuration values/options in steps 7 and 13 to meet an ASDT. The other implementation uses our proposed heuristic search and examines a limited set of configuration combinations as described earlier.

First, we consider a sample NS with one scaling level and one available hosting type. The NsDF provides one functionality (i.e., it has only one NFP) for which the requested ASDT is 31536 ms per year (i.e., 0.000001 of a year). The NFP has four VNFs (i.e., $Y = 4$) with configuration options shown in Table 4-1. All VNFs checkpoint to a peer. We assume that VLs have no failure (100% available) for this example since these experiments aim to compare the results of complete and heuristic searches for steps 7 and 13 of our proposed method.

Table 4-1: VNFs and networking details for a sample NS

	N	AFR_{VNF} (Per year)	Min. HI (ms)	Min. CpI (ms)	A_{vnf}	CND1 (ms)	CND2 (ms)	CND3 (ms)
VNF_1	6	3	100	50	0.999	50	180	500
VNF_2	10	2	50	10	0.99	150	200	3500
VNF_3	5	3	150	50	0.9999	40	100	2000
VNF_4	5	3	100	50	0.999	40	100	500

There are three available networking options for checkpointing with different CNDs. The minimum HI ($\frac{1}{MHR}$) and the CpI of each VNF are also given in Table 4-1. For VNF_1, the health-check interval is configurable with increments of 100 ms. For all other VNFs, the health-check increment is 50 ms. Similarly, for VNF_2 and VNF_4, the CpI is configurable with increments of 200

ms. CpI is not configurable for VNF_1 and VNF_3. Failover time is $FoT = 10$ ms for all VNFs. The takeover time for VNF_1 is $ToT = 15$ ms and for the others is $ToT = 10$ ms.

We applied the method with the complete and heuristic searches to this sample NS. The $Search\_Window$ for the heuristic algorithm was 1000 ms. For all the experiments, we used: $\alpha = \beta = \gamma = 1$. Table 4-2 shows the optimal configuration values and the number of required standbys calculated using the complete search. The cost of this configuration would be $C_N(NFP) = 0.0841$, and the SDT of the NFP is calculated as $SDT_{NFP} = 31.30$ seconds.

Table 4-2: Optimal configuration values, using complete search

	HI	CPI	Net. Delay	M
VNF_1	200 ms	50 ms	500 ms	2
VNF_2	150 ms	210 ms	200 ms	4
VNF_3	350 ms	50 ms	2000 ms	1
VNF_4	300 ms	250 ms	100 ms	2

Table 4-3 shows the results of the method using the heuristic search. The cost for this configuration is $C(NFP) = 0.0867$ and the SDT of the NFP is $SDT_{NFP} = 31.45$ seconds.

Table 4-3: Near-optimal configuration values, using heuristic search

	HI	CPI	Net. Delay	M
VNF_1	300 ms	50 ms	180 ms	2
VNF_2	150 ms	210 ms	200 ms	4
VNF_3	350 ms	50 ms	2000 ms	1
VNF_4	250 ms	250 ms	100 ms	2

Comparing the output of the two implementations shows that the results with the complete and heuristic searches are very close. The SDT with the heuristic search is only 0.5% different from the

SDT found by the complete search. The cost of the solution found with the heuristic search differs only by 3.1% from the cost of the solution found with the complete search. The method with the complete search results in slightly better cost and SDT.

We expected that with a higher number of VNFs, the execution time with the complete search would increase exponentially. So, we experimented with the two implementations using different numbers of VNFs in the NFP to benchmark the execution time and the accuracy of the heuristic search. For this experiment, the VNFs and networking information are given in Table 4-4.

Table 4-4: VNFs and networking details for one scaling level of the NFP

	N	AFR_{VNF} (Per year)	Min. HI (ms)	Min. CpI (ms)	A_{vnf}	CND1 (ms)	CND2 (ms)	CND3 (ms)
VNF_1	4	2	100	20	0.99	30	100	300
VNF_2	7	3	100	40	0.999	100	200	2000
VNF_3	6	2	100	50	0.999	50	100	2000
VNF_4	3	3	150	50	0.9999	100	500	3000
VNF_5	7	2	100	40	0.999	50	180	500
VNF_6	11	1	50	10	0.99	150	200	3500
VNF_7	4	4	100	100	0.999	40	100	2000
VNF_8	6	3	100	50	0.9999	40	100	2000
VNF_9	6	1	100	40	0.9999	50	150	600
VNF_{10}	8	2	50	30	0.999	10	200	3000
VNF_{11}	5	2	50	20	0.999	40	100	2500
VNF_{12}	6	1	50	50	0.999	40	150	2500
VNF_{13}	7	2	50	20	0.99	30	100	500
VNF_{14}	2	5	100	40	0.9	10	200	2000
VNF_{15}	6	2	50	30	0.999	10	100	2000
VNF_{16}	3	1	100	20	0.9999	10	200	1000
VNF_{17}	7	1	50	10	0.999	20	200	500
VNF_{18}	9	1	50	10	0.9999	10	180	1500
VNF_{19}	4	2	50	10	0.999	30	100	2000
VNF_{20}	10	1	50	10	0.9999	10	150	2000

Table 4-5 shows the execution times for each implementation, the number of explored combinations, as well as their outputs for the cost and SDT for the NFP.

Table 4-5: Optimal/near-optimal configuration values, using both implementations

Number of VNFs (Y)	Algorithm	Execution time (ms)	Number of combinations	Cost	SDT
2	Complete	3	8,786	0.0613	30.9
	Heuristic	1	135	0.0617	31.49
3	Complete	38	237775	0.0876	31.47
	Heuristic	16	8,026	0.0881	31.52
4	Complete	332	12,392,801	0.0979	31.44
	Heuristic	83	398952	0.0979	31.44
5	Complete	21,719	775,919,707	0.1339	31.3
	Heuristic	871	6313688	0.1339	31.3
6	Complete	N/A	N/A	N/A	N/A
	Heuristic	4,905	42,686,536	0.1593	31.5
7	Complete	N/A	N/A	N/A	N/A
	Heuristic	16,034	75,285,642	0.1976	31.48
8	Complete	N/A	N/A	N/A	N/A
	Heuristic	29,849	78,694,978	0.2528	31.41
12	Complete	N/A	N/A	N/A	N/A
	Heuristic	196,268	368,862,956	0.6578	31.52
16	Complete	N/A	N/A	N/A	N/A
	Heuristic	285,335	490,957,472	0.998	31.48
20	Complete	N/A	N/A	N/A	N/A
	Heuristic	623,901	1,123,860,704	1.6422	31.52

As Table 4-5 shows, with the complete search, the execution time and the number of combinations increase drastically as the number of VNFs for the NFP increases, and for $Y \geq 6$, we could not complete the execution anymore. However, the heuristic search is able to find a near-optimal solution even for an NFP with 20 different VNFs. As shown in Table 4-4, the NFP with 20 VNFs includes 121 (active) VNF instances in total and can be considered a large NFP.

Figure 4-10 and Figure 4-11 compare the SDTs and the costs. The results are almost the same for the two implementations for cases in which the complete search was able to find the optimal solution. The SDT with the heuristic search differs on average by 0.52% from the SDT with the

complete search with a standard deviation of 0.81, while the cost calculated with the heuristic search differs on average by 0.28% from the cost calculated with the complete search with a standard deviation of 0.28.

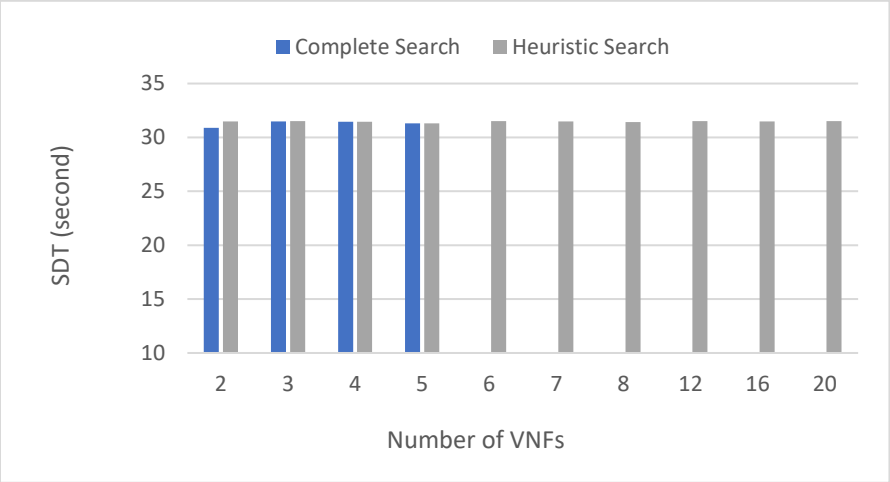


Figure 4-10: Optimal and near-optimal SDT comparison

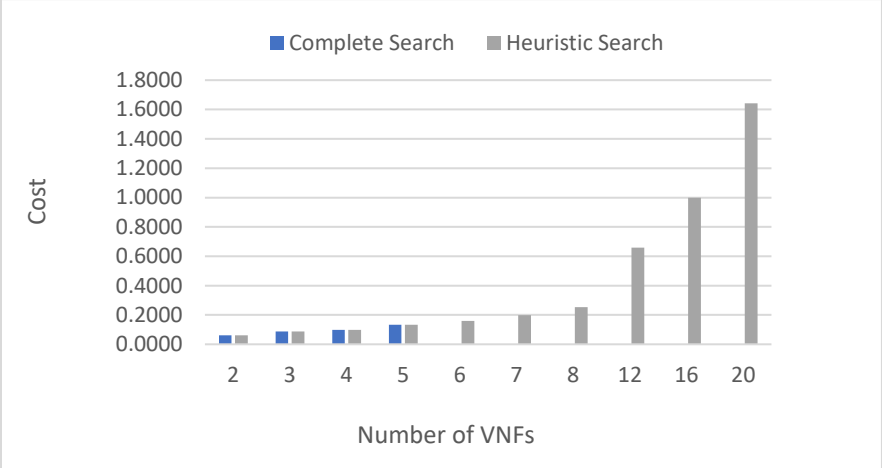


Figure 4-11: Optimal and near-optimal cost comparison

In addition, Figure 4-10 shows that the heuristic search meets the ASDT requirement (i.e., ASDT = 31.536 s per year) for all NFPs of different sizes. Also, Figure 4-11 shows that the networking cost increases as the number of VNFs increases because more VNFs should checkpoint their states and send health-check messages. This was also an expected result according to equation (74). It is noteworthy that although the cost growth due to the VNF number increase is inevitable, our approach minimizes the cost as much as possible.

4.9 Summary and Conclusion

In this chapter, we defined the service disruption time and the service data disruption for an NS functionality. Based on these definitions and the definition of availability, we stated the problem of service availability and continuity requirements satisfaction for NSs. Through our analysis, we showed that although redundancy is necessary to meet service availability and continuity requirements, it is not enough. Therefore, we explored parameters affecting the outage time and service disruption of VNFs and NSs.

We proposed an approach that includes a method of mapping the required availability, acceptable service disruption time, and acceptable service data disruption expressed by a tenant for different functionalities of the NS to configuration parameters. This method guarantees meeting these tenant requirements and minimizes the cost of networking and computing resources at the same time. This mapping method relies on several other methods we proposed to calculate the availability and failure rate of VNF instances, availability and service disruption of VNFs, and NFPs availability, service disruption, and resource cost. The proposed approach takes the availability characteristics of the infrastructure resources and VNF applications for a given NsDF. Then it determines the optimal health-check and checkpointing intervals of VNFs, calculates the required redundancy of VNFs and

VLs, and selects the optimal network and host options for the VNFs of the NS. This approach also considers the elasticity of services in the NFV environment. This approach generates the configuration parameters considering the worst-case scenarios for VNFs placement, VNFs failures, and VLs availability. Thus, it guarantees the satisfaction of the service availability and continuity requirements of the NS on the given infrastructure and for all the scaling levels of the VNFs and the NS.

In this chapter, we also evaluated the time complexity of the complete search, which is used to determine the optimal health-check and checkpointing intervals and minimize the networking cost. We showed that the complete search has an exponential time complexity in terms of the number of VNFs of the NS. Then, we proposed a heuristic search to make the approach applicable for large NSs. The heuristic search finds a near-optimal solution and reduces the time complexity to quadratic time. To compare the accuracy and the time complexity of the complete and the heuristic search, we conducted experiments with an implementation of the design-time approach with these two search strategies. The experiments showed that the proposed heuristic search reduces the execution time significantly while its outputs are very close to those generated using the complete search. Thus, making our method applicable for NSs with large numbers of VNFs.

Chapter 5

Runtime Adaptation Approach

In this chapter, we first define the problem we aim to solve. Next, we present our proposed adaptation framework for adjusting NSs at runtime to maintain the fulfillment of service availability and continuity requirements. We also present an ML model creation method that is used within the adaptation framework.

5.1 Problem Definition

5.1.1 Need for Runtime Adaptation

As discussed in Chapter 4, the service availability and continuity of an NS depend (partially) on the availability and failure rate of the utilized resources (e.g., hosts, storage, and network) and the bandwidth and latency of the available networking options for VNFs checkpointing. When the NS is designed to fulfill given service availability and continuity requirements, it can do so on instantiation as long as these availability characteristics of the resources (*availability constraints*) hold at runtime.

However, the resources provided to VNFs and VLs can change at runtime due to multiple reasons, such as hardware/software upgrades, migration, and failovers. Although the NFVI is expected to respect the availability constraints when a planned change (e.g., upgrade) is performed, unpredictable changes may cause violation of the availability constraints at runtime. For example,

the NFVI usually includes multiple types/options of hosts and networks that VNFs and VLs can utilize [83]. At runtime, the type of the host selected by the design-time approach for a VNF may change if the VNF fails and gets respawned on a different type of host (e.g., if all computing resources of the selected type are already consumed).

In addition, resource availability and failure rate are estimated at design time, which means the actual availability and/or failure rate of resources may differ at runtime. For example, the failure rate of a host changes over time and increases as the host ages. Therefore, the design-time configuration generation process uses the estimate of an average value for host failure rate for the NS lifetime.

If the actual availability or failure rate of some resources at runtime differs from the estimated value at design time, a violation of the availability constraints occurs. Therefore, the NS may no longer fulfill its availability and/or continuity requirements at runtime with the configuration generated at design time. Then, the NS should be adapted by adjusting its configuration parameters to compensate for availability constraints violations.

Resource changes can also improve the availability of VNFs and/or VLs. For example, a failover to resources of better characteristics may happen, or no failure may occur for a long time, reducing the failure rate of resources below the estimated rate. In these cases, we may be able to adjust the NS configuration so that the resource cost is decreased (e.g., reduce the number of standby instances) while still fulfilling the service availability and continuity requirements.

Therefore, we need a runtime adaptation solution to reconfigure and adjust the NS during its lifetime if:

- A change at runtime causes an availability constraint violation. In this case, the goal of the adjustment is to compensate for these changes so that the service availability and continuity requirements are always met.
- A change at runtime improves the availability of resources. In this case, we adjust the NS to reduce resource costs (if possible).

5.1.2 Determining Runtime Adjustments

The analytical methods of the design-time approach can be used to determine the new values for the configurable parameters of the NS when a resource characteristic changes at runtime. However, the execution time of this approach to determine the optimal configuration values for VNFs may not be tolerable at runtime, particularly for large NSs with stringent availability requirements. As shown previously (see Sub-Section 4.8.4), this time increases as the number of VNFs increases, even using the proposed heuristic (Alg. 4-1). For example, Table 4-5 shows that the execution time to determine the optimal configuration for the largest NS of the experiment is about 10 minutes if we use the heuristic search. Therefore, if we need to adjust the NS as soon as the change is detected (e.g., ultra-high availability is required), the runtime adaptation needs lightweight methods to determine the required adjustments for VNFs.

Calculating the availability and the number of required standby instances for VLs using the analytical method of the design-time approach has constant time complexity. Therefore, this analytical method can be used for runtime adaptation of NSs of any size.

5.2 Runtime Adaptation Framework

The runtime adaptation framework consists of a procedure supported by actors in the NFV reference architecture and an Adaptation Module (AM). In this sub-section, we discuss this procedure, the configurable parameters to adapt at runtime, and propose a placement for the AM in the NFV reference architecture. We also discuss the roles of the actors in the NFV reference architecture with respect to the runtime adaptation.

5.2.1 Configurable Parameters

At runtime, we can adjust the health-check interval, the checkpointing interval, and the number of standby instances to compensate for availability constraint violations or to reduce resource consumption. As in Chapter 4, the assumption is that at the VNF application level, health-checks are performed, and their intervals are configurable. Also, checkpointing is performed at the VNF application level, but the checkpointing intervals may or may not be configurable. In addition, the role of the VNF instances, i.e., whether an instance is active or standby, can be set at the VNF application level, and accordingly, the external VL instances can also be active or standby. The MANO functional blocks are not aware of these application-level parameters and roles.

