ETSI GS NFV-REL 003 V1.1.2 (2016-07)

Network Functions Virtualisation (NFV);
Reliability;
Report on Models and Features for End-to-End Reliability

Reference

RGS/NFV-REL003ed112

Keywords

availability, NFV, reliability, resiliency

Contents

Intellectual Property Rights

Foreword

This Group Specification (GS) has been produced by ETSI Industry Specification Group (ISG) Network Functions Virtualisation (NFV).

Modal verbs terminology

In the present document "**shall**", "**shall not**", "**should**", "**should not**", "**may**", "**need not**", "**will**", "**will not**", "**can**" and "**cannot**" are to be interpreted as described in clause 3.2 of the ETSI Drafting Rules (Verbal forms for the expression of provisions).

"**must**" and "**must not**" are **NOT** allowed in ETSI deliverables except when used in direct citation.

1 Scope

The present document describes the models and methods for end-to-end reliability in NFV environments and software upgrade from a resilience perspective. The scope of the present document covers the following items:

- Study reliability estimation models for NFV including modelling architecture.
- Study NFV reliability and availability methods.
- Develop reliability estimation models for these methods, including dynamic operational aspects such as impact of load and life-cycle operations.
- Study reliability issues during NFV software upgrade and develop upgrade mechanisms for improving resilience.
- Develop guidelines to realise the differentiation of resiliency for different services.

2 References

2.1 Normative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

Referenced documents which are not found to be publicly available in the expected location might be found at http://docbox.etsi.org/Reference.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long term validity.

The following referenced documents are necessary for the application of the present document.

Not applicable.

2.2 Informative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long term validity.

The following referenced documents are not necessary for the application of the present document but they assist the user with regard to a particular subject area.

[i.1] ETSI GS NFV 002: "Network Functions Virtualisation (NFV); Architectural Framework".

[i.2] ETSI GS NFV-REL 001: "Network Functions Virtualisation (NFV); Resiliency Requirements".

[i.3] http://www.crn.com/slide-shows/cloud/240165024/the-10-biggest-cloud-outages-of-2013.htm.

[i.4] http://www.crn.com/slide-shows/cloud/300075204/the-10-biggest-cloud-outages-of-2014.htm.

[i.5] ETSI GS NFV-MAN 001: "Network Functions Virtualisation (NFV); Management and Orchestration".

[i.6] ETSI GS NFV-SWA 001: "Network Functions Virtualisation (NFV); Virtual Network Functions Architecture".

[i.7] SA Forum SAI-AIS-AMF-B.04.01: "Service Availability Forum Application Interface Specification".

[i.8] ETSI GS NFV-REL 002: "Network Functions Virtualisation (NFV); Reliability; Report on Scalable Architectures for Reliability Management".

[i.9] ETSI GS NFV-REL 004: "Network Functions Virtualisation (NFV); Assurance; Report on Active Monitoring and Failure Detection".

[i.10] ETSI GS NFV-INF 010: "Network Functions Virtualisation (NFV); Service Quality Metrics".

[i.11] ETSI GS NFV-INF 001: "Network Functions Virtualisation (NFV); Infrastructure Overview".

[i.12] IEEE 802.1ax™: "IEEE standard for local and metropolitan area networks -- Link aggregation".

[i.13] IEEE 802.1aq™: "Shortest Path Bridging".

[i.14] ETSI GS NFV-REL 005: Network Functions Virtualisation (NFV); Assurance; Quality Accountability Framework.

[i.15] QuestForum: "TL 9000 Measurements Handbook", release 5.0, July 2012.

NOTE: Available at http://www.tl9000.org/handbooks/measurements_handbook.html.

[i.16] QuEST Forum: "Quality Measurement of Automated Lifecycle Management Actions," 1.0, August 18th, 2015.

NOTE: Available at http://www.tl9000.org/resources/documents/QuEST_Forum_ALMA_Quality_Measurement_150819.pdf.

[i.17] B. Baudry and M. Monperrus: "The multiple facets of software diversity: recent developments in year 2000 and beyond", 2014. <hal-01067782>.

[i.18] L.H. Crow: "Reliability analysis for complex repairable systems", in "Reliability and biometry - statistical analysis of lifelength" (F. Prochan and R.J. Serfling, Eds.), SIAM Philadelphia, 1974, pp. 379-410.

[i.19] Y. Deswarte, K. Kanoun and J.-C. Laprie: "Diversity against accidental and deliberate faults", Conf. on Computer Security, Dependability, and Assurance: From Needs to Solutions, Washington, DC, USA, July 1998.

[i.20] J.T. Duane: "Learning curve approach to reliability monitoring", IEEE™ Transactions on Aerospace, AS-2, 2, 1964, pp. 563-566.

[i.21] A.L. Goel and K. Okumoto: "Time dependent error detection rate model for software reliability and other performance measures", IEEE™ Transactions on Reliability, R-28, 1, 1979, pp. 206-211.

[i.22] Z. Jelinski Z. and P.B. Moranda: "Statistical computer performance evaluation", in "Software reliability research" (W. Freiberger, Ed.), Academic Press, 1972, pp. 465-497.

[i.23] J.E. Just and M. Cornwell: "Review and analysis of synthetic diversity for breaking monocultures", ACM Workshop on Rapid Malcode, New York, NY, USA, 2004, pp. 23-32.

[i.24] J.C. Knight: "Diversity", Lecture Notes in Computer Science 6875, 2011.

[i.25] P. Larsen, A. Homescu, S. Brunthaler and M. Franz: "Sok: automated software diversity", IEEE™ Symposium on Security and Privacy, San Jose, CA, USA, May 2014, pp. 276-291.

[i.26] B. Littlewood, P. Popov and L. Strigini: "Modeling software design diversity: a review", ACM Computing Surveys (CSUR), 33(2), 2001, pp. 177-208.

[i.27] P.B. Moranda: "Event altered rate models for general reliability analysis", IEEE™ Transactions on Reliability, R-28, 5, 1979, pp. 376-381.

[i.28] J.D. Musa and K. Okumoto: "A logarithmic Poisson execution time model for software reliability measurement", 7th Int. Conf. on Software Engineering, Orlando, FL, USA, March 1984, pp. 230-238.

[i.29] I. Schaefer et al.: "Software diversity: state of the art and perspectives", International Journal on Software Tools for Technology Transfer, 14, 2012, pp. 477-495.

[i.30] S. Yamada, M. Ohba and S. Osaki: "S-shaped reliability growth modelling for software error detection", IEEE Transactions on Reliability, R-35, 5, 1983, pp. 475-478.

[i.31] ETSI GS NFV-INF 003: "Network Functions Virtualisation (NFV); Infrastructure; Compute Domain".

[i.32] ETSI GS NFV 003: "Network Functions Virtualisation (NFV); Terminology for main concepts in NFV".

[i.33] IEEE Reliability Society: "Recommended Practice on Software Reliability", IEEE™ Std 1633, 2008.