Although the number of standby instances for the VNFs or VLs is not visible to the MANO, it is aware of the total number of instances (i.e., the summation of all the active and standby instances). The (total) number of instances of VNFs and VLs for each NS scaling level is (pre)defined in the NsDF at design time [9]. At runtime, the number of VNF and VL instances of a running NS instance can change by scaling according to these predefined numbers of the deployed NsDF or by switching the NS instance to another more appropriate NsDF. In either case, the MANO only changes the

number of instances according to the applicable NsDF. The OSS or NFVO can trigger the NS scaling or NsDF change. To provide the total number of instances in an NsDF for each scaling level, the required number of active and standby instances are determined at the NS design time by the NS designer.

For some availability constraint violations, scaling up in the same NsDF can provide the required number of standby instances to compensate for the violation. The purpose of scaling, in this case, is to increase the availability of the NS by adding more standby instances as opposed to increase the number of active instances to increase the performance. For example, let us assume an NS with two VNFs and three scaling levels as shown in Table 5-1. To meet the performance expectations at different scaling levels, the number of required active instances of each VNF for this NS is set in the NsDF, as shown in Table 5-1.

Table 5-1: An NsDF to meet a certain performance expectation

	VNF1	**VNF2**
Scaling Level 1	1	5
Scaling Level 2	2	8
Scaling Level 3	3	10

Furthermore, let us assume that to fulfill an availability expectation, the NS designer has determined the number of required standby instances for each VNF at each scaling level, using the analytical methods of the design-time approach. Therefore, the NsDF is updated (or a new one is created) by adding the required number of standby instances, as shown in Table 5-2.

Table 5-2: The updated NsDF (from Table 5-1) to meet an availability expectation

	VNF1			VNF2		
	Active	Standby	Total	Active	Standby	Total
Scaling Level 1	1	1	2	5	2	7
Scaling Level 2	2	1	3	8	3	11
Scaling Level 3	3	2	5	10	4	14

For this example, if there are no runtime changes and adaptation at any scaling level, the number of active and standby instances are always configured according to Table 5-2. Let us assume that the NS is running at scaling level 2, and an availability constraint violation happens. To compensate for this constraint violation, the AM may determine that the number of standby instances of VNF1 should be increased by one, while it should be increased by two for VNF2. Therefore, in total, VNF1 needs four instances, and VNF2 needs thirteen instances. In this case, scaling up to level 3 can provide the required number of instances (and more) for both VNFs. Note that the role of VNF instances can be set outside of the scope of the NFVO.

There are also cases where predefined scaling levels of an NsDF cannot provide the required number of instances at runtime to compensate for availability constraints violations. For example, consider the NS of the previous example at scaling level 2 again. Assume an availability constraint violation happens, and the AM determines that four more standby instances should be added to VNF2. So, in total, fifteen instances of VNF2 are needed. The current NsDF does not support this. This could also be the case where the NS is in the highest scaling level, and we need to add more standby instances, but there is no higher scaling level to switch to. For these cases, different NsDFs can be designed at design time and onboarded for the given NS. These NsDFs differ in the number of required (standby) instances for the same scaling levels of the NS and the underlying availability constraints implied towards the virtual resources. For instance, the NS designer may create one more

NsDFs for the previous example, as shown in Table 5-3. Note that switching to this NsDF would also solve the previous case. In fact, creating multiple NsDFs is a better solution since different NsDFs can be created to support all the possible changes at runtime with the exact required number of instances without resource overprovisioning.

For a given NS, the designer can provide multiple NsDFs to fulfill the same service availability and continuity requirements, if needed. The NsDFs will have different numbers of (standby) instances for the same scaling levels as illustrated in Table 5-2 and Table 5-3 by considering various possible infrastructure resource characteristics. In other words, by considering different availability constraints.

Table 5-3: The second NsDF for resources with lower availability

	VNF1			**VNF2**		
	Active	Standby	Total	Active	Standby	Total
Scaling Level 1	1	2	3	5	5	10
Scaling Level 2	2	2	4	8	6	14
Scaling Level 3	3	4	7	10	8	18

Runtime adjustment may also be possible to avoid overprovisioning resources while meeting the same service availability and continuity requirements. Particularly when resources were assigned to compensate for availability constraints violations, and they turn into excess as no failure occurs for a prolonged time, thus, improving the availability of resources. For example, assume that the NS of the previous example was switched to the NsDF of Table 5-3 to compensate for a change. Subsequent change(s) improve(s) the NS availability so that switching back to the original NsDF of Table 5-2 can fulfill again the requirements.

5.2.2 Runtime Adaptation Procedure

To perform the runtime adaptation, we need to perform the following steps:

- **Step 1**: Monitor changes (including resource changes, scaling, and failures) and (potential) violations of availability constraints.
- **Step 2**: Notify the AM about a change or (potential) violation.
- **Step 3**: Determine new values for adjustable configuration parameters if there was a change or there was no failure for a long period. Note that new values may be the same as current values, meaning, there is no need for adjustment.
- **Step 4**: Perform the reconfiguration as needed.

Monitoring of changes and events is usually performed continuously across the whole system in the NFV architecture. However, MANO does not monitor availability constraints violations since this is not a functionality required by the current specifications. On the other hand, the AM is aware of the availability constraints and can determine whether a reported change causes any violation.

Therefore, any detected change/state change of the resources should be reported to the AM. If a change is reported of there was no change (including failures) for a period, the AM evaluates the service availability and continuity of the NS. Then, the AM can determine the required adjustments and apply the reconfiguration. To determine the new values for these configurable parameters when a resource characteristic changes at runtime, the AM can use the analytical methods proposed for the design-time approach while taking into account the current values for the infrastructure resource characteristics. In this chapter, we also suggest other lightweight methods to determine the runtime adjustments.

5.2.3 Runtime Adaptation Module Placement

As mentioned earlier, the AM can use the analytical methods of the design-time approach to determine the required adjustments. Therefore, it needs the same inputs as the design-time approach, such as the service availability and continuity requirements to be fulfilled, the current availability and failure rate of different types of resources and VNFC applications. In addition, the AM should know the number of active and standby instances of VNFs and VLs at different NS scaling levels.

To determine a good placement for the AM in the NFV reference architecture, we should first define what this AM should be capable of. Steps 1 and 2 of the adaptation procedure (see Sub-Section 5.2.2) are performed by the entities of the NFV architecture. To perform steps 3 and 4 of the adaptation procedure, the AM should be able to perform the following activities.

- **Activity 1**: Analyze the NS availability and service disruption and determine any required adjustments, whether there is an availability constraint violation or reducing resource consumption is possible.
- **Activity 2**: Request for/perform NS scaling/NsDF change operations as needed. To do so, the AM needs to be aware of the number of active and standby instances for each scaling level for each NsDF. This information is generated at design time and can be given to the AM at NS deployment.
- **Activity 3**: Request for/perform health-check and/or checkpointing interval reconfiguration if needed. The application-level configuration of VNFs can be performed through their EMs, and the MANO is unaware of these configurations.
- **Activity 4**: Configure active and standby roles of VNFs and VLs if needed. Like Activity 3, role assignments can be performed through EMs.

The first and the second activities are at the NS level. Therefore, the NFVO and the OSS are suitable candidates to perform these two activities. Between the NFVO and the OSS, the OSS is a better candidate since the OSS is aware of the application-level configurations of VNFs and has direct access to the EMs according to the NFV reference architecture. Therefore, we consider the AM as part of or placed within the OSS.

5.2.4 Notification and Adaptation Operation Flows

Considering the first two steps of the four-step runtime adaptation procedure (see Sub-Section 5.2.2), the first step - monitoring changes in the NFVI resources - can be performed by the VIM since it is responsible for managing the resources provided to VNFs and VLs. If a change is detected by the VIM at the hardware or virtualization layers, it can also determine the impacted virtual resources (i.e., VLs or the virtual resources assigned to VNFs).

For the second step, we need a notification flow from the VIM to the AM placed as part of the OSS. For changes that impact virtual resources assigned to VNFs (e.g., VMs and VLs interconnecting VNF components within a VNF instance), the notification flow is depicted in Figure 5-1. In this case, the VIM reports the change to the corresponding VNFM by sending a notification if the VNFM has subscribed for such notifications. This notification includes the impacted resource(s) and the change. Based on the impacted resource(s), the VNFM should identify the VNF(s) impacted by the change and report them together with the change to the NFVO by sending a notification. Then, the NFVO can determine the impacted NS(s) and report them and the change to the AM, which in turn determines for each impacted NS if adjustments are necessary.

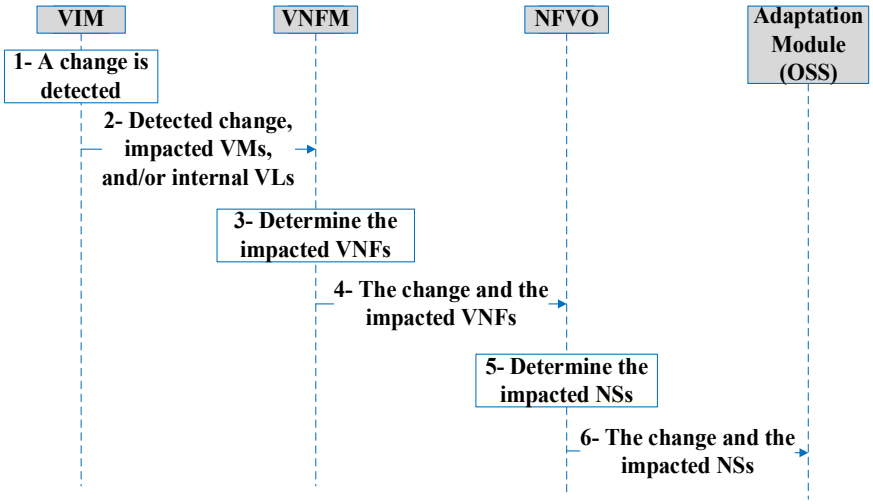


Figure 5-1: Notification flow for changes that impact VNFs

Figure 5-2 shows the notification flow for changes that impact VLs at the NS level. In this case, the VIM reports the change and the impacted VLs directly to the NFVO, which in turn determines the NS(s) impacted and notifies the AM.

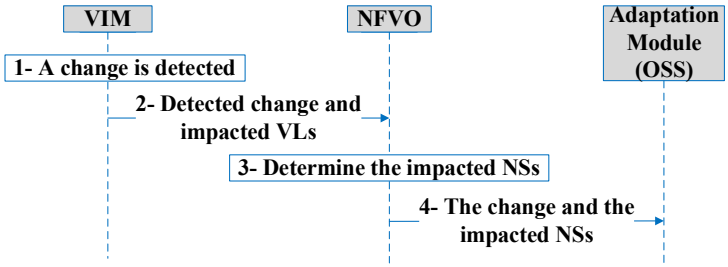


Figure 5-2: Notification flow for changes that impact NS-level VLs

Failures of VNFs and their internal VLs can also be monitored and reported by the EM directly to the AM. Also, hardware resource failures can be monitored by the OSS itself.

In any case, the AM collects the information about failures and estimates the actual failure rate of infrastructure resources, VNFs, and VLs. Thus, at any point in time, the AM is able to evaluate whether the availability of an NS and/or its constituents is unchanged, deteriorated, or improved.

Once the AM detects a change, it determines the applicable new values for the adjustable configuration parameters according to Step 3 of the adaptation procedure. It compares these new values with current ones to identify any required adjustments. Then, according to Step 4 of the adaptation procedure, if changing the scaling level or the NsDF is needed, the AM sends a request to the NFVO, as shown in Figure 5-3. The NFVO applies the requested scaling level and/or deployment flavor and reports the successful change to the OSS. Based on the new scaling level and/or NsDF, new VNF and/or VL instances may be instantiated, which should have a standby role. Accordingly, the AM requests the EM of each VNF to assign the standby role to the newly instantiated VNFs and/or their newly instantiated external VLs. Once the roles are assigned, the AM may request from the EMs to reconfigure the HI and CpI of their managed VNFs.

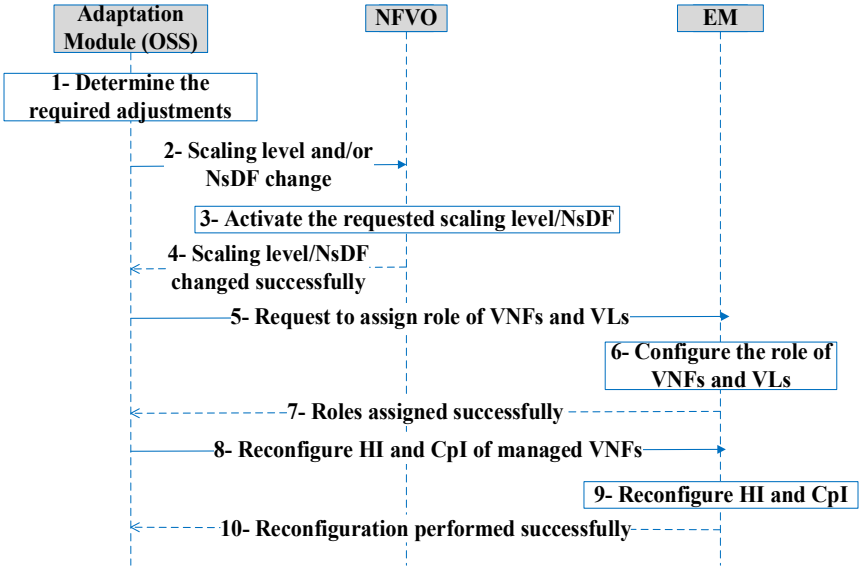


Figure 5-3: Steps for configuration parameters adjustment

Figure 5-4 shows the notification and adjustment paths in the NFV reference architecture. Green arrows indicate the path for change (including failure) notifications initiated by the VIM up to the AM. Red arrows show the cases where the OSS monitors the hardware resource failures and the EM monitors VNF and VL failures. Arrows in blue show the communications paths for the adjustment of configuration parameters.

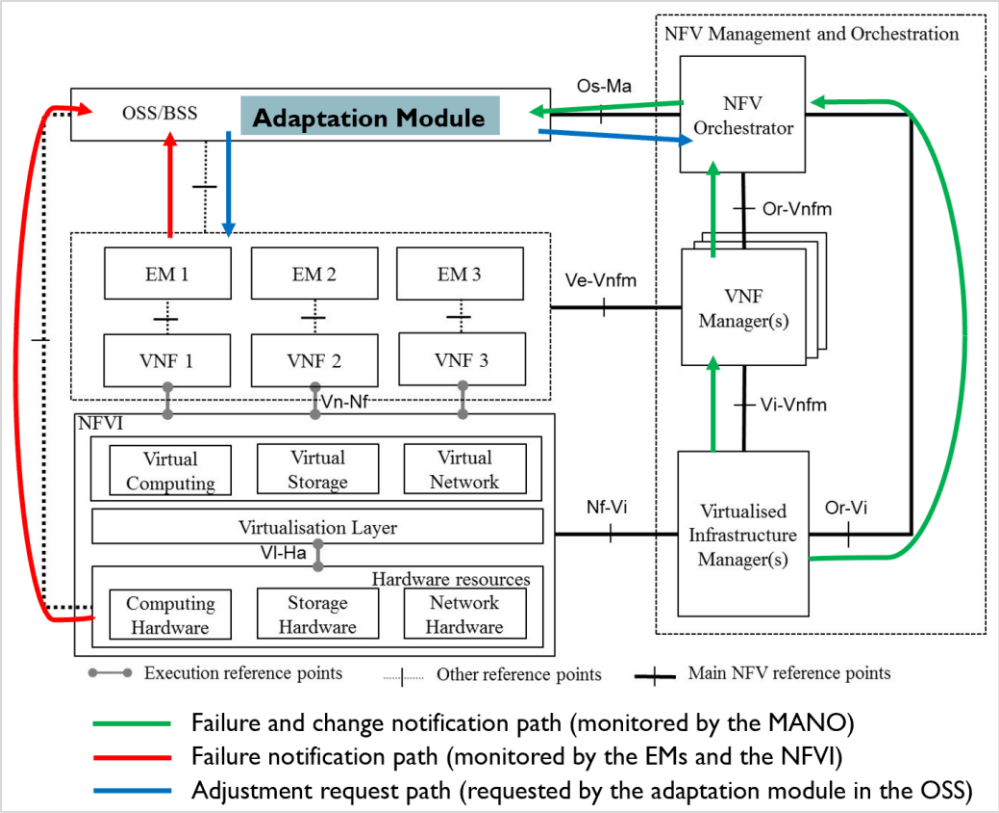


Figure 5-4: Notification and adjustment flows in the NFV reference architecture

5.2.5 Compliance with the Standards

The communication and messaging between different entities of the NFV architecture are performed through the reference points shown in Figure 5-4. Resource monitoring, in general, is

considered in the ETSI NFVI specifications to be supported by the MANO, and the required information elements for monitoring are also proposed in the specifications. So, the MANO can monitor the resource changes and report them to the OSS. To do so, the VIM monitors the resource changes and notifies the NFVO through the *Or-Vi* reference point and the VNFM through the *Vi-Vnfm* reference point if they have subscribed for a notification [84, 85]. The information element that carries the information of change notification is *VirtualisedResourceChangeNotification*. This information element can report a state change, e.g., failure or upcoming upgrade.