[i.34] NFV PoC#35 final report.

3 Definitions and abbreviations

3.1 Definitions

For the purposes of the present document, the terms and definitions given in ETSI GS NFV 003 [i.32], ETSI GS NFV-REL 001 [i.2] and the following apply:

fault detection: process of identifying an undesirable condition (fault or symptom) that may lead to the loss of service from the system or device

fault diagnosis: high confidence level determination of the required repair actions for the components that are suspected to be faulty

NOTE: Diagnosis actions are generally taken while the component being diagnosed is out of service.

fault isolation: isolation of the failed component(s) from the system

NOTE: The objectives of fault isolation include avoidance of fault propagation to the redundant components and/or simultaneous un-intended activation of active and backup components in the context of active-standby redundancy configurations (i.e. "split-brain" avoidance).

fault localization: determining the component that led to the service failure and its location

fault management notification: notification about an event pertaining to fault management

EXAMPLE: Fault management notifications include notifications of fault detection events, entity availability state changes, and fault management phase related state progression events.

fault recovery: full restoration of the original intended system configuration, including the redundancy configuration

NOTE: For components with protected state, this phase includes bringing the new protecting unit online and transferring the protected state from the active unit to the new unit.

fault remediation: restoration of the service availability and/or continuity after occurrence of a fault

fault repair: removal of the failed unit from the system configuration and its replacement with an operational unit

NOTE: For the hardware units that pass the full diagnosis, it may be determined that the probable cause was a transient fault, and the units may be placed back into the operational unit pool without physical repair.

state protection: protection of the service availability and/or service continuity relevant portions of system or subsystem state against faults and failures

NOTE: State protection involves replicating the protected state to a redundant resource.

3.2 Abbreviations

For the purposes of the present document, the following abbreviations apply:

3GPP	Third Generation Partnership Project
AIS	Application Interface Specification
API	Application Programming Interface
ATCA	Advanced TCA (Telecom Computing Architecture)
BER	Bit Error Rates
CoS	Class of Service
COTS	Commercial Off-The-Shelf
CP	Connection Point
CPU	Central Processing Unit
DARPA	Defence Advanced Research Projects Agency
DNS	Domain Name Service
E2E	End-to-End
ECMP	Equal-Cost Multi-Path
EM	Element Manager
EMS	Element Management System
ETBF	Exponential Times Between Failures
GTP	GPRS Tunnelling Protocol
HPP	Homogenous Poisson Process
IMS	IP Multimedia Subsystem
IP	Internet Protocol
LACP	Link Aggregation Control Protocol
LAN	Local Area Network
LB	Load Balancer
LCM	Life Cycle Management
LOC	Lines of Code
MAN	Metropolitan Area Network
MOS	Mean Opinion Score
MPLS	Multiprotocol Label Switching
MTBF	Mean Time Between Failure
MTD	Moving Target Defense
MTTF	Mean time To Failure
MTTR	Mean Time To Repair
NF	Network Function
NFP	Network Forwarding Path
NFVI	Network Functions Virtualisation Infrastructure
NFV-MANO	Network Functions Virtualisation Management and Orchestration
NFVO	Network Functions Virtualisation Orchestrator
NHPP	Non-Homogenous Poisson Processes
NIC	Network Interface Card
NOP	No Operation
NS	Network Service
NSD	Network Service Descriptor
OS	Operation System
OSNR	Optical Signal to Noise Ratio
OSPF	Open Shortest Path First
OSS	Operations Support System
PCI	Peripheral Component Interconnect
PDP	Packet Data Protocol
PNF	Physical Network Function
PNFD	Physical Network Function Descriptor
QoS	Quality of Service
RPO	Recovery Point Objective

RTO	Recovery Time Objective
SA	Service Availability
SAL	Service Availability Level
SDN	Software Defined Networking
SFF	Service Function Forwarding
SIP	Session Initiation Protocol
SLA	Service Level Agreement
SO	Service Outage
SQL	Structured Query Language
SR-IOV	Single Root I/O Virtualisation
TCP	Transmission Control Protocol
TMR	Triple Modular Redundancy
ToR	Top of Rack
VDU	Virtualisation Deployment Unit
VIM	Virtualised Infrastructure Manager
VL	Virtual Link
VLAN	Virtual Local Area Network
VLD	Virtual Link Descriptor
VM	Virtual Machine
VNF	Virtualised Network Function
VNFC	Virtualised Network Function Component
VNFCI	VNFC Instance
VNFD	Virtualised Network Function Descriptor
VNFFG	VNF Forwarding Graph
VNFFGD	VNF Forwarding Graph Descriptor
VNFI	VNF Instance
VNFM	VNF Manager
VXLAN	Virtual eXtensible Local Area Network
WAN	Wide Area Network

4 Overview

4.1 End-to-End network service chain

In most cases, an End-to-End (E2E) network service (e.g. mobile voice/data, Internet access, virtual private network) can be described by one (several) NF Forwarding Graph(s) linking end points through interconnected Network Functions (NFs). The network service behaviour is a combination of the behaviour of its constituent functional blocks, which can include individual NFs and virtual links. Therefore, the reliability and availability of a network service have to be estimated based on the reliability and availability of these constituent functional blocks.

These network functions can be implemented in a single operator network or interwork between different operator networks ETSI GS NFV 002 [i.1], by partitioning the E2E network service into multiple service chains, e.g. service chains for access network and core network. Each service chain can be regarded as a chain of NFs. Each network service has E2E characteristics referring to an explicitly demarked service chain that includes multiple network functions. A service chain may have the ingress demarcation to some peripheral elements, like the customer-facing edge of a network service, e.g. a session border controller protecting a voice-over LTE IMS core, and the other demarcation of this service chain might be the border gateway with another service provider for a voice call between service providers. Thus, the chain of this network service includes:

1) Both ingress and egress perimeter elements.

2) All PNFs and VNFs in the service delivery path between the two perimeter elements.

3) All networking and interworking equipment and facilities between the two perimeter elements.

4) Supporting infrastructure (e.g. data centres) and inputs (e.g. electric power, operator policies, etc.).

An E2E service, where both "ends" are customers, comprises several E2E service delivery chains, which are mutually connected in parallel or in series, to construct a network service graph.

4.2 Reliability model of an end-to-end service

Reliability and availability of E2E services are among the subjects that operators take into consideration when deploying service, which need network functions and links for connecting these functions. Though the quality metrics, such as key performance indicators (KPIs) for reliability, availability and others (see Annex B) are monitored after deployment, traditionally, network operators estimate the reliability and availability of E2E services by evaluating those of each "demarcated service chain" described in clause 4.1, and by calculating them according to the connected patterns of the chains.

This concept is applicable for networks in the virtualised environment as well as in the traditional physical environment. The availability of the end-to-end network service composed of several service chains can be estimated as a function of the availability of each service chain and the topological connection pattern of the chains.

An example of this concept is shown in Figure 1. The availability of an end-to-end service is calculated as the product of the availabilities of the demarcated service chains comprising the E2E network.

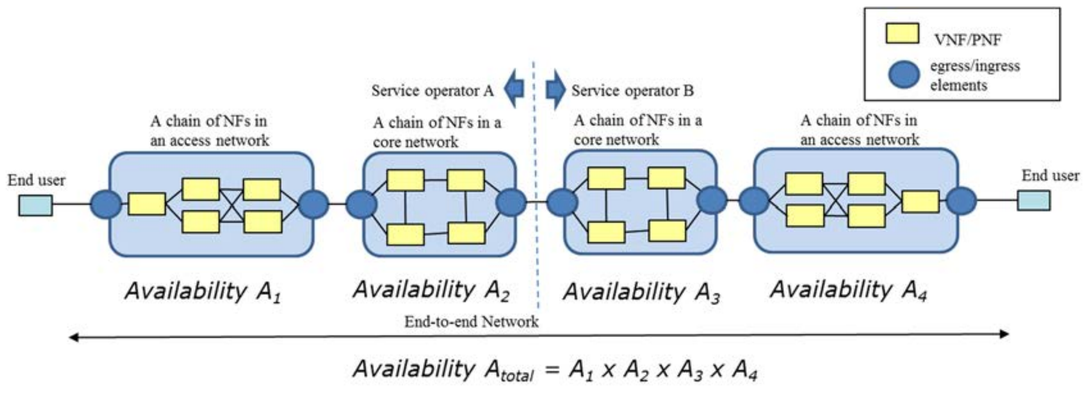

VNF/PNF
egress/ingress elements
Service operator A
Service operator B
A chain of NFs in an access network
A chain of NFs in a core network
A chain of NFs in a core network
A chain of NFs in an access network
End user
End user
Availability A_1
Availability A_2
Availability A_3
Availability A_4
End-to-end Network
Availability $A_{total} = A_1 \times A_2 \times A_3 \times A_4$

Figure 1: E2E availability of a network service composed of four demarcated service chains connected in series provided by two service operators

Thus, the first part of this study focuses on the area where network functions are virtualised, and analyses the models and features to maximize the reliability of the E2E service chains.

Though there are multiple methods to treat the availability of an E2E service, such as the ones shown in Annex C and ETSI GS NFV-REL 005 [i.14], the present document describes the modelling of an "E2E service" in an NFV environment for estimating its reliability and availability and the features to ensure the targeted objectives during operation.

In an NFV environment, VNFs and virtual links are placed over an NFV Infrastructure, which is composed of a virtualisation layer and hardware resources in physical locations. The present document investigates the relationship among these elements and NFV-MANO functions, taking the following functions into consideration in order to estimate the reliability and availability of a virtualised service chain: lifecycle operations, fault management cycle, and mechanisms to implement them which affect reliability and service downtime.

4.3 Structure of the present document

Reliability estimation techniques and software reliability models are presented in clause 5, and reliability/availability methods are further developed in clause 6 for use in an NFV environment. Following that, reliability estimation models are developed in two sample use cases based on these methods in clause 7. Software upgrade in an NFV environment is also described as one of the methods to increase availability and reliability in clause 8. Since the NFV framework is such that the service availability and reliability do not need to be "built to the peak" for all service flows, Service Level Agreements (SLAs) can be defined and applied according to given resiliency classes. Clause 9 presents a method for deploying service resilience requirements and principles for managing service availability and reliability differentiation of service flows.

5 Generic reliability and availability modelling and estimation

5.1 Introduction

This clause provides generic concepts on reliability and availability modelling and estimation. It starts with an example of estimation using the reliability block diagram technique. Software reliability modelling and estimation are then presented to show how to evaluate the reliability of software.

5.2 Reliability models and estimations

5.2.1 Basic equations

The reliability and availability of a complex system such as an NFV deployment can be modelled by breaking it down into its constituent components, of which the reliability and availability are known. For repairable components, this can be expressed using cycles of uninterrupted working intervals (uptime), followed by a repair period after a failure has occurred (downtime). The average length of the first interval is usually called the *Mean Time Between Failures* (MTBF), while the average length of the second is the *Mean Time To Repair* (MTTR, see clause 6.1 for a discussion of MTTR in NFV environments). Thus, the *availability* A of a component is:

$$\mathrm{A} = \frac{\mathrm{Uptime}}{\mathrm{Uptime+Downtime}} = \frac{\mathrm{MTBF}}{\mathrm{MTBF+MTTR}}. \qquad (5.1)$$

On the other hand, the *reliability* of a component is the probability that this component has not failed after a time period t, and is thus a function R(t) of t. It is typically modelled using the exponential distribution, using the failure rate $\lambda = \frac{1}{MTBF}$ as the parameter:

$$\mathrm{R(t)} = \mathrm{e}^{-\mathrm{t}\lambda} = \mathrm{e}^{\frac{-\mathrm{t}}{\mathrm{MTBF}}} \qquad (5.2)$$

The probability that a component has failed at least once within the same time period t is thus:

$$\mathrm{F(t)} = 1 - \mathrm{R(t)} = 1 - \mathrm{e}^{\frac{-\mathrm{t}}{\mathrm{MTBF}}} \qquad (5.3)$$

and is called *unreliability*.

Even if availability and reliability may appear to be interchangeable, they do have different meanings. From (5.1) and (5.2) it is clear that availability takes into account and is influenced by both MTBF and MTTR, whereas the reliability is only based on MTBF. As a result, two systems with the same MTBF can have quite different availabilities, while they have the same reliability (assuming that the exponential distribution is chosen for both in the reliability model).

To illustrate this difference, one can imagine a component that has a short MTBF, e.g. 10 hours, which means that R(t) is becoming low already for small values of t: R(20h) = 0,1353, i.e. the probability for the system to have run without failure for 20 hours is 13,53 %.

However, if this component has an even shorter MTTR (e.g. because it is using redundancy or it is simply re-instantiated quickly in case of failure), then the availability of the component would still be quite high, because it is available during a high share of the overall time. For an MTTR = 1 min, the availability would still be 99,83 %, although the reliability would continue being low because the component fails with a high probability after a short time.

A definition of what constitutes a failure in the context of NFV is also necessary. In extreme cases with very short service repair times, e.g. due to very fast failover in redundant configurations, the service might only be slightly degraded for a short time or even be seen as uninterrupted by external clients, particularly in scenarios with stateless VNFs. Thus, while individual components might fail, due to the composition of these components, e.g. in the form of resilience patterns, the composite might not experience a failure event. The basic effect of component composition on the reliability model is discussed in the following.

5.2.2 Modelling of composed systems

A service deployed in an NFV context can generally be assumed to be composed of a number of VNFs, which in turn may be composed of VNFCs. In order to be able to estimate and manage the reliability and availability of the composite system, these characteristics need to be derived from the individual parts it comprises. Two abstract patterns exist for this derivation, which will be shortly introduced in the following.