To receive the change notifications impacting a VL, the NFVO should subscribe to the VIM. The *virtualisedResourceId* and the *virtualisedResourceGroupId* attributes of *VirtualisedResourceChangeNotification* information element indicate the impacted VL, and the *changedResourceData* attribute of this information element provides the change information. In case of a change impacting a VNF, the VIM notifies the VNFM (which should have subscribed for) about the impacted virtual resources and the change. Similarly, the VIM can use the same information element through the *Vi-Vnfm* reference point. Then, the VNFM determines the impacted VNFs and notifies the NFVO about the impacted VNFs and the change if the NFVO has subscribed for them. The *AlarmNotification* information element can be used to send an alarm from the VNFM to the NFVO [86]. This information element has only one attribute (i.e., *alarm*), and this attribute involves multiple sub-attributes, including the impacted VNF instance ID (i.e., *managedObjectId*) and alarm details (i.e., *faultDetails*). When the VNFM or the VIM notifies the NFVO, it determines the impacted NS(s) and notifies the AM, which is part of the OSS, if the OSS has subscribed for alarms of the impacted NS(s). Similarly, the NFVO can use the *AlarmNotification* information element through the *Os-Ma* reference point [87].

Sending a request from the OSS to the NFVO to change NS scaling level or switch the NsDF is supported by the specifications as part of the NS lifecycle management [87]. The information element which can be used for the NsDF switch is *ChangeNsFlavourData*. The communication between the OSS and EMs is not within the scope of the NFV specifications. However, VNF configurations requested by the AM are part of the activities supported by the OSS and EMs [14].

5.3 Machine Learning Models for Runtime Adaptation

For large NSs, at runtime we need to determine the required adjustments as quickly as possible after an availability constraint violation. One good solution to replace the time-complex algorithms/methods of the design-time approach is creating ML models to determine VNFs adjustments. The AM can use a combination of a lightweight analytical method for VLs redundancy adjustment and ML model(s) to determine the required adjustments for VNFs. Using an ML model to make a prediction has constant time complexity, and usually, it takes only a few milliseconds.

In the following sub-sections, we discuss what ML approach can be used for our problem. Then, we present a model creation method for a case study.

5.3.1 Problem Formulation

The first step of applying ML to networking problems is formulating the problem, which means selecting the ML approach and algorithm that can solve the problem [24]. We need ML models to predict numerical values (for VNFs configurations). The models should be able to determine the adjustments at any point of the NS lifetime. Different ML approaches are introduced in Chapter 2. Unsupervised ML cannot be used since the nature of our problem is not pattern recognition or data structuring. Reinforcement ML also cannot be used because it requires a training phase at runtime.

However, we need a solution that can make a good prediction even if a failure happens immediately after the NS is instantiated. Supervised ML can replace the heavy analytical methods of the design-time approach with regression models. Supervised models can be constructed at design time and used to predict numerical values as soon as the NS is instantiated. DNN is one of the most powerful algorithms for solving regression problems and creating DL models [22, 23].

The details of our proposed DL models creation method for runtime adaptation are explained on a case study. So, this case study is introduced first, followed by the description of constructing the models.

5.3.2 Case Study

Figure 5-5 shows a sample NS with three VNFs and four VLs [8]. It has three network forwarding paths (NFP), which can be mapped to three NS functionalities. VNF1 and VNF3 are shared by all NFPs, while VNF2 serves only NFP1. All NFPs share VL1 and VL4. NFP1 and NFP2 share VL2, and VL3 is only used by NFP3.

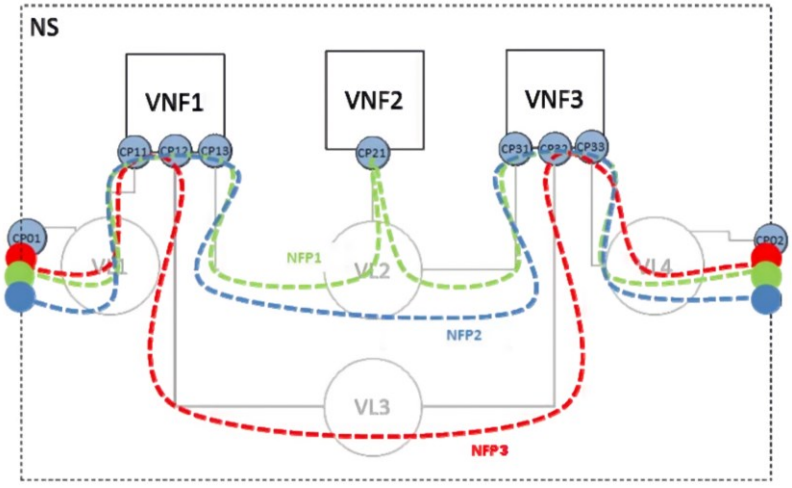


Figure 5-5: Sample NS with three VNFs and four VLs [8]

Figure 5-6 shows the VNFCs and IntVLs for the sample NS of Figure 5-5 [8]. VNF1 and VNF3 each consist of only one VNFC. However, VNF2 has three VNFCs and one IntVL.

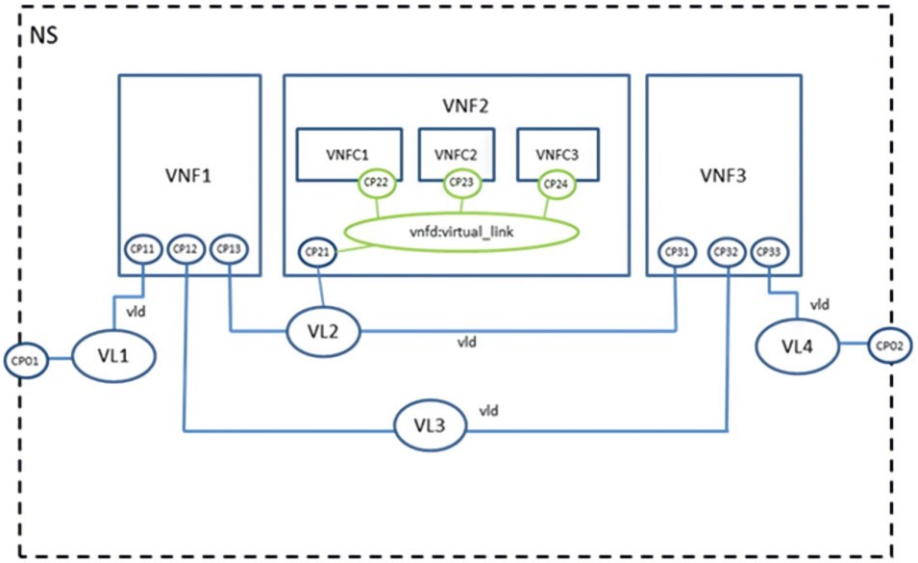


Figure 5-6: VNFCs and IntVLs of the sample NS [8]

We assume this example NS is designed to satisfy the following requirements for each of its functionalities:

Table 5-4: Functional and non-functional requirements of the sample NS

	Maximum service data rate	**Service availability and continuity requirements**
Functionality1 (Mapped to NFP1)	1 Mbps	ASDT = 31.55 seconds per year
Functionality2 (Mapped to NFP2)	10 Mbps	RA = 0.999999 of a year
Functionality3 (Mapped to NFP3)	1 Mbps	RA = 0.99999 of a year ASDD = 1 Mb per failure

This information is available as input for creating the DL models for the runtime adjustments together with details of the NS, including the NFPs, VNFs, VLs, mapping of functionalities to NFPs, NS scaling levels, and the maximum service data rate of each NFP. Some of this information is part of the preliminary NsDF. The NS scaling levels of the NsDF are shown in Table 5-5.

Table 5-5: Sample NS scaling levels

	Number of Instances for VNF1	**Number of Instances for VNF2**	**Number of Instances for VNF3**
Nsl1	1	4	5
Nsl2	2	7	8
Nsl3	3	10	12

Information about the VNFs is available in the form of the VnfDF and characterization of the VNF application and its internal reliability features. As part of the VnfDFs, the VNFCs, IntVLs, and VNF scaling levels of each VNF are as follows:

Table 5-6: VNF scaling levels, VNFCs, and IntVLs of the VNFs of the sample NS

	VNFCs	**IntVLs**	**VNF scaling levels**
VNF1	✓ VNF1-VNFC1	--	✓ VNF1-SL1 ➢ 2 * VNF1-VNFC1 ✓ VNF1-SL2 ➢ 3 * VNF1-VNFC1
VNF2	✓ VNF2-VNFC1 ✓ VNF2-VNFC2 ✓ VNF2-VNFC3	✓ VNF2-IntVL1	✓ VNF2-SL1 ➢ 1 * VNF2-VNFC1 ➢ 2 * VNF2-VNFC2 ➢ 1 * VNF2-VNFC3 ➢ 1* VNF2-IntVL1 ✓ VNF2-SL2 ➢ 2 * VNF2-VNFC1 ➢ 3 * VNF2-VNFC2 ➢ 3 * VNF2-VNFC3 ➢ 1* VNF2-IntVL1
VNF3	✓ VNF3-VNFC1	--	✓ VNF3-SL1 ➢ 2 * VNF3-VNFC1 ✓ VNF3-SL2 ➢ 3 * VNF3-VNFC1

The application-level information of each VNF of the example NS is provided in Table 5-7.

Table 5-7: Application-level information of each VNF of the sample NS

	VNF1	VNF2	VNF3
Minimum health-check interval (ms)	20	50	20
Health-check interval increment step (ms)	20	50	20
Failover time (ms)	10	10	10
Takeover time (ms)	5	10	5
Checkpoint size (bit)	10,240	10,240	20,480
Checkpoint preparation time (ms)	5	15	10
Checkpoint commitment time (ms)	5	10	5
Checkpointing method is synchronous	Yes	Yes	No
Checkpointing interval is constant	Yes	Yes	Yes
Checkpointing interval is configurable	Yes	No	Yes
Minimum checkpointing interval (ms)	20	50	20
Checkpointing interval increment step (ms)	20	N/A	10

For the VNFC applications of each VNF, the availability and Average Failure Rate (AFR) are:

Table 5-8: Availability and failure rate of VNFC applications

	VNFC	Availability	AFR
VNF1	VNF1-VNFC1	0.99999	1
VNF2	VNF2-VNFC1	0.9999	1.5
	VNF2-VNFC2	0.9995	2
	VNF2-VNFC3	0.9995	2
VNF3	VNF3-VNFC1	0.99999	1

The maximum availability and failure rate of VLs (including IntVLs) that the infrastructure can provide are 0.9995 of availability and one failure per year. Finally, the infrastructure available for the deployment is characterized by the different hosting options in Table 5-9 and the various network options in Table 5-10.

Table 5-9: Hosting options available for the sample NS

	Availability	AFR	Cost coefficient
Host option 1	0.999	2	0.85
Host option 2	0.9995	1.5	0.9
Host option 3	0.9999	1	1

Table 5-10: Networking options available for the sample NS

	Latency (ms)	Max bandwidth (bps)
Network option 1	3	20,971,520
Network option 2	10	15,728,640
Network option 3	50	15,728,640

5.3.3 Deep Learning Models for the Sample Network Service

In this sub-section, we illustrate the creation of DL models for the sample NS introduced previously. In the next sub-section, we present a generic method for the creation of these models for any given NS.

5.3.3.1 Training Dataset and Feature Selection

To train a DNN and create the DL model, we need a training dataset. Training data can be collected from a real system or generated synthetically [25]. For our problem, DL models are needed as soon as the NS is instantiated. So, they should be created at design time. Therefore, no data can be collected from a running system since the NS is not deployed yet. However, we can generate the training dataset using our design-time methods. In fact, our purpose of creating DL models is to mimic the behavior of the design-time approach at runtime while taking into account the current characteristics of the resources.

To generate the training dataset, random values can be generated for the constrained characteristics of the infrastructure, that is, for the host availability and failure rates, the VL availability, and the network latency and bandwidth. Then, for each set of values (i.e., input features value), the corresponding adjustable configuration parameters values (i.e., outputs/labels) are determined using the design-time approach to meet the service availability and continuity requirements.

Once we have generated the training dataset, we should select the data features and preprocess the data. Considering the input and output parameters of the design-time approach, the data structure shown in Table 5-11 can be considered to determine the HI, the CpI, and the number of standby instances (SB) for the VNFs of the example NS. These compose the output part of the data structure. Input features considered for the training data structure are the NS scaling level, the AFR of each VNF, the Network Latency (NL), and the Bandwidth (BW) of the network used for checkpointing for each VNF. We do not include the availability of VNFs in this table because availability and AFR convey the same information. The availability decreases if the AFR increases and vice versa. Having correlated features does not improve the training and only slows it down. In addition, all other input parameters of the design-time approach are constant at runtime (e.g., the service availability and continuity requirements), and there is no need to include them in the input features. As an example, one record of the training data is shown in Table 5-11.

Table 5-11: Structure of the training data to determine HI, CpI, and SB together

Features (Input)										**Output**								
NS Level	VNF1 AFR	VNF2 AFR	VNF3 AFR	VNF1 NL	VNF2 NL	VNF3 NL	VNF1 BW	VNF2 BW	VNF3 BW	VNF1 HI	VNF2 HI	VNF3 HI	VNF1 CpI	VNF2 CpI	VNF3 CpI	VNF1 SB	VNF2 SB	VNF3 SB
Nsl1	1.85	12.5	3	3	50	50	22,649,242	15,728,640	15,728,640	660	600	1,780	700	50	490	2	3	1

As shown previously in the flowchart of Figure 4-9, to determine the number of standby instances of a VNF, the HI of the VNF is first determined. Then, the VnfEA is calculated using the VNFs failure rate and the HI. Finally, the number of standby instances is determined using the VnfEA. In short, the number of standby instances of a VNF is determined based on the VNFs failure rate and HI. But in the label structure of Table 5-11, both the HI and the SB are output parameters; therefore, their dependency might not be learned properly by a DL model.

This logic of the analytical methods of the design-time approach is better reflected by constructing two DL models. One model can determine the HI and CpI parameters using the data structure of Table 5-12. Then, a second DL can determine the SB for each VNF using the data structure shown in Table 5-13, where the input features are the AFRs of the VNFs together with the HI values determined by the first DNN model. At runtime, these two DL models can be chained through the HI values produced by the first model, which are used as input by the second model.

Table 5-12: Training data structure for the first DL to determine the HI and CpI values

Features (Input)										**Output**					
NS Level	VNF1 AFR	VNF2 AFR	VNF3 AFR	VNF1 NL	VNF2 NL	VNF3 NL	VNF1 BW	VNF2 BW	VNF3 BW	VNF1 HI	VNF2 HI	VNF3 HI	VNF1 CpI	VNF2 CpI	VNF3 CpI
Nsl1	1.85	12.5	3	3	50	50	22,649,242	15,728,640	15,728,640	660	600	1,780	700	50	490

Table 5-13: Training data structure for the second DL to determine SB values

Features (Input)							**Output**		
NS Level	VNF1 AFR	VNF2 AFR	VNF3 AFR	VNF1 HI	VNF2 HI	VNF3 HI	VNF1 SB	VNF2 SB	VNF3 SB
Nsl1	1.85	12.5	3	660	600	1,780	2	3	1

We expect that applying the domain knowledge and creating two DL models will result in more accurate predictions since, this way, DL models learn the behavior of the design-time approach

better. We examine this in the following sub-section. Therefore, we generated a training dataset with 75000 records for the data structure of Table 5-11 and the data structures of Table 5-12 and Table 5-13. In the data generation, the following input feature changes were considered to simulate changes in the infrastructure at runtime:

- Network delay and bandwidth
    - To simulate switching to a different network option for single VNFs (as in case of a single network interface failover) or all the VNFs using given options (as in case of a router failover)
    - To simulate link congestions and router overloads variations in the delay and bandwidth (up to 50 %)
- Host availability and AFR
    - To simulate failover/migration of single VNFCs or all the VNFCs to another type
    - To simulate the change of AFR and availability (up to one 9 of availability) of single VNFCs or all the VNFCs using the same host type
- VL availability and AFR (up to one 9 of availability) of VLs and IntVLs

For each input data, a random number of features were selected and changed according to the above ranges. Once all the labels were generated using the design-time approach, the dataset was pre-processed according to the general methodology, including encoding categorical data and data scaling and normalization [23]. Then, all duplicates were removed to ensure no overlap between the training and the validation sets. Finally, 10% of the remaining data was set aside for the model validation.