When combining components, two basic dependencies are possible, parallel and serial. A serial dependency of two subcomponents (Figure 2) means in the general sense that both need to function in order for the composite to function, i.e. both need to be available at the same time. As an example, SC1 could be the virtual infrastructure while SC2 the VNF running on top of it. Both need to work at the same time in order for the system to be available.

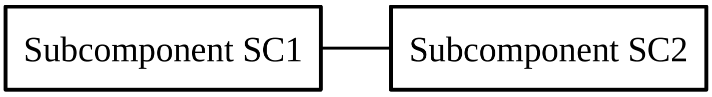


Figure 2: Serial composition of components

Therefore, for such a serial dependency, the availability of the composite C of two (independent) subcomponents SC1 and SC2 is:

$$\mathrm{A_C} = \mathrm{A_{SC1}} \times \mathrm{A_{SC2}} \qquad (5.4)$$

Regarding the reliability, the composite system does not fail during time period t only if all of its subcomponents do not fail during this period. Thus,

$$\mathrm{R_C(t)} = \mathrm{R_{SC1}(t)} \times \mathrm{R_{SC2}(t)} \qquad (5.5)$$

In contrast, a parallel dependency of two independent subcomponents models the situations where either of the two can be used (Figure 3). This abstract model assumes that the two subcomponents are fully redundant, i.e. that the service offered by the composite can be provided by SC1 or SC2 without any difference in service quality (in particular, it is assumed that there is no service interruption due to failover). The Active-Active resilience patterns described in the present document are typical examples for the instantiation of this abstract model, as long as the failure of one subcomponent does not imply overloading the remaining one or loss of information.

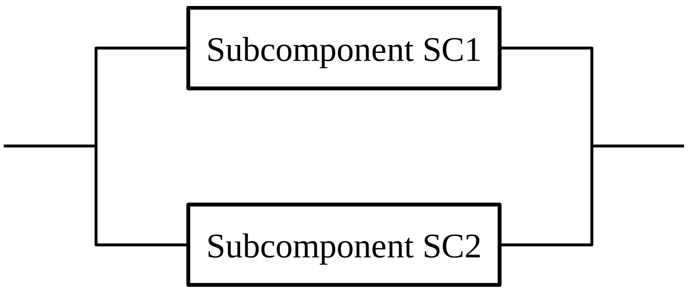


Figure 3: Parallel composition of components

Under these assumptions, the availability of such a composite parallel system C is the probability that at least one of the subcomponents is available:

$$\mathrm{A_C} = 1 - ((1 - \mathrm{A_{SC1}}) \times (1 - \mathrm{A_{SC2}})) \qquad (5.6)$$

With respect to the reliability, C is considered to fail during a time period t if both SC1 and SC2 fail during t:

$$F_C(t) = F_{SC1}(t) \times F_{SC2}(t),$$

or, in the general case with N redundant subcomponents,

$$F_C(t) = \prod_{i=1}^{N} F_{SCi}(t).$$

Thus, the reliability of such a composite component is:

$$R_C(t) = 1 - \prod_{i=1}^{N} F_{SCi}(t) = 1 - \prod_{i=1}^{N}(1 - R_{SCi}(t)) \qquad (5.7)$$

It should be noted that this is a simplification of a real system, since an instantaneous switch from one subcomponent to another is impossible. However, the assumption is justified if the composite failover time is low. The resilience patterns and mechanisms described in the present document are designed to keep these times as low as possible.

Using these two patterns, larger and more complex systems can be constructed, modelled and analysed, as it is done for the NFV use cases in clause 7.1.

5.3 Software reliability models and estimation

5.3.1 Introduction

Software reliability, an attribute of software quality

Software quality, an essential discipline of software engineering, encompasses diverse aspects of software excellence. It spans from financial concerns such as developing code respecting budget and time planning, to producing code which is easily maintainable: well documented code or *readability/understandability* and *reusability*, for instance, are contributing factors of this target. Programs easy to operate thanks to a well-designed user interface also contribute to software quality: *usability* characterizes this type of engineering effort. Other milestones in the journey towards software quality include:

- compliance to functional requirements;
- *portability* (i.e. ease in redeploying a program from one environment to another);
- performance efficiency (e.g. applications in high speed environments risk negatively impacting customer satisfaction due to response-time degradation);
- *scalability*;
- robustness from a *security* standpoint (e.g. prevention from SQL injection, cross-site scripting);
- adequate sizing;
- *testability* (i.e. code's ability to support acceptance criteria).

In accordance with the reliability definition mentioned in clause 5.2.1, software reliability is the probability of failure-free code operation for a specified environment and a specific period of time. Reflecting the level of risk and the likelihood of potential application failures, reliability also measures the defects injected due to modifications made to the software ("stability"). The root causes of poor reliability are found in a combination of non-compliance with good architectural and coding practices. This non-compliance can be detected by measuring the static quality attributes of an application. Assessing the static attributes underlying an application's reliability provides an estimate of the level of business risk and the likelihood of potential application failures and defects the application will experience when placed in operation.

Hardware reliability vs. software reliability

As software was introduced in communication equipment much later than hardware, the natural tendency is to compare it with hardware reliability. Physical faults (e.g. wear-out) are usually considered dominant in hardware, while design defects characterize software (i.e. they can occur without warning and are not function of operational time). Thus, traditional techniques such as failing over to a redundant component does not always ensure mitigation of the initial defect since it is more likely the same defect will be present in a redundant software component than in a redundant hardware component.

Maintenance may reduce hardware system wear-out, while software debug should increase its reliability, although the introduction of new bugs can influence reliability the other way round, as shown below.

Software reliability is strongly related to the operational profile; actually, although a program still contains plenty of bugs after its release, failures may not appear if the code usage does not activate these faults. It is noteworthy that 6 faults per 1 000 lines of code (LOC) are considered to be introduced by a professional programmer, while communication networks run millions of LOC.This software characteristic is much less present in the hardware context where reliability predictions are expressed whatever the usage conditions (under certain bracket conditions). Software failures thus result from changes in usage, in operating conditions, or new features added through an upgrade operation.

Hardware which has failed needs to be repaired before any further use, while relaunching a software program could be enough to restart it after a software failure. A straightforward way to express hardware and software reliability differences is through their respective failure rate. The classical bathtub curve (Figure 4) shows three phases of hardware reliability:

- a decreasing failure rate (early failures), also called burn in or infant mortality period;
- a constant failure rate during the useful life of a system (random failures);
- an increasing failure rate (due to wear-out) reflecting the system's end of life.