5.3.3.2 Models Training and Validation

We used TensorFlow to create DL models [21]. TensorFlow is a free and open-source machine learning library widely used for DL model construction [21]. First, we determined the hyper-parameters value to train the DNNs. The goal of determining the hyper-parameters value is to achieve fast learning while avoiding overfitting and underfitting [24]. We used the random search method to determine the hyper-parameters value [88]. As a result, for DNNs of both data structures introduced in the previous sub-section, we had:

- Number of hidden layers = 19
- Number of nodes for each hidden layer = 35
- Learning rate = 0.00003
- Regularization rate = 0.005
- Batch size = 256

The number of nodes in the output layer equals the output parameters of the data structure of each DL model. In the case of the single-DL model, it is three times the number of VNFs in the NS. In the case of the chained DL models, it is twice the number of VNFs for the first DL and the number of VNFs in the NS for the second DL. The activation function selected for hidden layers was the Rectified Linear Unit (ReLU) function, and the output layer used a linear function since the problem is a regression [21]. The loss function was the mean squared error, while the optimizer is the ADAM (adaptive moment estimation) algorithm [21].

We implemented a prototype of the chained-DL models in Python using the TensorFlow library. The prototype was trained with the training portion (90%) of the dataset running for 20,000 epochs. The trained prototype was checked using the validation portion (10%) of the dataset.

Table 5-14 shows the standard deviation for each output parameter predicted by the prototype and the respective output value in the validation set (i.e., the optimal value determined by the analytical models). Considering the average value of each output parameter in Table 5-14, the corresponding standard deviation indicates a good prediction. Also, this table shows that DL models could predict VNF2_CpI and VNF3_SB values with 100% accuracy. This usually happens if a parameter's value is constant for all the records of the training (and validation) dataset. In this example, the CpI of the VNF2 is not configurable (see Table 5-7). So, its value for all the records is 50 ms. Also, the number of standby instances for VNF1 for all the dataset records is the same (i.e., one instance) since the design time approach generated the same labels for different input values.

Table 5-14: Validation result of using chained DLs to predict new configuration values

Output Parameter	Average	Standard Deviation
VNF1_HI	1071.332	63.486
VNF1_CpI	882.395	26.925
VNF2_HI	698.098	35.447
VNF2_CpI	50.000	0.000
VNF3_HI	2564.516	119.640
VNF3_CpI	487.007	10.394
VNF1_SB	2.072	0.111
VNF2_SB	3.040	0.170
VNF3_SB	1.000	0.000

To compare the two proposed training data structures, we also generated a dataset with the structure of Table 5-11 (i.e., a single DL model) and implemented a prototype using TensorFlow. Again, 75 000 records were generated, 90% of which were used for training and 10% for validation.

The DNN was trained for 20,000 epochs. The validation results for the number of standbys are shown in Table 5-15. As shown in this table, while the average values are the same for the two solutions, the standard deviations are lower (better) for the chained models, as expected. Therefore, in general, the single DL is more likely to result in service availability and continuity unsatisfaction and/or more resource costs.

Table 5-15: Validation results for single and chained DLs

	Average value	Standard deviation	
		for single DNN	for chained DNNs
VNF1	2.071905	0.35640833	0.11144095
VNF2	3.0402668	0.41133732	0.1699092
VNF3	1.0	0.13673876	0.0

5.3.4 A Generic Method for Deep Learning Models Construction

For each NS instance, specific runtime adjustment models are necessary as the models depend on the NS design, the given infrastructure, and the service availability and continuity requirements. The runtime adjustment models include two DL models for adjusting the configuration of the VNFs if swift adaptation is required for a large NS. Otherwise, analytical models can be used instead.

In addition, runtime adjustment needs an analytical model for the redundancy adjustment of VLs. Note that calculating the number of required standbys for VLs using the method of the design-time approach has constant time complexity. Therefore, there is no need for an additional ML model to replace it.

In the previous section, we discussed creating DL models for a sample NS. The following steps can be followed to construct DL models for any given NS design to be deployed on a given infrastructure to satisfy the given tenant's availability and service continuity requirements.

1. Identify the input parameters (features) of the analytical methods that can change at runtime and the range within which they can change.
    - Input parameters are NS scaling level and AFR, NL, and BW of VNFs. But Some of them may not change during runtime. For example, the bandwidth of a network link can be guaranteed for the runtime (e.g., a dedicated physical network is used for the link).
    - If an input parameter does not change during runtime, its value is the same for all the records of the training dataset. Therefore, it can be removed from the training data structure.
2. Determine the configurable output parameters.
    - Output parameters are HI, CpI, and SB of VNFs, but CpI may not be configurable for a VNF.
    - Unconfigurable output parameters should not be included in the training data structure.
3. Generate two sets of input parameters with values randomly changing within their range.
    - The first set includes the NS scaling level and AFR, NL, and BW of VNFs.
    - The second set includes the NS scaling level and AFR and HI of VNFs.
4. Apply the generated sets of input parameters to their respective analytical methods to generate the corresponding output parameters of the analytical models. In other words, generate the labels to create the training set of each DL model.
    - For the first set of input parameters, HI and CpI of VNFs are generated.
    - For the second set of input parameters, SB for VNFs is generated.

- If the value of an output parameter is the same for all the records, it is better to be removed from the data structure.

5. Preprocess the training sets (e.g., scaling, normalization, etc.) and eliminate duplicates.
6. Split each set into training and validation sets.
    - Usually, 80-90% of the records are used for training, and the remaining 10-20% are used for validation.
7. Determine the DNNs architecture (the number of hidden layers and their nodes).
    - The random search method can be used.
8. Determine other hyper-parameters values, including the learning rate, regularization rate, and batch size.
    - The random search method can be used to determine the values.
    - The activation function for hidden layers is the Rectified Linear Unit (ReLU) function
    - The output layer uses a linear function.
    - The loss function is the mean squared error
    - The optimizer is the ADAM (adaptive moment estimation).
9. Train the two DNNs by the training datasets and save the respective DL models.
    - If there is a divergence/very slow convergence while training a model, meaning the learning loss during training is increasing/very slowly decreasing, the parameters in steps 6 and 7 need to be tuned.
10. Apply the validation datasets to the respective trained DL models to validate the models.
    - The result of a model is unsuccessful if the model predictions diverge unacceptably from the values of output parameters in the validation dataset.

- In this case, the models need to be changed/refined, starting with changing the parameter determined in steps 6 and 7.

5.4 Summary and Conclusion

In this chapter, we proposed a runtime adaptation framework for adjusting NS configuration at runtime to maintain the satisfaction of the service availability and continuity requirements when a change happens in resources used by the NS. This framework includes an adaptation module that is part of the OSS and interacts with other entities of the NFV architecture. It receives change and failure notifications from the MANO and the EM and determines the required adjustments. Then, it requests for VNFs and the NS reconfiguration.

Resource changes may deteriorate the service availability and continuity of the NS at runtime. The runtime adaptation framework reacts to these changes to compensate for availability constraints violations. Also, resource changes may improve the service availability and continuity of the NS at runtime. In these cases, the framework adjusts the NS to reduce resource consumption (if possible) while fulfilling the service availability and continuity requirements.

A runtime adaptation procedure supported by change notification and adaptation operation flows was also proposed in this chapter. We showed how functional blocks of the MANO, the EM, and the OSS could work together to notify the adaptation module about the changes and failures and reconfigure the NS. We showed that the proposed framework complies with the ETSI NFV specifications.

In this chapter, we also proposed a method for constructing machine learning models for runtime adaptation. The adaptation module can use these models to determine the required

adjustments at runtime instead of the analytical methods, especially for large NSs. We showed that deep learning among others is the machine learning approach which can solve this problem. We also proposed the data structure for training datasets through an analysis using a sample NS and a prototype for model construction.

In the next chapter, we present a proof of concept, including prototypes of our design-time and runtime approaches and their application in an NFV cloud environment to demonstrate the feasibility of our solutions. We also present the result of multiple experiments performed with two case studies and different service availability and continuity requirements.

Chapter 6

Prototypes, Testbed, and Experiments

We implemented prototypes of our approaches and performed experiments with two case studies to demonstrate the feasibility of our solutions. In this chapter, we first introduce the objectives of the experiments. We next present our prototypes, the case studies and the testbed prepared for the experiments. We also present the test scenarios and discuss the results of the experiments.

6.1 Objectives of Experiments

The main objectives of our experiments are:

1. Demonstrating the feasibility of the proposed solutions.
2. Evaluating whether the service availability and continuity requirements are satisfied for the case studies when the design-time approach configures the NS and the availability characteristics of resources at runtime do not deteriorate from values were estimated at design time.
3. Evaluating whether the same service availability and continuity requirements for the NS are fulfilled at runtime when availability constraint violations happen, and the runtime adaptation approach is used to compensate for those violations.

As we will explain in this chapter, to evaluate the feasibility of our proposed solutions, we have prepared prototypes of our approaches, built case studies (i.e., NSs), and created an NFV cloud environment to perform experiments. To evaluate the fulfillment of the service availability and continuity requirements of NSs using our design-time and runtime approaches, we have conducted multiple experiments to compare the actual availability and service disruption of deployed NSs in a given period with their required service availability and acceptable service disruption.

6.2 Prototypes

6.2.1 Design-time Approach

We developed a prototype of the design-time approach in Java. This prototype reads the input parameters from a MySQL database. Input parameters include the following data:

- NS information
    - List of the VNFs and VLs of the NS
    - NS scaling levels
    - NS functionality mappings to NFPs
    - The service data rate of NFPs
    - VNFs and VLs of each NFP
- Tenant requirements
    - RA, ASDT, and/or ASDD for each NS functionality
    - Period for which the requirements should be met
- VNFs information
    - List of the VNFCs and IntVLs of VNFs

    - VNFs scaling levels
    - VNFC applications availability and failure rate
    - Allowed degree of host sharing for VNFC instances (derived from anti-affinity rules)
    - Checkpointing method of each VNF (if stateful)
    - Application-level property of each VNF (minimum HI, FoT, ToT, minimum CpI, checkpoint message size, checkpoint preparation time, and checkpoint commitment time)
- VLs availability
- Infrastructure information
    - Availability, failure rate, and cost of each host type
    - Latency and bandwidth of each network option (for VNFs checkpointing)

Figure 6-1 shows the main classes of the prototype. The properties of different classes of this figure represent the abovementioned input parameters. According to Figure 6-1, an NS references one or multiple NFPs, VNFs, and scaling levels. It also references zero or many VLs and one infrastructure (which the NS will use). An NFP references one or multiple VNFs and zero or multiple VLs. A VNF or a VL may be referenced by one or more NFPs. A VNF references one or multiple VNFCs, checkpointing methods, VNF scaling levels, and zero or more VLs (i.e., IntVLs). The infrastructure references one or more host types and network options.

Methods of the design-time approach are implemented as the methods of different classes of Figure 6-1, including calculating the availability and failure rate of VNF instances, determining the availability and service disruption of VNFs and NFPs, and finding the optimal configuration for the

NS. The outputs of these methods are stored in the MySQL database, including the optimal HI, CpI, host type, and network option for VNFs, as well as the number of standby instances for VNFs and VLs.

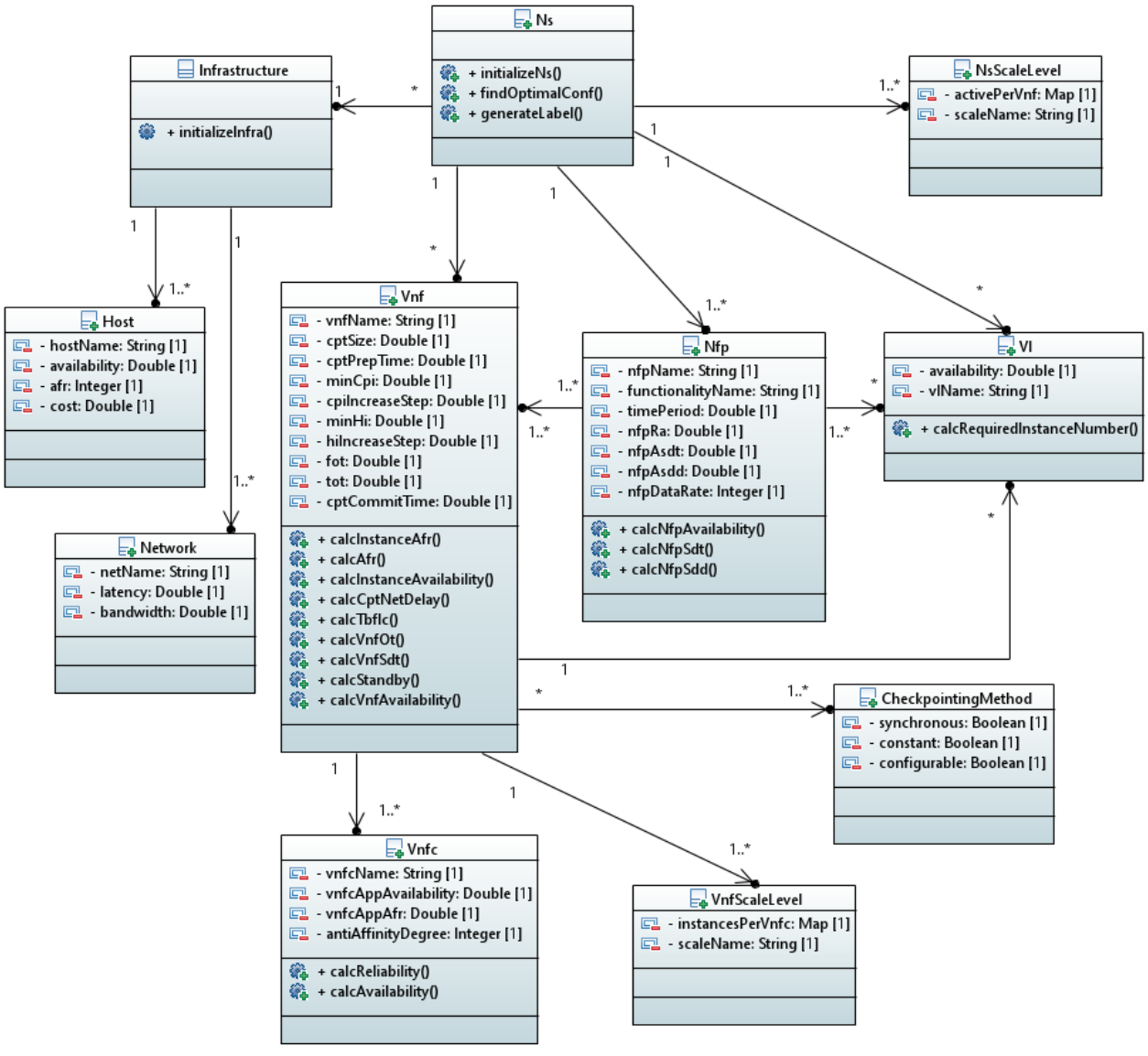


Figure 6-1: Main classes of the design-time approach prototype

As discussed in Chapter 5, a data generator is also needed at design time to create datasets for training DL models of NSs. Therefore, a method is added to the NS class of the prototype of the design-time approach (generateLabel() method in Figure 6-1). This method generates different input values randomly for possible changes described in Sub-Section 5.3.3.1. Then, it reinitializes the NS and the infrastructure with these input values and determines the value of the corresponding output parameters (i.e., the optimal configuration).

Each set of input parameters includes random values for the availability and failure rate of hosts, VNFCs, VLs, and IntVLs, and the bandwidth and latency of the network for the VNFs checkpointing. The outputs are new values for HI and CpI of VNFs and the number of standby instances for VNFs and VLs. The dataset generator stores each input set and its corresponding outputs in the MySQL database.

6.2.2 Runtime Approach

Runtime approach prototype includes a program to create DL models and an implementation of the AM.

6.2.2.1 Deep Learning Models Construction

The DL models are created at design time, but the purpose is to use them at runtime by the AM. Therefore, the model creation is considered as part of the runtime approach. We have developed a prototype of our proposed method for chained-DL model construction in Python. The TensorFlow library is used in this prototype to train and save the models [21]. The random search method has determined the hyper-parameters value [88]. For the DNNs of models of tested case studies, the

following hyper-parameters value resulted in a good learning rate and loss decrease during training (as mentioned also in Sub-Section 5.3.3.2):

- Number of hidden layers = 19
- Number of nodes for each hidden layer = 35
- Learning rate = 0.00003
- Regularization rate = 0.005
- Batch size = 256

The number of nodes in the input and output layers of the models is determined separately for each experiment since they depend on the NS, as explained in Sub-Section 5.3.3.2. For this prototype, the activation function for hidden layers is the Rectified Linear Unit (ReLU) function, and the linear function is used for the output layer [21]. The loss function is the mean squared error, and the optimizer is the ADAM (adaptive moment estimation) algorithm [21].

DL models should be trained and constructed separately for each experiment since they are specific to the NS, the availability and acceptable service disruption requirements, and the availability characteristics of the infrastructure. For each experiment, the chained models are trained with 90% of the dataset and validated with the remaining 10%.

6.2.2.2 Adaptation Module

Another prerequisite for the runtime adaptation is the AM. We have developed a prototype of the AM in Python. In summary, this prototype performs the followings:

- Receive failure and change notifications from the MANO and EM

- Re-evaluate the availability and service disruption of the VNFs and the NS
    - Some methods of the design-time approach are used for this purpose
- Determine the required adjustments
    - DL models predict the adjustments
- Request for NS reconfiguration if needed
    - Requests the MANO to scale the NS
    - Requests the EM to reconfigure HI and CpI of the VNFs and configure the role of VNFs and VLs instances

The AM communicates with the MANO through RESTful APIs. Also, the AM offers RESTful APIs to the EM and the MANO so that they can send failure and change notifications to the AM.

6.3 Case Studies

This sub-section introduces two NSs prepared for our experiments

6.3.1 Video Streaming Case Study

6.3.1.1 Network Service

This NS provides a video streaming functionality. Figure 6-2 shows this NS, which has two VNFs and one VL. The decoder VNF decodes a video and sends the stream to the multi-caster VNF. The end users have to connect to a multi-caster IP address to receive the stream.

Figure 6-2: A video streaming NS

We assume this NS has two scaling levels to satisfy specific workloads. Table 6-1 shows the required (active) instances of VNFs for each NS scaling level.

Table 6-1: Scaling levels of the video streaming NS

	Decoder VNF	**Multi-caster VNF**
NS Level 1	1	2
NS Level 2	2	3

6.3.1.1 Tenant Requirement

For this case study, we chose ASDT for the type of tenant requirement. We consider two different values of ASDT for two sets of tests. The value of the first ASDT is 120 seconds, i.e., the service disruption time of the NS for each test should remain below 120 seconds. The value of the second ASDT is 180 seconds. The period of each test (i.e., the NS lifetime) is 24 hours. For this requirement, the goal of our experiments is to compare the actual SDT of the NS to the ASDT after each test run.

6.3.1.2 Virtual Network Functions

We developed the decoder and multi-caster VNFs. Both have one VNFC and zero IntVL. Also, each VNF has one VNF scaling level with one VNFC instance. We created VM images for the

VNFC of both VNFs, which use Ubuntu Server 20.04 as their operating system. Ubuntu is a distribution of Linux available at [89].

The decoder VNF uses FFmpeg software to provide its functionality. FFmpeg is an open-source software to record, convert, and stream video and audio [90]. FFmpeg uses multiple communication protocols to send the video stream to one or more predefined receivers. In our experiments, the Real-Time Messaging Protocol (RTMP) is used to send the stream by FFmpeg. Each instance of the decoder VNF can send the stream to one or two multi-caster VNFs. Note that FFmpeg does not send the stream directly to the end users because the list of end users is not predefined and can change anytime.

The multi-caster VNF uses Nginx software to make the video stream available to the end users. Nginx is an open-source software used as a (generic) proxy server [91]. We configure the Nginx to receive a stream and multicast it in its IP address using RTMP. End users can use a video player software that supports RTMP (e.g., VLC media player) to connect to an instance of the multi-caster VNF and receive the video stream.

To add the capability of N+M redundancy model to both VNFs, we developed a Python program called HA (High Availability). All instances of each VNF run this program, which is responsible for:

- Health-check messaging between instances of the same VNF
    - The HA program uses client/server TCP sockets to implement this functionality. Each instance sends a health-check message to all other instances of the VNF

every HI (Health-check Interval). The value of HI is stored in a configuration file and is reconfigurable at runtime.

- Assigning active or standby roles to VNF instances
    - The number of active (N) and standby (M) instances of the VNF is stored in a configuration file and is reconfigurable at runtime. When a VNF instance is instantiated or restarted, it has the standby role by default. Then, its role may change by the HA program running on the same instance, according to the value of N in the configuration file and the current number of active instances. The HA program creates and updates a list of active and standby instances. It also checks the value of N and M every HI and acts accordingly if they change.
- Failover
    - If standby instances do not receive a health-check message from an active instance for *HI+timeout*, they consider it failed. Then, the HA program running on the standby instance with the lowest IP address changes the role of the instance to active.
- Assigning a virtual IP to each active instance of the multi-caster VNF and removing it when an instance becomes standby
    - The IP address of the active instances of the multi-caster VNF should be known and accessible to the external network (e.g., for the end users). The primary IP address of each VNF instance may change during the NS lifetime (e.g., due to a failover). But a virtual IP is assigned as the secondary IP address to an active VNF instance, and it is removed when the instance becomes standby so that

another active instance may use this IP (if a standby instance becomes active). The external network knows the list of virtual IPs.

The decoder VNF also needs to support checkpointing mechanism since it is stateful. We have developed a Python program for this purpose. This program:

- Uses cline/server TCP sockets to send the checkpoint from active instances to standby instances
- Stores the checkpoint in a file when a standby instance receives it
- Restores the latest checkpoint if a standby instance becomes active

The checkpointing program reads the FFmpeg state (state of video decoding) every CpI (Checkpointing Interval) and sends it to the standby instances. The value of CpI is stored in a configuration file and is reconfigurable at runtime. The multi-caster VNF is not stateful and does not need checkpointing.

Table 6-2 shows the application-level information of the decoder and multi-caster VNFs for this case study.

Table 6-2: Application-level information of VNFs of the video streaming NS

	Decoder VNF	**Multi-caster VNF**
Minimum health-check interval (ms)	500	500
Health-check interval increment step (ms)	200	200
Failover time (ms)	500	100
Takeover time (ms)	3000	2000
Checkpoint size (bit)	1024	N/A
Checkpoint preparation time (ms)	50	N/A
Checkpoint commitment time (ms)	5	N/A
Checkpointing method is synchronous	Yes	N/A
Checkpointing interval is constant	Yes	N/A
Checkpointing interval is configurable	Yes	N/A
Minimum checkpointing interval (ms)	1100	N/A
Checkpointing interval increment step (ms)	550	N/A

For the VNFC applications of each VNF, the availability and AFR are:

Table 6-3: Availability and AFR of VNFC applications for video streaming NS

	VNFC	Availability	AFR
Decoder VNF	VNFC1	0.99995	0.5
Multi-caster VNF	VNFC1	0.99995	0.5

6.3.1.3 Availability of Hosts and Networks

The infrastructure available for our experiments provides one type of host and one network option. We use dedicated physical hosts to provide resources to the VNFs. We can impose a different number of failures on the hosts for different test runs to change the availability and failure rate of VNFs for each experiment. We assume that the estimated availability of the hosts at design time is 0.9999, and their AFR is 2 per 24 hours.

The physical network of the infrastructure is not dedicated to our experiments, and intentional failure is not permitted. Therefore, we assume that the availability of the network is 100%.

6.3.1.4 Deployment Configuration of the Network Service

We have given the ASDT requirements and input parameters to the design time approach to determine the optimal configuration for the NS instantiation. For the set of tests in which the requirement is 120 seconds of ASDT, the optimal configuration is shown in Table 6-4.

Table 6-4: Optimal configuration for ASDT=120s

	NS Scaling Level	HI (ms)	CpI (ms)	SB
Decoder VNF	NS Level1	5100	5500	1
	NS Level 2	7700	7700	1
Multi-caster VNF	NS Level1	8700	N/A	1
	NS Level 2	11700	N/A	2

For the set of tests where the requirement is ASDT=180s, the optimal configuration is:

Table 6-5: Optimal configuration for ASDT=180s

	NS Scaling Level	HI (ms)	CpI (ms)	SB
Decoder VNF	NS Level1	8100	8800	1
	NS Level 2	12300	11000	1
Multi-caster VNF	NS Level1	17900	N/A	1
	NS Level 2	21100	N/A	1

We also created a new NsDF, which has three NS scaling levels compared to the scaling levels of the original NsDF (see Table 6-1). Table 6-6 shows the number of VNF instances for each NS scaling level of the new NsDF for the set of tests that ASDT=120s. Table 6-7 shows similar information when ASDT=180s.

Table 6-6: NS scaling levels of the new NsDF when ASDT=120s

	Decoder VNF			Multi-caster VNF		
	Active	Standby	Total	Active	Standby	Total
Scaling Level 1	1	1	2	2	1	3
Scaling Level 2	2	1	3	3	2	5
Scaling Level 3	2	2	4	3	3	6

Table 6-7: NS scaling levels of the new NsDF when ASDT=180s

	Decoder VNF			Multi-caster VNF		
	Active	Standby	Total	Active	Standby	Total
Scaling Level 1	1	1	2	2	1	3
Scaling Level 2	2	1	3	3	1	4
Scaling Level 3	2	2	4	3	2	5

The NS is instantiated using the new NsDF to support the required number of standby instances of VNFs during the NS lifetime.

6.3.2 Web Service Case Study

6.3.2.1 Network Service

The second case study is a web-based ad-post network service. This NS provides a web interface for users to post their ads or search for published items on the website. Figure 6-3 shows this NS, which has two VNFs and one VL. The webserver VNF provides the web interface, stores the posted ad data in the database VNF, and searches for items in the database if a search request is received from an end user.

Figure 6-3: A web service NS

Table 6-8 shows our assumption for the number of scaling levels of the NS to satisfy a certain performance. This table also indicates the number of (active) instances of each VNF for each NS scaling level.

Table 6-8: Scaling levels of the web service NS

	Database VNF	Webserver VNF
NS Level 1	1	2
NS Level 2	2	3

6.3.2.2 *Tenant Requirement*

To experiment with a different type of tenant requirement, we consider satisfying RA for the second case study. We assume two different values of RA for two different sets of experiments: *RA=0.9995* and *RA=0.999*. The period of each individual test for this case study is 24 hours. The goal is to compare the actual availability of the NS to the RA after each test. Note that service continuity is not part of the tenant requirements for this case study.

6.3.2.3 *Virtual Network Functions*

Similar to the first case study, we developed both VNFs of this case study and each has one VNFC and zero IntVL. Also, each VNF has one VNF scaling level with one instance of the VNFC. Ubuntu Server 20.04 is used to create a VM image for the VNFC of each VNF.

The database VNF uses MariaDB, a free and open-source relational database management software [92]. The webserver VNF uses Apache software to provide its functionality [93]. The web interface is developed in PHP and is accessible using a web browser.

To add the capability of N+M redundancy model to both VNFs, we use the same Python program (i.e., HA) introduced in Sub-Section 6.3.1.2. The only difference is that in the case of web service NS, the HA program assigns virtual IPs to the active instances of both VNFs. Because the (secondary) IP address of the database VNF instances should be predefined for the webserver VNF instances, and the external network should know the (secondary) IP address of the webserver VNF instances.

Each database VNF instance can serve one or two webserver VNF instances, and each webserver VNF instance has access to one database VNF. When there are multiple instances of the

database VNF, they need to synchronize their records in the MariaDB database. Galera is used to create a cluster of MariaDB databases to support data synchronization between multiple instances of the database VNF [94].

Table 6-9 shows the application-level information of the database and webserver VNFs for this case study. Since ASDT or ASDD is not part of the tenant requirements, checkpointing characteristic details of the VNFs are not used for the experiments. Therefore, they are not included in Table 6-9.

Table 6-9: Application-level information of VNFs of the web service NS

	Database VNF	**Webserver VNF**
Minimum health-check interval (ms)	500	500
Health-check interval increment step (ms)	200	200
Failover time (ms)	100	100
Takeover time (ms)	100	100

For the VNFC applications of each VNF, the availability and AFR are:

Table 6-10: Availability and AFR of VNFC applications for web service NS

	VNFC	**Availability**	**AFR**
Database VNF	VNFC1	0.99995	0.5
Webserver VNF	VNFC1	0.9999	1

6.3.2.4 Availability of Hosts and Networks

For this case study, we use the same infrastructure used for the first case study. Therefore, we have one type of host and one network option. Hosts are dedicated to our experiments, and we assume that at design time, their estimated availability is 0.9999, and their AFR is 2. The network is not dedicated to our experiments. Therefore, we cannot impose failures on it and assume the availability of the network is 100%.

6.3.2.5 *Deployment Configuration of the Network Service*

We have given the requirement and input parameters to the design time approach to determine the optimal configuration for the NS instantiation. For the set of tests in which the requirement is RA=0.9995, the optimal configuration is shown in Table 6-11.

Table 6-11: Optimal configuration for RA=0.9995

	NS Scaling Level	HI (ms)	SB
Database VNF	NS Level1	3500	1
	NS Level 2	4300	1
Webserver VNF	NS Level1	6500	1
	NS Level 2	8300	2

For the set of tests where the requirement is RA=0.999, the optimal configuration is:

Table 6-12: Optimal configuration for RA=0.999

	NS Scaling Level	HI (ms)	SB
Database VNF	NS Level1	6100	1
	NS Level 2	8900	1
Webserver VNF	NS Level1	11300	1
	NS Level 2	17100	1

We also created a new NsDF with three NS scaling levels for the set of tests that RA=0.9995. Table 6-13 shows the number of VNF instances for each NS scaling level of the new NsDF. For the set of tests that RA=0.999, no more scaling levels needed to be added. However, the original NsDF should be updated to add the required number of standby instances (see Table 6-12) to each NS scaling level. Table 6-14 shows the number of VNF instances for this case.

Table 6-13: NS scaling levels for the new NsDF when RA=0.9995

	Database VNF			Webserver VNF		
	Active	Standby	Total	Active	Standby	Total
Scaling Level 1	1	1	2	2	1	3
Scaling Level 2	2	1	3	3	2	5
Scaling Level 3	2	2	4	3	3	6

Table 6-14: NS scaling levels for the updated NsDF when RA=0.999

	Database VNF			Webserver VNF		
	Active	Standby	Total	Active	Standby	Total
Scaling Level 1	1	1	2	2	1	3
Scaling Level 2	2	1	3	3	1	4

6.4 Testbed

To perform validation experiments, we need a testbed to instantiate and manage the NSs. The testbed includes an EM, an NFV cloud, a fault injector, and log collectors. In this sub-section, we introduce these components.

6.4.1 Element Manager

An EM is needed to configure VNF applications after instantiation and reconfigure them at runtime. Therefore, we created an EM which is a VNF itself and has one VNFC. Ubuntu Server 20.04 is used to create a VM image for this VNFC, and the EM application is prototyped in Python. The EM can (re)configure HI, CpI, and the number of active (N) and standby (M) instances stored in the configuration file of VNF instances. Each VNF instance checks for the changes in the content of this configuration file periodically and acts accordingly. The interval of this check is one HI by default; however, it is reconfigurable. When the value of N or M is changed, it can cause changing the role of VNF instances. The EM provides web APIs to the AM to be called for VNFs

(re)configuration. It also monitors VNF failures and notifies the AM. The monitoring capability of the EM is based on Zabbix, an open-source software for monitoring network nodes, servers, and applications [95].

6.4.2 Fault Injector

To test the runtime adaptation framework, we need to change the failure rate of hosts and VNFs during runtime. A fault injector program is developed in Python, which can impose a predefined or random number of failures on hosts and VNFs. The fault injector reboots the operating system of a host to fail the host, or the operating system of the VNFC of a VNF instance to fail the VNF instance.

6.4.3 Local Monitor and Log Collector

The video streaming NS includes a decoder VNF and a multi-caster VNF. This NS continuously plays a video for the test period (24 hours) except when one of its VNFs functionality encounters a period of outage. A local monitoring program and a log collector are developed for each VNF of the NS. The local monitoring program on the VNFC of a decoder VNF instance notifies the local log collector on the VNFC of the same VNF instance when the FFmpeg application starts or stops decoding and streaming the video. The log collector writes these events into a local file. We can download the log file at any time and analyze the logs. The local monitoring program on the VNFC of a multi-caster VNF instance monitors the Nginx service and notifies the local log collector if it starts or stops.

Similar local monitoring program and local log collector are used for the VNFs of the web service NS. In this case, the program monitors Apache service for webserver VNF instances and MariaDB service for database VNF instances. We have also developed a traffic generator for this

case study which sends HTTP requests to the webserver VNF (with configurable intervals) to post ads or search for items. The traffic generator also logs the success or failure of each request it sends.

The local monitoring program on the VNFC of a VNF instance also notifies the local log collector when the operating system of the VNFC is rebooted.

6.4.4 NFV Cloud

We use OpenStack and Tacker to create a real NFV environment for our experiments. Considering the ETSI NFV architecture, OpenStack can play the role of a VIM and manage the NFVI. Tacker is a software module that adds the VNFM and the NFVO functionalities to the OpenStack controller. The OpenStack modules used for our implementation are:

- **Keystone**: it is the identification, authentication, and authorization manager.
- **Neutron**: it is the network manager to create and manages virtual links.
- **Glance**: it manages a store for VM images for VNF components.
- **Nova**: it manages the life cycle of VM instances.
- **Placement**: it provides a resource inventory and helps Nova with instance scheduling and resource optimization.
- **Heat**: it enables composite cloud applications orchestration using a declarative template.
- **Ceilometer**: it is a monitoring service to collect, normalize and transform event data produced by other OpenStack services.
- **Aodh**: it triggers actions based on rules against event data collected by Ceilometer.
- **Gnocchi**: it provides a scalable means of storing data collected by Ceilometer.
- **Mistral**: it is a workflow manager and manages (defines and executes) multi-step tasks.

- **Barbican**: it is an encryption key management service.

Figure 6-4 shows the video streaming NS instantiated on our testbed created using OpenStack and Tacker. As shown in this figure, the NFVI is deployed on four hardware servers (HW SRV), the MANO runs on an additional hardware server, while the AM and the fault injector run on a personal computer (PC) as part of the OSS.