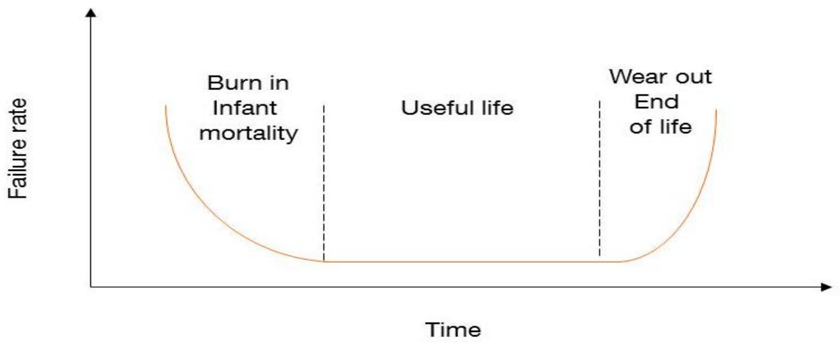


Figure 4: Bathtub curve for hardware failure rate

Software defects always exist when a system is deployed. The software's failure distribution curve (Figure 5) reflects changes in operational conditions that exercise those defects as well as new faults introduced by upgrades. There are two major differences between the hardware and software curves. One difference is that in the last phase, software does not have an increasing failure rate as hardware does. In this phase, software is approaching obsolescence. As there is no motivation for any upgrades or changes to the software, the failure rate is not modified. The second difference is that in the useful life phase, software experiences an increase in failure rate each time an upgrade is made. The failure rate levels off gradually, partly because of the defects found and fixed after the upgrades.

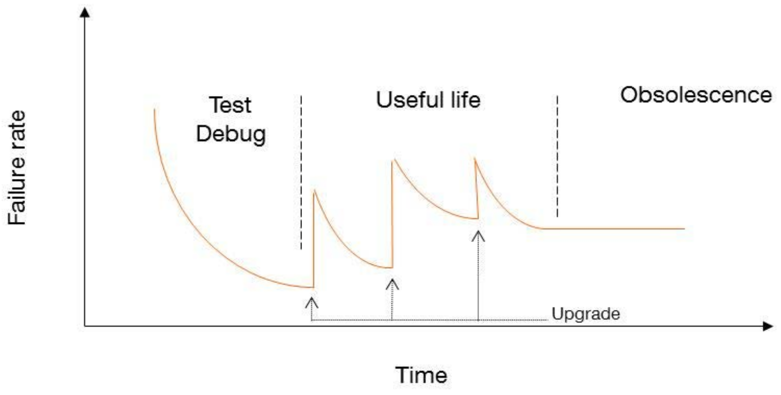


Figure 5: Software failure rate

The upgrades in Figure 5 imply feature upgrades, and not upgrades for reliability. For feature upgrades, the complexity of software is likely to be increased, since the functionality of software is enhanced. Even bug fixes may be a reason for more software failures, if the bug fix induces other defects into software. For reliability upgrades, it is possible to incur a drop in software failure rate, if the goal of the upgrade is enhancing software reliability, such as bugs tracking and removing, redesign or reimplementation of some modules using better engineering approaches.

Approaches for software reliability

As faults are the source of unreliability, a traditional way to apprehend the methods used to maximize software reliability is to classify them in four categories:

- Fault prevention: avoiding fault introduction during software design, e.g. proving program correctness using model checking or formal verification, is part of this approach.
- Fault removal: detecting faults after their introduction in a program is generally done through software testing.
- Fault tolerance: these *qualitative* methods help a software system to behave correctly, even in presence of faults - redundancy, as described in different parts of the present document, is a common method of this kind - another approach of this category (i.e. diversity) will be described in clause 5.3.2.
- Fault forecasting: estimating the number of remaining faults in order to predict future failures is a *quantitative* way to verify that reliability objectives are reached, often through probabilistic/statistic concepts - evaluating and measuring software reliability is the subject of clause 5.3.3.

5.3.2 Diversity, an approach for fault tolerance

As for redundancy, diversity is considered as an essential means for the building of resilience [i.19]. Early experiments with software diversity investigated N-version programming and recovery blocks to increase the reliability of embedded systems. These works advocate design and implementation diversity as a means for tolerating faults. Indeed, similar to natural systems, software systems including diverse functions and elements are able to cope with many kinds of unexpected problems and failures. The first part of this clause is related to managed technical approaches aiming at encouraging or controlling software diversity. The second part comprises techniques for artificially and automatically synthesizing diversity in software.

Managed diversity [i.29**]**

Diversity may be introduced intentionally during the design phase ('design diversity'), or may be exploited thanks to the existence of different software solutions supporting the same functional requirements ('natural diversity').

Design diversity

Since the late 1970's, many authors have devised engineering methods for software diversification to cope with accidental and deliberate faults. *N-version programming* and *recovery blocks* were the two initial proposals to introduce diversity in computation to limit the impact of bugs.

N-version programming may be defined as the independent generation of N ≥ 2 functionally equivalent programs from the same initial specification [i.26]. This consists in providing N development teams with the same requirements. Those teams then develop N independent versions, using different technologies, processes, verification techniques, etc. The N versions are run in parallel and a voting mechanism is executed on the N results. The increased diversity in design, programming languages and humans is meant to reduce the number of faults by emergence of the best behaviour, the emergence resulting from the vote on the output value. Figure 6 shows the case of 3-version, also known as *Triple Modular Redundancy(TMR).*

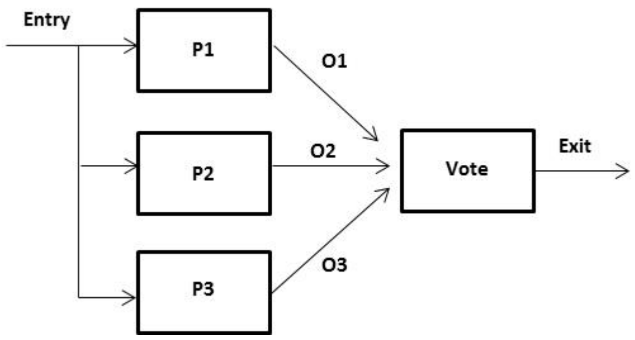


Figure 6: TMR operation

This initial definition of the N-version paradigm has been refined along different dimensions: the process, the product and the environment necessary for N-version development. For instance, random diversity (i.e. let independent teams develop their version) is distinguished from enforced diversity in which there is an explicit effort to design diverse algorithms or data structures.

At the same time, recovery blocks were proposed as a way of structuring the code, using diverse alternative software solutions, for fault tolerance. The idea is to have recovery blocks in the program, i.e. blocks equipped with error detection mechanisms, and one or more spares that are executed in case of errors. These spares are diverse variant implementations of the function (Figure 7).