The Yoga version of Tacker implements features of the NFVO and the VNFM described in the ETSI NFV specifications for Release 2. Therefore, it is suitable to test our approaches and check if all these modules work together for the intended purposes.

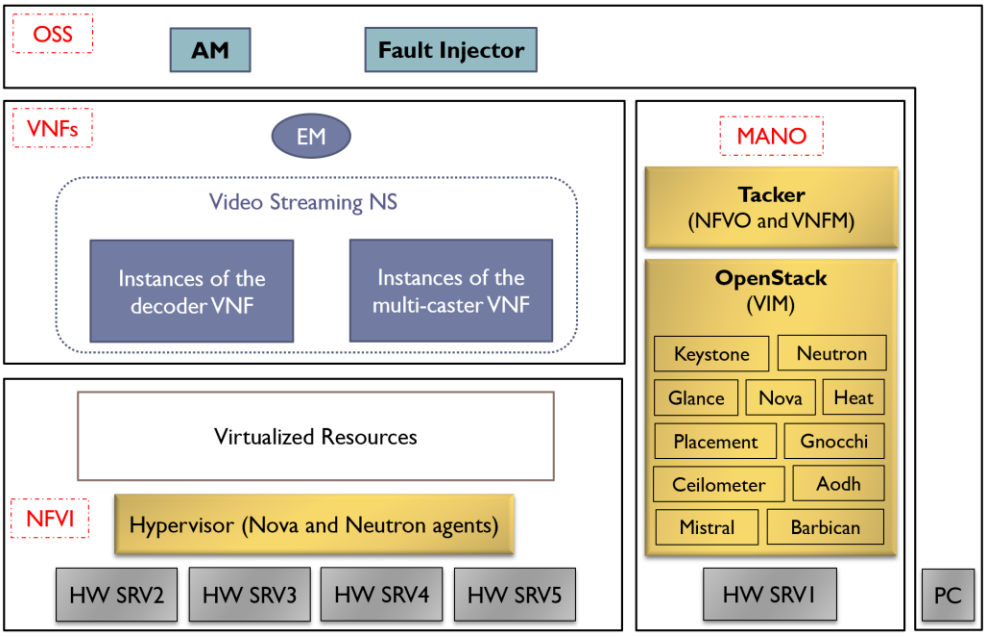


Figure 6-4: NFV cloud realized using OpenStack and Tacker

6.5 Limitations of Performed Experiments

Although the main ideas of the proposed approaches are tested through different experiments, we faced some limitations in performing our tests as explained below.

6.5.1 Capacity and Performance of Resources

The hardware servers available for our experiments have four CPU cores each. Therefore, each host can serve up to three VNF instances because each VNF instance and the host operating system need at least one CPU core to provide the minimum performance. Also, four hardware servers are available to host VNF instances. As a result, the maximum number of VNF instances for both VNFs of each NS can be twelve. Thus, we cannot perform experiments with the existing infrastructure resources for large NS instances.

The limited capacity of the hosts also constrains the performance of the VNFs. For example, an active instance of the decoder VNF consumes almost 100% of one CPU core to decode the video. This VNF instance also needs to execute health-checking and checkpointing, which are CPU intensive for very low HI and CpI. Therefore, we cannot consider stringent values for service availability and continuity requirements for our tests.

Despite the abovementioned limitations, it is worth mentioning that we have performed a diverse and a good number of experiments to assess our approaches. As we will show in this chapter, the results confirm the feasibility and applicability of our proposed solutions.

6.5.2 Tacker and OpenStack Limitations

The latest version of Tacker and OpenStack (i.e., Yoga) is used to implement the MANO functional blocks. This version of Tacker implements features of the NFVO and the VNFM described in Release 2 of ETSI NFV specifications. However, Release 4 of the specifications is currently published.

The Yoga version of Tacker does not support NsDF change at runtime for a running NS. As explained in Sub-Section 5.2.1, the runtime adaptation framework can change the number of standby instances of VNFs by scaling the NS, changing the NsDF, or both at runtime. The Tacker supports remote calls for NS scaling, but no Tacker API exists for NsDF change at runtime.

6.6 Test Scenarios and Experiments Results

As discussed earlier, we have two case studies, each with a different type of tenant requirement. Also, we have two different values for each tenant requirement. We performed twenty tests in total to cover all cases. Each test run lasted for 24 hours. Table 6-15 shows the number of experiments for each requirement value and type for each case study.

Table 6-15: Experiments for each requirement and case study

	Video Streaming		**Web Service**	
Requirement Type	**ASDT**		**RA**	
Requirement Value	120s	180s	0.9995	0.999
Number of Tests for Design-time Approach	1	1	1	1
Number of Tests for Runtime Approach	4	4	4	4

To test the design-time approach, we keep the failure rate of resources at runtime equal to the value used to determine the optimal configuration at design time. Therefore, performing one test per

requirement value is enough since the test condition will be the same if we repeat the test. To test the runtime approach, we impose a random number of failures on the VNFs at runtime for each test run. Therefore, we repeat the test for each requirement value four times to have a better picture of the results.

In the rest of this sub-section, we introduce the test scenarios and present the results of the experiments.

6.6.1 Test Scenarios

6.6.1.1 Design-time Approach

To test the design-time approach, we follow these steps for each test run:

1. Determine the optimal configuration for the NS
2. Update the NsDF (if needed) to include the total number of VNF instances (i.e., active and standby instances)
3. Create a supplementary artifact for the AM that contains the optimal configuration (i.e., HI, CpI, and the number of active and standby instances of VNFs for NS scaling levels)
4. Deploy the updated NsDF and instantiate the NS using Tacker and OpenStack
5. Using the AM, initialize the NS with the optimal configuration
6. Run the NS and record the start time using the AM
7. Impose failures to VNFs using the Fault Injector (the failure rates at runtime are equal to values expected at design time)
8. Collect logs for service outages and disruptions
9. Stop the NS after 24 hours using the AM (each test period is 24 hours)

10. According to the tenant requirement, calculate the actual availability or service disruption of the NS using the logs
11. Evaluate the fulfillment of the tenant requirement

6.6.1.2 Runtime Adaptation Approach

To test the runtime time approach, we follow these steps for each test run:

1. Determine the optimal configuration for the NS
2. Generate training datasets for possible runtime changes
3. Create DL models for runtime adaptation (to be used by the AM) using our model creation method
4. Update the NsDF (if needed) to add the number of standby instances and additional NS scaling level(s) to the original NsDF
5. Create a supplementary artifact for the AM that contains the optimal configuration (i.e., HI, CpI, and the number of active and standby instances of VNFs for NS scaling levels)
6. Deploy the updated NsDF and instantiate the NS using Tacker and OpenStack
7. Using the AM, initialize the NS with the optimal configuration
8. Run the NS and record the start time using the AM
9. Impose failures to VNFs by the Fault Injector (random number of failures are imposed for each test run)
10. Monitor the failures by the EM and notify the AM
11. Adapt the NS accordingly by the AM (the AM uses DL models to predict the required adjustments)
12. Collect logs for service outages and disruptions

13. Stop the NS after 24 hours (each test period is 24 hours)
14. According to the tenant requirement, calculate the actual availability or service disruption of the NS using the logs
15. Evaluate the fulfillment of the tenant requirement

6.6.2 Experiments Results and Analysis

In summary, the result of the experiments showed that the tenant requirement was fulfilled for both case studies with different requirement values using our design-time approach. Also, where the failure rate of VNFs increased at runtime, the service outage/disruption per failure was decreased as a result of runtime adaptation framework adjustments. Therefore, the tenant's availability and continuity requirements were satisfied for all the test runs.

The details of experiments for each case study are explained in the rest of this sub-section.

6.6.2.1 Video Streaming Case Study

Table 6-16 shows the experiments result for five test runs in which the requirement is ASDT=120s. The goal of the first test run (i.e., Test D1) is to assess the design-time approach, while for others (i.e., Test R1 to R4), the goal is to test the runtime approach. Table 6-16 shows the SDT of each failure of each VNF during the test period. Also, the total number of VNF failures, the average SDT for one failure, and the overall SDT of the NS (the summation of the SDTs of the VNFs) for each test run are shown in this table.

Table 6-16: Experiments result for ASDT=120s

		SDT of decoder VNF for each failure (seconds)						SDT of multi-caster VNF for each failure (seconds)						Total number of failures	Average SDT of one failure	Overall SDT
		#1	#2	#3	#4	#5	#6	#1	#2	#3	#4	#5	#6			
Design-time Approach	**Test D1**	18.24	19.18	17.73	-	-	-	14.47	13.92	15.57	-	-	-	6	16.52	**99.11**
Runtime Approach	**Test R1**	13.71	10.64	11.26	10.17	-	-	9.55	8.73	9.12	7.54	7.06	6.03	10	9.38	**93.81**
	Test R2	11.76	14.37	12.97	14.74	-	-	8.56	6.56	7.58	-	-	-	7	10.93	**76.54**
	Test R3	17.21	12.04	12.34	14.51	12.83	-	6.84	5.48	6.09	7.55	-	-	9	10.54	**94.89**
	Test R4	16.38	11.18	8.77	13.90	12.13	13.31	7.73	5.38	8.41	-	-	-	9	10.80	**97.19**

The AFR of each VNF at design time for this case study was about three failures (calculated based on the failure rates of resources and VNFC applications presented in Sub-Section 6.3.1, using method of Sub-Section 4.4.2). For all the test runs, one failure was imposed to one VNF at a time. For the first run (Test D1), we kept the failure rate of VNFs at runtime same as the estimated values at design time. For other test runs (Test R1 to R4), a random number of failures were imposed on VNFs at runtime to assess the result of runtime adaptation approach adjustments.

As shown in Table 6-16, the overall SDT of the NS for all test runs is below the accepted threshold (i.e., ASDT=120s). This table shows that when the failure rate of VNFs increases at runtime, the average SDT of a VNF failure is reduced due to NS adjustment performed by the runtime adaptation framework. For example, the lowest average SDT in this table belongs to Test R1, which has the highest number of VNF failures (10 failures in total for both VNFs).

A VNF failure can happen at any point between consecutive health-check intervals and consecutive checkpointing intervals. As a result, we may have different service outage and disruption times for different failures if we have the same HI and CpI configuration. In the best case,

a failure happens when a health-check message is ready to send and after a checkpoint is sent. In the worst case, a failure occurs after a health-check message is sent and a checkpoint is ready to send. Our approaches guarantee the fulfillment of service availability and continuity requirements for the worst-case scenario. The SDTs in Table 6-16 do not necessarily represent the worst-case service disruption times because, in our experiments, failures happened at different points between consecutive health-check intervals and checkpointing intervals.

However, if all the failures for our experiments happened after a health-check message was sent and before a checkpoint was ready to send, we would have the maximum possible SDT for failures, as shown in Table 6-17. The maximum possible SDT due to a failure is the summation of the *current HI*, *timeout*, *ToT*, *FoT*, and the *current CpI*. Table 6-17 also shows the average value for the maximum possible SDTs and the maximum possible overall SDT for each test run. Therefore, in the worst case, the NS would encounter the maximum overall SDT shown in this table for each test run, and still satisfy the ASDT.

Table 6-17: Maximum Possible SDT of Failures for experiments with ASDT=120s

		Maximum possible SDT for decoder VNF for each failure (seconds)						Maximum possible SDT for multi-caster VNF for each failure (seconds)						Total number of failures	Average max. SDT of one failure	Max. overall SDT
		#1	#2	#3	#4	#5	#6	#1	#2	#3	#4	#5	#6			
Design-time Approach	Test D1	21.90	21.90	21.90	-	-	-	16.80	16.80	16.80	-	-	-	6	19.35	116.10
Runtime Approach	Test R1	14.60	13.00	14.20	12.60	-	-	16.80	10.20	12.40	8.40	9.20	7.60	10	11.90	119.00
	Test R2	14.50	17.55	21.00	23.75	-	-	16.80	10.40	15.20	-	-	-	7	19.03	119.20
	Test R3	21.90	14.40	14.40	17.05	17.00	-	9.65	8.10	8.10	8.10	-	-	9	13.19	118.70
	Test R4	21.90	15.10	9.40	15.70	15.75	15.40	10.00	7.10	9.10	-	-	-	9	13.27	119.45

It is worth mentioning that the maximum overall STD values in Table 6-17 are close to the ASDT value (i.e., 120s) for all the test runs. This is because our approaches minimize the cost while guaranteeing the satisfaction of the requirements for the worst-case scenario. In other words, they determine the *optimal* configuration, as discussed in Chapter 4.

The result of test runs to satisfy ASDT=180s is shown in Table 6-18.

Table 6-18: Experiments result for ASDT=180s

		SDT of decoder VNF for each failure (seconds)								SDT of multi-caster VNF for each failure (seconds)							Total number of failures	Average SDT of one failure	Overall SDT
		#1	#2	#3	#4	#5	#6	#7	#8	#1	#2	#3	#4	#5	#6	#7			
Design-time Approach	Test D1	24.37	23.72	21.99	-	-	-	-	-	18.08	19.37	15.88	-	-	-	-	6	20.58	123.45
Runtime Approach	Test R1	12.96	11.49	11.14	10.30	13.08	11.13	13.27	14.53	13.49	6.43	6.65	6.34	11.06	-	-	13	10.91	141.87
	Test R2	14.88	16.82	23.40	19.22	-	-	-	-	15.17	8.58	9.18	7.18	7.59	-	-	9	13.56	122.02
	Test R3	21.21	14.22	14.32	15.05	12.15	14.72	-	-	6.58	11.20	10.56	-	-	-	-	9	13.33	120.01
	Test R4	15.93	11.44	19.77	11.06	12.40	-	-	-	6.78	8.75	8.85	8.26	10.58	8.36	9.29	12	10.96	131.47

Similarly, the first run evaluates the design-time approach, and the rest are for assessing the runtime adaptation approach. Similar results can be seen in this table. The tenant requirement is fulfilled for all the experiments, the NS adapted to the change in the failure rate of VNFs to adjust their SDT, and the average SDT of a failure is decreased when the number of VNF failures is increased.

Table 6-19 shows the maximum possible SDT for the experiments with the requirement of ASDT=180s. This table shows that in the worst case, the maximum possible overall SDT for each test run remains below the ASDT. In addition, the maximum possible overall SDT of each test run is close to the value of the requirement since our approaches determine the optimal configuration.

Table 6-19: Maximum Possible SDT of Failures for experiments with ASDT=180s

		Maximum possible SDT of decoder VNF for each failure (seconds)								Maximum possible SDT of multi-caster VNF for each failure (seconds)							Total number of failures	Average max. SDT of one failure	Max. overall SDT
		#1	#2	#3	#4	#5	#6	#7	#8	#1	#2	#3	#4	#5	#6	#7			
Design-time Approach	Test D1	29.80	29.80	29.80	-	-	-	-	-	26.20	26.20	26.20	-	-	-	-	6	28.00	168.00
Runtime Approach	Test R1	14.50	13.40	13.20	15.00	17.10	12.90	13.90	16.20	26.20	8.00	8.00	8.20	13.20	-	-	13	13.83	179.80
	Test R2	17.20	20.50	29.15	30.10	-	-	-	-	26.20	15.60	15.40	11.80	12.80	-	-	9	19.86	178.75
	Test R3	29.80	17.25	19.45	20.45	19.80	18.00	-	-	13.40	16.80	18.60	-	-	-	-	9	19.28	173.55
	Test R4	29.80	14.80	24.05	13.55	15.20	-	-	-	8.80	11.20	12.20	12.80	13.40	12.40	11.20	12	14.95	179.40

6.6.2.2 *Web Service Case Study*

For this NS, the tenant requirement is RA=0.9995 for one set of five test runs and RA=0.999 for another set of five test runs. Table 6-20 and Table 6-21 show the results for these two sets of experiments. The first test run of each table tests the design-time approach, and others test the runtime adaptation framework. These tables show that the tenant requirement is satisfied when the failure rates do not change. Also, when the failure rate of VNFs changes at runtime, the adaptation framework adjusts the NS and decreases the average outage time of a VNF failure as the total number of failures increases.

Table 6-20: Experiments result for RA=0.9995

		Outage time of database VNF for each failure (seconds)							Outage time of webserver VNF for each failure (seconds)								Total number of failures	Average outage time of one failure	NS availability
		#1	#2	#3	#4	#5	#6	#7	#1	#2	#3	#4	#5	#6	#7	#8			
Design-time Approach	Test D1	4.5	4.0	5.0	-	-	-	-	8.5	9.0	8.0	-	-	-	-	-	6	6.50	0.999549
Runtime Approach	Test R1	3.0	2.0	1.0	1.5	1.5	2.0	1.5	3.5	3.5	3.0	2.5	2.5	3.0	3.5	3.5	15	2.50	0.999566
	Test R2	3.5	3.0	3.0	2.5	3.5	2.5	-	7.2	3.1	4	3.9	-	-	-	-	10	3.62	0.999581
	Test R3	3.5	2.5	1.0	3.0	3.0	4.0	-	3.5	3.5	3.0	4.0	5.0	-	-	-	11	3.27	0.999583
	Test R4	2.0	1.0	2.0	1.0	-	-	-	7.0	2.0	3.5	2.5	2.5	4.0	4.0	4.5	12	3.00	0.999583

Table 6-21: Experiments result for RA=0.999

		Outage time of database VNF for each failure (seconds)								Outage time of webserver VNF for each failure (seconds)									Total number of failures	Average outage time of one failure	NS availability
		#1	#2	#3	#4	#5	#6	#7	#8	#1	#2	#3	#4	#5	#6	#7	#8	#9			
Design-time Approach	Test D1	8.0	7.0	8.5	-	-	-	-	-	14.5	14.0	15.5	-	-	-	-	-	-	6	11.25	**0.99922**
Runtime Approach	Test R1	5.5	5.0	4.5	5.5	4.5	5.5	-	-	5.5	3.5	9.0	6.0	8.5	-	-	-	-	11	5.73	**0.99927**
	Test R2	3.5	5.5	2.5	1.0	3.0	3.5	3.5	1.0	11.0	3.5	3.5	3.5	3.0	4.0	3.0	4.5	3.5	17	3.71	**0.99927**
	Test R3	5.5	9.0	5.0	6.5	3.0	-	-	-	9.5	12.5	11.0	-	-	-	-	-	-	8	7.75	**0.99928**
	Test R4	4.0	3.0	3.5	1.0	4.0	6.5	1.0	-	3.5	4.0	6.0	3.0	7.0	-	-	-	-	12	3.88	**0.99946**

The maximum possible outage time of each VNF failure for the experiments of this case study is shown in Table 6-22 and Table 6-23. The maximum possible outage time due to a failure is the summation of the *current HI*, *timeout*, *ToT*, and *FoT*. The minimum possible availability of the NS is also shown in Table 6-22 and Table 6-23 for each test run. According to these tables, the optimal configurations fulfill the RA for the worst-case scenario of each test run while minimizing the networking costs.

Table 6-22: Maximum Possible Outage Time of Failure for experiments with RA=0.9995

		Maximum possible outage time of database VNF for each failure (seconds)							Maximum possible outage time of webserver VNF for each failure (seconds)								Total number of failures	Average outage time of one failure	Minimum NS availability
		#1	#2	#3	#4	#5	#6	#7	#1	#2	#3	#4	#5	#6	#7	#8			
Design-time Approach	Test D1	5.0	5.0	5.0	-	-	-	-	9.2	9.2	9.2	-	-	-	-	-	6	7.10	**0.999507**
Runtime Approach	Test R1	5.0	2.4	1.2	1.6	1.6	2.2	1.6	3.8	3.8	3.6	3.0	2.6	3.2	3.6	3.6	15	2.85	**0.999505**
	Test R2	3.8	3.4	3.4	2.8	4.0	3.0	-	9.2	4.0	4.8	4.4	-	-	-	-	10	4.28	**0.999505**
	Test R3	5.0	3.2	1.8	3.4	3.4	4.4	-	3.8	4.2	3.4	4.4	6.0	-	-	-	11	3.91	**0.999502**
	Test R4	2.4	1.2	2.2	1.2	-	-	-	9.2	3.0	3.9	3.0	3.0	4.8	4.4	4.6	12	3.58	**0.999503**

Table 6-23: Maximum Possible Outage Time of Failure for experiments with RA=0.999

		Maximum possible outage time of database VNF for each failure (seconds)								Maximum possible outage time of webserver VNF for each failure (seconds)									Total number of failures	Average outage time of one failure	Minimum NS availability
		#1	#2	#3	#4	#5	#6	#7	#8	#1	#2	#3	#4	#5	#6	#7	#8	#9			
Design-time Approach	Test D1	9.6	9.6	9.6	-	-	-	-	-	17.8	17.8	17.8	-	-	-	-	-	-	6	13.70	0.99905
Runtime Approach	Test R1	9.6	8.0	5.4	7.2	4.8	7.0	-	-	6.4	6.4	11.4	7.0	9.6	-	-	-	-	11	7.53	0.99904
	Test R2	4.8	7.2	3.2	1.4	4.0	4.8	4.6	2.2	17.8	4.2	4.2	4.0	4.6	4.8	4.0	4.8	4.4	17	5.00	0.99902
	Test R3	5.8	10.2	6.2	8.4	3.8	-	-	-	17.8	15.6	16.4	-	-	-	-	-	-	8	10.53	0.99903
	Test R4	9.6	5.4	6.2	1.8	5.8	9.0	3.2	-	7.4	6.0	7.8	5.6	9.4	-	-	-	-	12	6.43	0.99911

It is also noteworthy that the networking overhead of runtime adjustment on the VNFs for the performed experiments was minimal and negligible because each adjustment needed one notification message from the EM to the AM and one adjustment message from the AM to the EM. Note that the health-check and checkpoint messaging are required for any system with service availability and continuity requirements, regardless of what solution they use to fulfill these requirements. Therefore, the monitoring, health-check, and checkpoint messaging overheads are not considered part of our solutions' overhead. In addition, our solutions reduce the HI and CpI (wherever possible) to improve resource efficiency.

6.7 Summary and Conclusion

In this chapter, we discussed the objectives of the experiments performed with prototypes of our design-time and runtime adaptation approaches. This included assessing the feasibility of our solutions in a realistic environment and evaluating the availability and service disruption of different case studies when the NS is configured and managed by our approaches to fulfill availability and acceptable service disruption requirements.

Moreover, we introduced two case studies (i.e., NSs) that we prepared for our experiments. The first case study provides video streaming functionality, while the second one provides web-based ad posting functionality. We also presented different VNFs we developed for each NS to support the NS functionality. To evaluate different tenant requirements satisfaction, we considered two ASDT values for the first case study and two RA values for the second case study to be fulfilled for different test runs.

The testbed created to perform our experiments and demonstrate the feasibility of our solutions was also presented in this chapter. This testbed includes a VNF cloud environment realized by OpenStack and Tacker, which can instantiate and run NSs, including our case studies. An EM was also developed to manage the VNF applications and monitor their failures. In addition, the testbed included a fault injector developed to impose failures on VNFs at runtime and log collectors developed to gather information needed for analyzing the actual availability and service disruption of NSs after each test run.

In total, twenty experiments were conducted (each experiment lasted 24 hours) to evaluate our approaches. For some test runs, there was no change in the failure rate of VNFs at runtime since the goal was to test our design-time approach. For other experiments, the failure rate of VNFs was increased randomly during runtime to test the runtime adaptation approach.

The results of the experiments showed that the tenant requirement was satisfied for all the test runs, and the requirement satisfaction was guaranteed for the worst-case scenario while minimizing the resource cost. In addition, the networking overhead imposed by the runtime adaption framework was minimal. Therefore, the results of the experiments demonstrated, to some extent the validity of our proposed solutions. However, applying our solutions to other NSs of larger sizes in an industrial

environment with lower resource limitations will be necessary to complete the validation of our approaches.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

In this thesis, we devised a solution consisting of design-time and runtime approaches for service availability and continuity management for network services in the context of NFV. We introduced quantitative definitions for service disruption time and service data disruption to provide measurable metrics for service disruption. We proposed a design-time approach to map the service availability and continuity requirements of NSs to constraints on low-level configuration parameters. We also proposed a runtime adaptation approach to adjust NSs at runtime to maintain the satisfaction of the tenant's availability and continuity requirements throughout the lifetime of the NS when the availability and failure rate of resources and VNF applications deteriorate at runtime.

The design-time approach includes a requirement mapping method fed by other analytical methods, which calculate the availability and failure rate of VNF instances, outage time and service disruption of VNF profiles, redundancy of VNFs and VLs, and service outage and disruption of NFPs. This approach takes as input the NS design, the tenant's availability and acceptable service disruption requirements, and the availability characteristics of the VNF applications and the infrastructure resources. The design-time approach determines the optimal values for health-check and checkpointing intervals of VNFs and the redundancy for VNFs and VLs. This approach

minimizes networking and computing costs while determining the optimal configuration. In addition, if there are multiple host types to host VNFs and/or different networking options for VNFs checkpointing, the most appropriate ones are selected to fulfill the service availability and continuity requirements regarding resource costs. The design-time approach guarantees that the service availability and continuity requirements of the NS can be satisfied as long as the availability characteristics of the VNF applications and infrastructure resources are not deteriorated at runtime.

We also analyzed the time complexity of using a complete search method for the design-time approach and showed that it is exponential. Therefore, we proposed a heuristic search method to make our approach affordable. Performed simulations showed that the complete search method cannot be used for NSs with more than six VNFs. In contrast, the proposed heuristic can determine the near-optimal configuration for large NSs with quadratic time complexity.

The runtime approach includes an adaptation framework that collects change and failure events of resources and VNFs at runtime and notifies an adaptation module. The adaptation module re-evaluates the satisfaction of service availability and continuity requirements of the NS using some of the analytical methods of the design-time approach. If needed, the adaptation module determines required adjustments and requests for NS reconfiguration to compensate for any availability constraint violation.

The adaptation module can use the analytical methods of the design-time approach to determine the necessary adjustments at runtime. We showed that the execution time of these methods is not tolerable for large NSs at runtime when quick adjustments are needed, even using our proposed heuristic search. Therefore, we proposed a method to create deep learning models to replace the time-complex analytical methods of the adaptation module. We showed that two chained deep

learning models could be used for this purpose, and we determined the structure of training datasets (i.e., input features and output parameters).

We conducted several experiments with real NSs to demonstrate the feasibility of our proposed solutions and show if they can configure the NSs to fulfill the tenant's availability and continuity requirements. We prototyped the design-time and runtime approaches and developed different VNFs to create two NSs for the experiments. Moreover, we prepared a testbed, including an NFV cloud infrastructure, an element manager, a fault injector, and log collectors. The results of the experiments confirmed that our approaches guarantee the satisfaction of the service availability and continuity requirements for the prepared case studies. They also showed to some extent, the validity of our proposed solutions. However, creating larger NSs in an industrial infrastructure with more available resources will be necessary to confirm the validity of our approaches.

7.2 Future Work

In this thesis, we devised a solution to configure and manage NSs to satisfy service availability and continuity requirements in the presence of service outages. According to the definition of service availability, an NS encounters an outage if its service is not provided. If the service is provided but at a lower capacity than expected due to the failure of some VNF instances (not all instances of the same type), we have service degradation. Future work can target service degradation management of NSs, for example, to provide a solution to keep it below an acceptable threshold.

As mentioned earlier, we assumed that redundant instances of VNFCs, IntVLs, VNFs, and VLs for an NS are placed in the same *Zone*. However, it is possible to spread redundant instances between different *NFVI-PoPs* (NFV Infrastructure Point of Presence) and *ZoneGroups* (multiple zones

grouped together), if available. As potential future work, one can investigate the impact of multi-layer redundancy on the availability of NSs.

In this thesis, we proposed solutions for NSs realized by VM-based VNFs. Recently, containerized VNFs are also gaining attentions in the industry. Currently, there is no MANO implementation to support all the required functionalities to manage containerized VNFs. However, the ETSI NFV specification group has been working on updating the current specifications and providing new ones regarding the containerized VNFs. Therefore, future work can also consider investigating NS availability and service continuity where containerized VNFs realize the NS.

We created real NSs and testbed to perform our experiments. The results of the experiments proved to some extent, the validity of our approaches. But the resource limitations constrained the size of NSs created for the experiments. Validating the solutions proposed in the thesis in an industrial infrastructure that can host larger NSs remains for future work.

Bibliography

[1] ETSI ISG NFV, "ETSI GS NFV 002 V1.2.1: Architectural Framework," December 2014. [Online]. Available: https://docbox.etsi.org/isg/nfv/open/Publications_pdf/Specs-Reports/NFV%20002v1.2.1%20-%20GS%20-%20NFV%20Architectural%20Framework.pdf.

[2] ETSI, "Network Functions Virtualisation (NFV)," [Online]. Available: https://www.etsi.org/technologies/nfv. [Accessed July 2022].

[3] AT&T, "VNF Guidelines for Network Cloud and OpenECOMP," February 2017. [Online]. Available: https://wiki.onap.org/download/attachments/1015849/VNF%20Guidelines%20for%20Network%20Cloud%20and%20OpenECOMP.pdf?api=v2. [Accessed July 2022].

[4] M. Toeroe and F. Tam, Service Availability: Principles and Practice, First ed., John Wiley & Sons, 2012.

[5] ETSI ISG NFV, "ETSI GS NFV-REL 001 V1.1.1:Resiliency Requirements," January 2015. [Online]. Available: http://www.etsi.org/deliver/etsi_gs/NFV-REL/001_099/001/01.01.01_60/gs_NFV-REL001v010101p.pdf.

[6] Huawei Technologies, "Challenges of 5G Ultra Reliability," 2018. [Online]. Available: http://cqr.committees.comsoc.org/files/2018/07/05-Dehan_Li-Challenges-of-5G-Ultra-ReliabilityETR-052618.pdf. [Accessed July 2022].

[7] ETSI ISG NFV, "ETSI GS NFV-REL 003 V1.1.2:Reliability; Report on Models and Features for End-to-End Reliability," July 2016. [Online]. Available: https://www.etsi.org/deliver/etsi_gs/NFV-REL/001_099/003/01.01.01_60/gs_nfv-rel003v010101p.pdf.

[8] ETSI ISG NFV, "ETSI GR NFV-MAN 001 V1.2.1: Report on Management and Orchestration Framework," December 2021. [Online]. Available: https://www.etsi.org/deliver/etsi_gr/NFV-MAN/001_099/001/01.02.01_60/gr_NFV-MAN001v010201p.pdf.

[9] ETSI ISG NFV, "ETSI GS NFV-IFA 014 V4.2.1: Network Service Templates Specification," May 2021. [Online]. Available: https://docbox.etsi.org/isg/nfv/open/Publications_pdf/Specs-Reports/NFV-IFA%20014v4.2.1%20-%20GS%20-%20Network%20Service%20Templates%20Spec.pdf.

[10] ETSI ISG NFV, "ETSI GS NFV-IFA 011 V4.2.1: VNF Descriptor and Packaging Specification," May 2021. [Online]. Available: https://docbox.etsi.org/isg/nfv/open/Publications_pdf/Specs-Reports/NFV-IFA%20011v4.2.1%20-%20GS%20-%20VNF%20Packaging%20Spec.pdf.

[11] ETSI ISG NFV, "ETSI GS NFV 003 V1.5.1: Terminology for Main Concepts in NFV," January 2020. [Online]. Available: https://www.etsi.org/deliver/etsi_gr/NFV/001_099/003/01.05.01_60/gr_NFV003v010501p.pdf.

[12] ETSI ISG NFV, "ETSI GS NFV-SWA 001: Virtual Network Functions Architecture," December 2014. [Online]. Available: https://www.etsi.org/deliver/etsi_gs/NFV-SWA/001_099/001/01.01.01_60/gs_NFV-SWA001v010101p.pdf.

[13] VMware, "vCloud Director User's Guide," March 2019. [Online]. Available: https://docs.vmware.com/en/VMware-Cloud-Director/9.7/vcd_97_user_guide.pdf.

[14] J. Sathyan, Fundamentals of EMS, NMS and OSS/BSS, CRC Press, 2016.

[15] T. Zhang, G. Eakman, I. Lee and O. Sokolsky, "Overhead-Aware Deployment of Runtime Monitors," in *International Conference on Runtime Verification*, 2019.

[16] C. Makaya, A. Dutta, S. Das, D. Chee, F. J. Lin, S. Komorita and H. Yokota, "Service Continuity Support in Self-organizing IMS Networks," in *2nd International Conference on Wireless Communication, Vehicular Technology, Information Theory and Aerospace & Electronic Systems Technology (Wireless VITAE)*, Chennai, 2011.

[17] F. Sultan, K. Srinivasan, D. Iye and L. Iftode, "Migratory TCP: Connection Migration for Service Continuity in the Internet," in *22nd International Conference on Distributed Computing Systems*, Vienna, 2002.

[18] B. Meroufel and G. Belalem, "Optimization of Checkpointing/Recovery Strategy in Cloud Computing with Adaptive Storage Management," *Concurrency and Computation: Practice and Experience,* vol. 30, no. 24, 2018.

[19] S. Marzouk, A. J. Maâlej, B. I. Rodriguez and M. Jmaiel, "Periodic Checkpointing for Strong Mobility of Orchestrated Web Services," in *2009 Congress on Services - I*, Los Angeles, 2009.

[20] M. I. Jordan and T. M. Mitchell, "Machine Learning: Trends, Perspectives, and Prospects," *Science,* vol. 349, no. 6245, pp. 255-260, 2015.

[21] N. Shukla, Machine Learning with TensorFlow, Shelter Island, NY: Manning Publications, 2018.

[22] O. I. Abiodun, A. Jantan, A. E. Omolara, K. V. Dada, N. A. Mohamed and H. Arshad, "State-of-the-art in Artificial Neural Network Applications: A Survey," *Heliyon,* vol. 4, no. 11, 2018.

[23] P. Rivas, Deep Learning for Beginners, Birmingham: Packt Publishing Ltd., 2020.