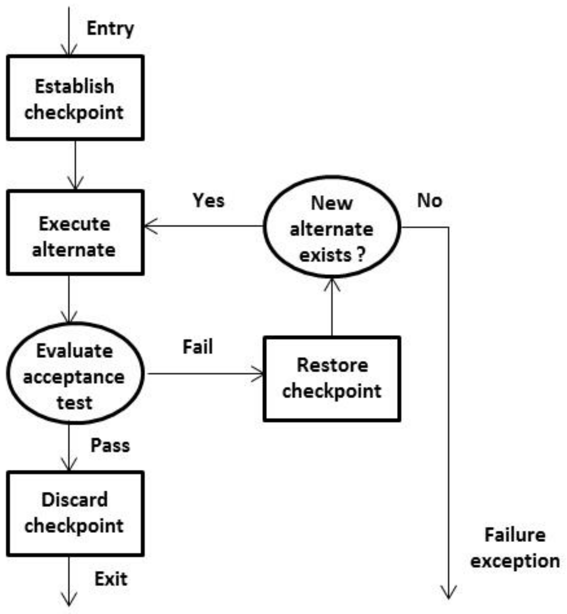


Figure 7: Operation of the recovery blocks

In some latest works, both N-version design and recovery blocks were included in the same global framework and used in multiple domains, e.g. firewalls design. While the essential conceptual elements of design diversity have remained stable over time, most subsequent works have focused on experimenting and quantifying the effects of this approach on fault tolerance.

Natural diversity

As stated earlier, this case covers the existence of different software that provide similar functionalities and emerging spontaneously from software development processes. For example, the programs that can be customized through several parameters embed a natural mechanism for diversification: two instances of the same program, tuned with different parameters can have different behaviours in terms of robustness and performance.

Software market is a strong vector driving the natural emergence of this type of software diversity, e.g. the business opportunities offered by the Web result in the existence of many competing browsers. These browsers are diverse in their implementation, in their performance, in some of their plugins, yet they are functionally very similar and can be used for one another in most cases. Other examples include operating systems, firewalls, database management systems, virtualisation environments, routers, middleware, application servers, etc. The following set of works illustrates natural diversity for different purposes:

- A mechanism for survivability monitors and controls a running application in order to tolerate unpredictable events such as bugs or attacks; the approach relies on fine grain customization of the different components in the application, as well as runtime adaptation.
- The existing diversity in router technology helps to design a network topology that has a diverse routing infrastructure; a novel metric to quantify the robustness of a network is used to compare different, more or less diverse, routing infrastructures.

- An intrusion detection mechanism leverages the natural diversity of COTS used for database management and Web servers which have few common mode failures and are thus good candidates for N-version design; the architecture is based on three diverse servers running on three different operating systems.
- "N-version protection" exploits the diversity of antivirus and malware systems and is based on multiple and diverse detection engines running in parallel; the prototype system intercepts suspicious files on a host machine and sends them in the cloud to check for viruses and malware against diverse antivirus systems.
- The diversity of software packages in operating systems is leveraged to increase the global resilience of a network of machines; the diversification of distributed machines allows the setting of a network tolerant to attacks.
- The use of different sequences of method calls in Javascript code, which happen to have the same behaviour, helps to harness the redundancy to setup a runtime recovery mechanism for Web applications.
- An intrusion avoidance architecture is proposed, based on multi-level software diversity and dynamic software reconfiguration in IaaS cloud layers; the approach leverages the natural diversity of COTS found in the cloud (operating system, Web server, database management system and application server), in combination with dynamic reconfiguration strategies.

Automated diversity

To this extent, N-version programming also achieves artificial diversity but the diverse program variants are produced manually. Automated diversity [i.25] rather emphasizes the absence of human in the loop and is in clear opposition to managed diversity. Different dimensions may characterize the various approaches to automated diversity:

- software systems are engineered at several *scales*: from a set of interacting machines in a distributed system down to the optimization of a particular loop;
- *genericity* explores whether the diversification technique is domain-specific or not, the *integrated* dimension is about the assembly of multiple diversification techniques in a global approach.

Randomization

The mainstream software paradigms are built on determinism. All layers of the software stack tend to be deterministic, from programming language constructs, to compilers, to middleware, up to application-level code. However, it is known that randomization can be useful to improve security: for instance, in compiler based-randomization, a compiler processes the same code with different memory layouts to decrease the risk of code injection.

How is randomization related to diversity? A randomization technique creates, directly or indirectly, a set of unique executions for the very same program. Randomized program transformations introduce diversity into applications. Execution diversity can be defined in a broad sense: it may mean diverse performances, outputs, memory locations, etc. (see Annex D).

Integrated diversity

This is about works that aim at automatically injecting different forms of diversity at the same time in the same program. In this context, the diversity is stacked or the different forms of diversity are managed with a specific diversity controller (see Annex D).

5.3.3 Software reliability evaluation and measurement

The introduction and the nature of faults in a program usually follow the different steps of the software lifecycle:

- analysis/requirements: if faults are introduced at this level, the software will not meet user expectations;
- design: a bad translation of requirements characterizes this faulty step;
- implementation/coding and verification/test: coding and bug fixing may lead to additional future defects;
- operations/maintenance: daily usage and software update/upgrade are potentially sources of troubles.

During the analysis, design and implementation steps, no failure is observed yet. The only elements which could be used for reliability prediction are the program structure and software metrics (e.g. number of LOC, cyclomatic complexity based on the control flow graph). One can thus estimate the "quality" of a program, but not its reliability, and there is no strict correlation between those two concepts. During the verification and operations steps, failures can be observed as the program runs, as well as corrections are made to debug or modify/patch it. Methods for software reliability evaluation and measurement are mostly based on observation and statistical analysis of these series of failures and corrections.

It is noteworthy that two directions can be followed when dealing with software reliability:

- Structural approach: this white box view considers a program as a complex system made of components; the program reliability depends on each individual component's reliability and the system's architecture (relations between the components).
- Black box approach: this global view takes the program as a single entity which cannot be decomposed.

Software reliability evaluation assesses the confidence level in a software-based system, i.e. the risk taken when running it. It also helps to ensure that the software has reached a certain level of reliability, as specified in the initial objectives, usually expressed in term of *failure intensity*. If reliability measures show that the objective is not reached yet, it may be possible to estimate the test effort, e.g. time, which needs to be provided in order to attain the target. As such, reliability measures provide criteria for stopping the tests, i.e. the tests end when one can prove, with a certain confidence level, that the reliability objective can feasibly be reached [i.33].

Software reliability is the probability that the software runs without failures during a certain time and in a certain environment: it is thus a temporal notion. The time considered can be the execution time (CPU), the calendar time, or even the number of operations or transactions. As software failures happen after the injection of bad inputs, testing all types/combinations of inputs will necessarily detect the potential problems. This is usually impossible as the entry space is too large: *operational profile* helps to define the inputs used and the execution frequency of the software components.

Homogeneous Poisson processes

Based on the exponential law which characterizes the absence of memory, these models are used for non-repairable systems which are not associated with wear and are not improved. This is the case of software if it is not corrected. Consequently, between two bug removal phases, *the failure intensity of software is constant,* and times between failures are random variables following the exponential law. It is noteworthy that the durations of unavailable software (a program stops running following a failure) are not taken into account because MTTRs are usually negligible. Figure 8 shows:

- the series T_i representing the time of failure i with $T_0 = 0$,
- the series X_i representing the time between two consecutive failures with $X_i = T_i - T_{i-1}$,
- the failures process counting with N_t = cumulative number of failures occurred between 0 and t.