[24] M. Wang, Y. Cui, X. Wang, S. Xiao and J. Jiang, "Machine Learning for Networking: Workflow, Advances and Opportunities," *IEEE Network,* vol. 32, no. 2, pp. 92 - 99, 2017.

[25] S. Nikolenko, Synthetic Data for Deep Learning, San Francisco: Springer, 2021.

[26] OpenStack, "Open Source Cloud Computing Infrastructure," May 2022. [Online]. Available: https://www.openstack.org/. [Accessed May 2022].

[27] OpenStack, "Tacker," May 2022. [Online]. Available: https://wiki.openstack.org/wiki/Tacker. [Accessed May 2022].

[28] E. Bauer and D. Adams, Reliability and Availability of Cloud Computing, John Wiley & Sons, 2012.

[29] E. Bauer, R. Adams and D. Eustace, Beyond Redundancy: How Geographic Redundancy Can Improve Service Availability and Reliability of Computer-based Systems, John Wiley & Sons, 2012.

[30] T. Critchley, High Availability IT Services, CRC Press, 2015.

[31] M. v. Steen and A. S. Tanenbaum, Distributed Systems, Maarten van Steen, 2018.

[32] K. P. Birman, Reliable Distributed Systems: Technologies, Web Services, and Applications, Springer, 2005.

[33] K. P. Birman, Guide to Reliable Distributed Systems: Building High-Assurance Applications and Cloud-Hosted Services, Springer, 2012.

[34] W. Jia and W. Zhou, Distributed Network Systems: From Concepts to Implementations, Springer, 2005.

[35] Y. Izrailevsky and C. Bell, "Cloud Reliability," *IEEE Cloud Computing,* vol. 5, no. 3, pp. 39 - 44, 2018.

[36] P. Garraghan, R. Yang, Z. Wen, A. Romanovsky, J. Xu, R. Buyya and R. Ranjan, "Emergent Failures: Rethinking Cloud Reliability at Scale," *IEEE Cloud Computing,* vol. 5, no. 5, pp. 12 - 21, 2018.

[37] R. Achary and P. Raj, Cloud Reliability Engineering: Technologies and Tools, Boca Raton: CRC Press, 2021.

[38] A. Kanso, F. Khendek, M. Toeroe and A. Hamou-Lhadj, "Automatic Configuration Generation for Service High Availability with Load Balancing," *Concurrency and Computation: Practice & Experience,* vol. 25, no. 2, p. 265–287, 2013.

[39] P. Pourali, M. Toeroe and F. Khendek, "Pattern Based Configuration Generation for Highly Available COTS Components Based Systems," *Information and Software Technology,* vol. 74, pp. 143-159, 2016.

[40] F. Jingyuan, Y. Zilong, G. Chaowen, G. Xiujiao, R. Kui and Q. Chunming, "GREP: Guaranteeing Reliability with Enhanced Protection in NFV," in *ACM SIGCOMM Workshop on Hot Topics in Middleboxes and Network Function Virtualization*, London, 2015.

[41] W. Ding, H. Yu and S. Luo, "Enhancing the Reliability of Services in NFV with the Cost-efficient Redundancy Scheme," in *IEEE International Conference on Communications (ICC)*, Paris, France, 2017.

[42] Q. Han, W. Zang and J. Lan, "Virtual Network Protection Strategy to Ensure the Reliability of SFC in NFV," in *6th International Conference on Information Engineering*, Dalian Liaoning, 2017.

[43] 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 *GLOBECOM*, Singapore, 2017.

[44] S. Sharma, A. Engelmann, A. Jukan and A. Gumaste, "VNF Availability and SFC Sizing Model for Service Provider Networks," *IEEE Access,* vol. 8, pp. 119768 - 119784, 2020.

[45] T. Soenen, W. Tavernier, D. Colle and M. Pickavet, "Optimising Microservice-based Reliable NFV Management & Orchestration Architectures," in *International Workshop on Resilient Networks Design and Modeling (RNDM)*, Alghero, Italy, 2017.

[46] J. Fan, C. Guan, Y. Zhao and C. Qiao, "Availability-aware Mapping of Service Function Chains," in *IEEE INFOCOM*, Atlanta, GA, USA, 2017.

[47] L. Fang, X. Zhang, K. Sood, Y. Wang and S. Yu, "Reliability-aware Virtual Network Function Placement in Carrier Networks," *Journal of Network and Computer Applications,* vol. 154, 2020.

[48] G. Moualla, T. Turletti and D. Saucez, "An Availability-aware SFC Placement Algorithm for Fat-Tree Data Centers," in *7th International Conference on Cloud Networking (CloudNet)*, Tokyo, 2018.

[49] A. Hmaity, M. avi, F. Musumeci, M. Tornatore and A. Pattavina, "Virtual Network Function Placement for Resilient Service Chain Provisioning," in *8th International Workshop on Resilient Networks Design and Modeling (RNDM)*, Halmstad, Sweden, 2016.

[50] 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 Transactions on Network and Service Management,* vol. 14, no. 3, pp. 554 - 568, 2017.

[51] K. Wolter, Stochastic Models for Fault Tolerance: Restart, Rejuvenation and Checkpointing, Springer, 2010.

[52] M. Nabi, M. Toeroe and F. Khendek, "Availability in the Cloud: State of the Art," *Network and Computer Applications,* vol. 60, pp. 54-67, 2016.

[53] I. P. Egwutuoha, D. Levy, B. Selic and S. Chen, "A Survey of Fault Tolerance Mechanisms and Checkpoint/Restart Implementations for High Performance Computing Systems," *The Journal of Supercomputing,* vol. 65, p. 1302–1326, 2013.

[54] M. Hasan and M. Singh Goraya, "Fault Tolerance in Cloud Computing Environment: A Systematic Survey," *Computers in Industry,* vol. 99, pp. 156-172, 2018.

[55] 3GPP, "TS 28.516 V16.0.0; Fault Management (FM) for Mobile Networks that Include Virtualized Network Functions," July 2020. [Online]. Available: https://www.3gpp.org/ftp/Specs/archive/28_series/28.516/28516-g00.zip.

[56] I. Angelopoulos, E. Trouva and G. Xilouris, "A Monitoring Framework for 5G Service Deployments," in *IEEE 22nd International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*, Lund, Sweden, 2017.

[57] G. Gardikis, I. Koutras, G. Mavroudis, S. Costicoglou, G. Xilouris, C. Sakkas and A. Kourtis, "An Integrating Framework for Efficient NFV," in *IEEE NetSoft Conference and Workshops (NetSoft)*, Seoul, Korea, 2016.

[58] M. Xie, C. Banino-Rokkones, P. Grønsund and A. J. Gonzalez, "Service Assurance Architecture in NFV," in *IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, Berlin, Germany , 2017.

[59] N. Yuan, W. He, J. Shen, X. Qiu, S. Guo and W. Li, "Delay-Aware NFV Resource Allocation with Deep Reinforcement Learning," in *IEEE/IFIP Network Operations and Management Symposium (NOMS)*, Budapest, Hungary, 2020.

[60] C. A. Ouedraogo, E.-F. Bonfoh, S. Medjiah, C. Chassot and S. Yangui, "A Prototype for Dynamic Provisioning of QoS-oriented Virtualized Network Functions in the Internet of

Things," in *4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*, Montreal, Canada, 2018.

[61] S. Rahman, T. Ahmed, M. Huynh, M. Tornatore and B. Mukherjee, "Auto-Scaling Network Service Chains Using Machine Learning and Negotiation Game," *IEEE Transactions on Network and Service Management,* vol. 17, no. 3, pp. 1322 - 1336, 2020.

[62] S. Lange, H.-G. Kim, S.-Y. Jeong, H. Choi, J.-H. Yoo and J. W.-K. Hong, "Machine Learning-based Prediction of VNF Deployment Decisions in Dynamic Networks," in *20th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, Matsue, Japan, 2019.

[63] ETSI ISG NFV, "ETSI GS NFV-REL 002 V1.1.1: Reliability; Report on Scalable Architectures for Reliability Management," September 2015. [Online]. Available: http://www.etsi.org/deliver/etsi_gs/NFV-REL/001_099/002/01.01.01_60/gs_nfv-rel002v010101p.pdf.

[64] ETSI ISG NFV, "ETSI GS NFV-REL 004 V1.1.1: Assurance; Report on Active Monitoring and Failure Detection," April 2016. [Online]. Available: https://www.etsi.org/deliver/etsi_gs/NFV-REL/001_099/004/01.01.01_60/gs_nfv-rel004v010101p.pdf.

[65] ETSI ISG NFV, "ETSI GS NFV-REL 006 V3.1.1: Reliability; Maintaining Service Availability and Continuity Upon Software Modification," February 2018. [Online]. Available: https://www.etsi.org/deliver/etsi_gs/NFV-REL/001_099/006/03.01.01_60/gs_nfv-rel006v030101p.pdf.

[66] ETSI ISG NFV, "ETSI GR NFV-REL 007 V1.1.1: Reliability; Report on the resilience of NFV-MANO critical capabilities," September 2017. [Online]. Available: https://www.etsi.org/deliver/etsi_gr/NFV-REL/001_099/007/01.01.01_60/gr_nfv-rel007v010101p.pdf.

[67] ETSI ISG NFV, "ETSI GR NFV-REL 010 V3.1.1: Reliability; Report on NFV Resiliency for the Support of Network Slicing," June 2019. [Online]. Available: https://www.etsi.org/deliver/etsi_gr/NFV-REL/001_099/010/03.01.01_60/gr_NFV-REL010v030101p.pdf.

[68] ETSI ISG NFV, "ETSI GR NFV-REL 011 V4.1.1: Network Functions Virtualisation (NFV) Release 4; Management and Orchestration; Report on NFV-MANO software modification," November 2020. [Online]. Available: https://www.etsi.org/deliver/etsi_gr/NFV-REL/001_099/011/04.01.01_60/gr_NFV-REL011v040101p.pdf.

[69] ETSI ISG NFV, "ETSI GR NFV-REL 012 V1.1.1: Report on Availability and Reliability Under Failure and Overload Conditions in NFV-MANO," November 2021. [Online]. Available: https://docbox.etsi.org/isg/nfv/open/Publications_pdf/Specs-Reports/NFV-REL%20012v1.1.1%20-%20GR%20-%20MANO%20robustness.pdf.

[70] ETSI ISG NFV, "ETSI GS NFV-IFA 010 V3.3.1:Management and Orchestration; Functional requirements specification," September 2019. [Online]. Available: https://www.etsi.org/deliver/etsi_gs/NFV-IFA/001_099/010/03.03.01_60/gs_NFV-IFA010v030301p.pdf.

[71] R. Gupta, H. Naik and P. Beckman, "Understanding Checkpointing Overheads on Massive-ScaleSystems: Analysis of the IBM BlueGene/P System," *High Performance Computing Applications,* vol. 25, pp. 180-192, 2010.

[72] S. Sadi and B. Yagoubi, "On the Optimum Checkpointing Interval Selection for Variable Size Checkpoint Dumps," in *CIIA*, 2015.

[73] N. Nazarzadeoghaz, F. Khendek and M. Toeroe, "Automated Design of Network Services from Network Service Requirements," in *Innovation in Clouds, Internet and Networks and Workshops (ICIN)*, Paris, 2020.

[74] S. Herker, W. Kiess, X. An and A. Kirstädter, "On the Trade-off Between Cost and Availability of Cirtual Networks," in *IFIP Networking Conference*, Trondheim, Norway, 2014.

[75] A. Immonen and E. Niemelä, "Survey of Reliability and Availability Prediction Methods from the Viewpoint of Software Architecture," *Software & System Modeling,* vol. 7, pp. 49-65, 2008.

[76] I. Bazovsky, Reliability Theory and Practice, Dover Publications, 2004.

[77] E. Ruijters and M. Stoelinga, "Fault Tree Analysis: A Survey of the State-of-the-art in Modeling, Analysis and Tools," *Computer Science Review,* Vols. 15-16, pp. 29-62, 2015.

[78] F. A. Tillman, C.-L. Hwang and W. Kuo, "Determining Component Reliability and Redundancy for Optimum System Reliability," in *IEEE Transactions on Reliability* , 1997.

[79] B. Satzger, A. Pietzowski, W. Trumler and T. Ungerer, "A Lazy Monitoring Approach for Heartbeat-Style Failure Detector," in *Third International Conference on Availability, Reliability and Security*, Barcelona, 2008.

[80] K. Manoj, C. Abhishek and K. Vikas, "A Comparison between Different Checkpoint Schemes with Advantages and Disadvantages," *International Journal of Computer Applications,* pp. 36-39, 2014.

[81] B. Nicolae, A. Moody, E. Gonsiorowski, K. Mohror and F. Cappello, "VeloC: Towards High Performance Adaptive Asynchronous Checkpointing at Large Scale," in *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, Rio de Janeiro, 2019.

[82] X. Xiao, "Chapter 2 - What Is QoS?," in *Technical, Commercial and Regulatory Challenges of QoS*, Massachusetts, Morgan Kaufmann, 2008, pp. 13-35.

[83] B. Chatras and F. F. Ozog, "Network Functions Virtualization: the Portability Challenge," *IEEE Network,* vol. 30, no. 4, pp. 4-8, 2016.

[84] ETSI GS NFV, "ETSI GS NFV-IFA 005 V3.4.1: Management and Orchestration; Or-Vi reference point - Interface and Information Model Specification," June 2020. [Online]. Available: https://www.etsi.org/deliver/etsi_gs/NFV-IFA/001_099/005/03.04.01_60/gs_NFV-IFA005v030401p.pdf.

[85] ETSI GS NFV, "ETSI GS NFV-IFA 006 V3.4.1: Management and Orchestration; Vi-Vnfm reference point - Interface and Information Model Specification," June 2020. [Online]. Available: https://docbox.etsi.org/isg/nfv/open/Publications_pdf/Specs-Reports/NFV-IFA%20006v3.4.1%20-%20GS%20-%20Vi-Vnfm%20ref%20point%20Spec.pdf.

[86] ETSI GS NFV, "ETSI GS NFV-IFA 007 V3.4.1: Management and Orchestration; Or-Vnfm reference point - Interface and Information Model Specification," June 2020. [Online]. Available: https://www.etsi.org/deliver/etsi_gs/NFV-IFA/001_099/007/03.04.01_60/gs_NFV-IFA007v030401p.pdf.

[87] ETSI GS NFV, "ETSI GS NFV-IFA 013 V3.4.1: Management and Orchestration; Os-Ma-nfvo reference point - Interface and Information Model Specification," June 2020. [Online]. Available: https://www.etsi.org/deliver/etsi_gs/NFV-IFA/001_099/013/03.04.01_60/gs_NFV-IFA013v030401p.pdf.

[88] J. Bergstra and Y. Bengio, "Random Search for Hyper-Parameter Optimization," *Journal of Machine Learning Research,* vol. 13, pp. 281-305, 2012.

[89] Canonical Ltd., "Ubuntu," [Online]. Available: https://ubuntu.com. [Accessed August 2022].

[90] FFmpeg, "FFmpeg," [Online]. Available: https://ffmpeg.org/. [Accessed August 2022].

[91] F5, "F5 NGINX Sprint," [Online]. Available: https://www.nginx.com. [Accessed August 2022].

[92] MariaDB, "MariaDB," [Online]. Available: https://mariadb.com/. [Accessed August 2022].

[93] Apache Software Foundation, "Apache," [Online]. Available: https://apache.org/. [Accessed August 2022].

[94] MariaDB, "What is MariaDB Galera Cluster?," [Online]. Available: https://mariadb.com/kb/en/what-is-mariadb-galera-cluster/. [Accessed August 2022].

[95] Zabbix, "Open Source Network Monitoring Solution," [Online]. Available: https://www.zabbix.com/. [Accessed July 2022].

Appendix I

Table A-I-1: Mathematical notations used in this thesis

Symbol	Description
A_X	Availability of X
ADT_X	Acceptable down time for X
$ASDD_X$	Acceptable service data disruption for X
$ASDT_X$	Acceptable service disruption time for X
$C_C(X)$	Overall computing cost of X
$C_h(X)$	Hosting cost of X
$C_N(X)$	Networking cost of X
CND_X	Checkpointing network delay for X
CpI_X	Checkpointing interval for X
CPN	Number of possible configuration values for the checkpointing interval of a VNF
FDT_X	Failure detection time for X
FoT_X	Failover time for X
HI_X	Health-check Interval for X
HRN	Number of possible configuration values for the health-check rate of a VNF
MHR_X	Monitoring health-check rate for X
$MTTR_X$	Mean time to repair for X
NON	Number of possible networking options for remote checkpointing of a VNF
OT_X	Outage time of X
R_X	Reliability of X
RT_X	Restart time for X
SDD_X	Service data disruption of X

SDT_X	Service data time of X
t	Time period
$TBFLC_X$	Time between a failure and the latest committed checkpoint for X
TDT_X	Total down time of X
ToT_X	Takeover time for X
$VnfEA_X$	Expected availability for VNF X
λ_X	Failure rate of X