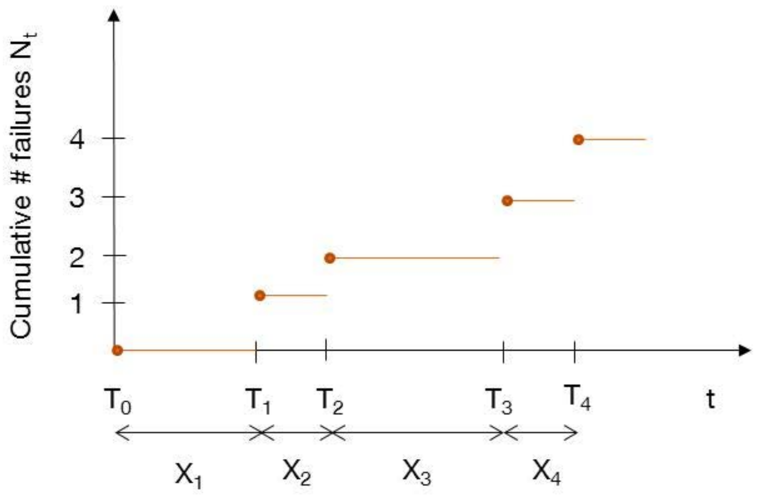


Figure 8: Failures process

The following formulas can then be calculated.

- The stabilized reliability is: $R(t) = \exp(-\lambda t)$
- MTTF is time-independent: $\text{MTTF} = \frac{1}{\lambda}$

It is noteworthy that Mean Time To Failure is appropriate in this case of non-repairable systems, but for the sake of simplicity, « MTBF » will not be used in the rest of this clause, even if the software is fixed after a failure has occurred.

- On average, the failure n will occur at time: $E[T_n] = \frac{n}{\lambda}$
- The mean number of failures occurred between 0 and t is proportional to t (as the failure intensity is constant): $E[N_t] = \lambda t$

As the parameter λ is used in all formulas given above, an estimation of its value is needed; using the times of failure obtained during test or operations for n failures, it can be shown that:

$$\hat{\lambda}_n = \frac{n}{\sum_{i=1}^{n} X_i} = \frac{n}{T_n}$$

NOTE: To use this model without failure data (e.g. a high quality software does not show any failure during the tests), the validation can be based on an upper limit of T_1, i.e. if no failure is found after running a software during T_1, this software is considered reliable enough. The value T_1 can be chosen based on the definition of a certain confidence level, e.g. with a risk of error of 5 %, T_1 needs to be three times the targeted MTTF.

Exponential Times Between Failures (ETBF) models

As in the previous case, the times between two successive failures are observed, and MTTRs are not counted. The software is fixed each time a failure has occurred, i.e. the independent X_i follow different exponential laws (as there is no wear out). As shown in Figure 9, the failure intensity follows a stairway function: a good debugging reduces λ_t (the first two repair operations of the figure), while a bad one increases it (third repair operation of the figure).

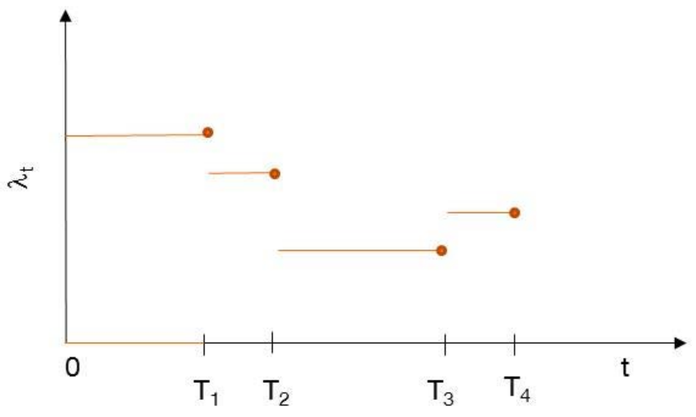


Figure 9: ETBF failure intensity

The following formulas can be calculated:

- Reliability and MTTF only depend on the number of (already) manifested failures:

$$R(t) = \exp(-\lambda_{n+1} t)$$
$$\text{MTTF} = \frac{1}{\lambda_{n+1}}$$

- On average, the failure n will occur at time:

$$E[T_n] = \sum_{i=1}^{n} E(Xi) = \sum_{i=1}^{n} \frac{1}{\lambda i}$$

For illustration purposes, two models of this kind (Jelinski-Moranda, geometric) can be found in Annex E.

Non-Homogeneous Poisson Processes (NHPP) models

These reliability models are used for repairable systems for which the failure intensity depends uniquely on time. As λ(t) is continuous, it does not change after a software correction (Figure 10). It may be the case if a program contains a huge number of (not severe) faults, of which each correction only removes a small amount.

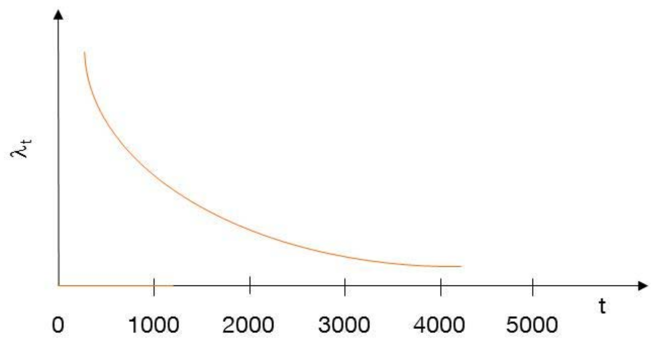


Figure 10: NHPP failure intensity

Annex E shows two types of NHPP models (Power-Law process, Goel-Okumoto), and presents as well some other models of this kind such as the S-shaped ones.

In summary, through program corrections and the use of failure data obtained from tests or operations, the (non-exhaustive) software reliability models sketched above allow the ability to estimate useful metrics such as R(t), MTTF, number of remaining faults, etc. The results can be used to provide criteria for stopping tests, planning maintenance actions, ensuring that SLA will be met, e.g. by computing the reliability of an end-to-end network service (composed of different VNFs, i.e. software).

Given the multiplicity of candidate models, the problem for practitioners is to choose among them the most appropriate to model a particular application by exploiting its failure data set. This choice may be based on the validation of the model's *hypotheses*, i.e. are they relevant to the study context? Another potential criteria is the model's *parameters*:

(i) are they appropriate to the analysis?

(ii) is it easy to estimate them with a high confidence degree?

Available *data* constitute the last decision criteria: statistical appropriateness checks are needed to verify the mapping between data sets and the reliability models.

6 Reliability/availability methods

6.1 Overview

6.1.1 NFV architecture models

Before the investigation of reliability/availability methods for NFV, NFV architecture model, its constituents and their relationships are identified.

To use a reliability model for a given NFV deployment, certain input parameters are needed to feed this model. Apart from the logical structure of the system that is to be modelled (including applied resilience patterns), an important parameter is the Mean Time Between Failures (MTBF) of the individual components of the system. These components include in the particular case of NFV the physical hardware resources, the elements of NFV-MANO used for, e.g. virtualisation, scaling or migration, as well as the VNF implementations themselves.

It is a non-trivial task to obtain valid and useful MTBF values, in particular due to the fact that estimations or measurements of failure rates are only really applicable in the context in which they were obtained. Small changes in the execution environment can render MTBF values irrelevant.

Reliability estimation is performed off-line for traditional PNFs during the network architecture design. A similar estimation can be conducted for the virtualised part of an NS. Although a reliability estimation can be calculated before service deployment, this estimation needs to be re-evaluated as the virtualised infrastructure may change during the service lifetime. In addition, in the NFV context, the dynamicity features allow the choice of virtual resources during some regular operational activities (e.g. scaling, migration), or abnormal events (traffic surge, natural disaster). To accommodate this possibility, MTBF values need to be accessible on-line.

From the viewpoint of NFV-MANO, there are two basic options to obtain these values as input for reliability estimation:

- The data are made available via the data repositories as part of NFV-MANO in the NFV architecture (Figure 11). In particular, the MTBF (and other reliability parameters, if available) of the physical components of the system as well as of the other (virtualisation) components of the NFVI are in this case part of the NFVI resources repository. Similarly, the same values for the VNF implementation should be part of the VNFD in the VNF catalogue. Here, the even more complex issue of software reliability is relevant, since VNFs are essentially software.
  This option basically enables feeding the input values to NFV-MANO from outside sources, such as the equipment or the software provider.
  Especially if new components are considered with which no experience exists in the scope of an NFV deployment, this information would need to be provided by such external sources.

- Alternatively, the data are gathered by NFV-MANO itself during service operation via a monitoring infrastructure. Once particular components have been in use for an extended amount of time, NFV-MANO might include a functionality that gathers statistical data about NS element failures (and the causes of such failures), which can then be used to estimate parameters like the MTBF more accurately. This functionality might come in the form of probes in the infrastructure and the VNFs, monitoring a set of parameters of interest for reliability estimation and reporting them to an estimation function within NFV-MANO, as discussed in ETSI GS NFV-REL 004 [i.9]. The monitoring information could be provided to the NFVO via the VNFM and VIM, since these are responsible for the collection of performance and fault information of their attached elements ETSI GS NFV-MAN 001 [i.5].
  An advantage of this approach is that the element reliability could be re-evaluated during runtime, based on updated measurements. If, for example, the hardware monitoring reports elevated temperatures, the failure rate of the affected elements could be increased to reflect a higher failure probability.

While such monitoring may also directly gather information about the distribution of the time intervals between failures and the time necessary to repair/fix the faulty elements, the monitored information might not be directly usable as input for the reliability models discussed in the following. Such indirect indicators include:

- In the case of hardware: bit error rates (BER) or Optical Signal to Noise Ratio (OSNR) for network resources; SMART values for storage components; temperature measurements, memory errors for compute resources.

- In the case of the virtualisation layer and the VNF: exceptions and warnings, dropped packets by the hypervisor, software response times.

These would need to be processed before being usable, i.e. mapped to the actual input parameters, e.g. MTBF or failure rates, of the reliability models. Since this mapping is a separate model highly dependent on the type of hardware used, information about this mapping should be provided also via the NFVI resources catalogue. These mappings and models could evolve over time, as more detailed information and measurements become available. When particularly relevant metrics are identified, they can be included in updated models by inserting a new version in the according catalogue. Thus, the reliability estimation can be fine-tuned over time using monitoring information.

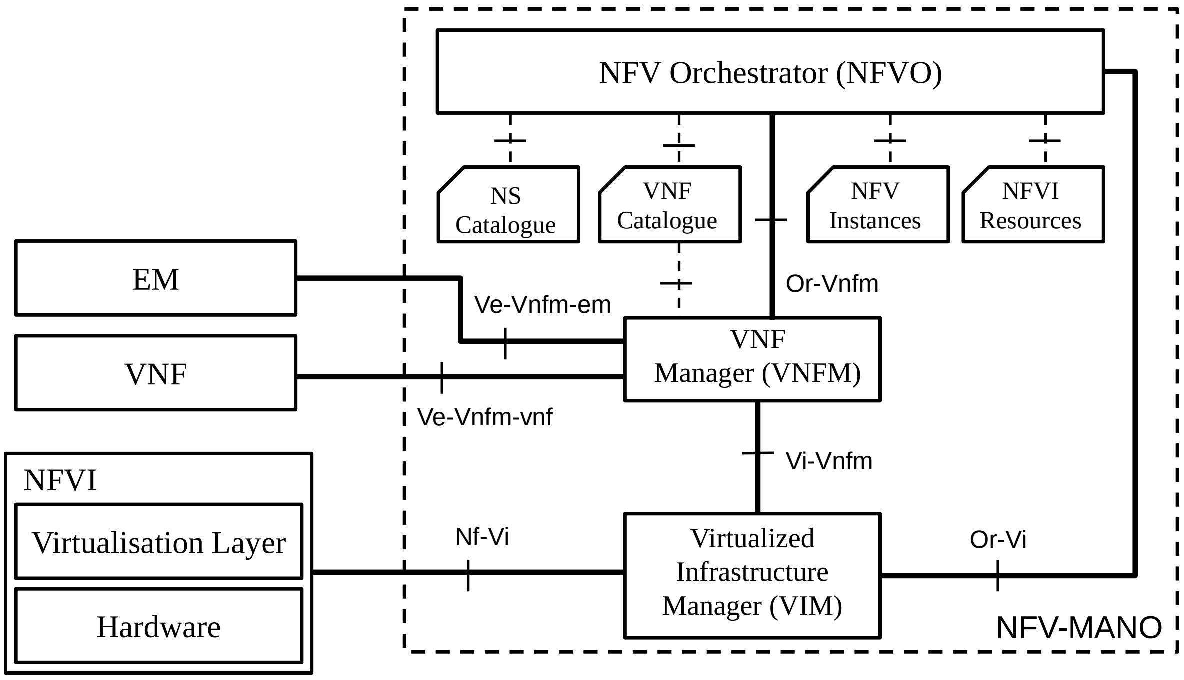


Figure 11: Parts of the ETSI NFV architecture relevant for reliability estimation [i.5]

The same is true for values monitored in the virtualisation layer and for the VNFs themselves, which should also be mapped using appropriate models to be able to derive an estimation of the failure rates. Here, these models would be part of the NFVI resources and the VNF catalogue, respectively. Care is to be taken to identify the root cause of a failure, since a VNF failure can be caused by an issue with the VNF itself, as well as by a hardware failure. Thus, a VNF failure that has as root cause a hardware failure should have no impact on the mapping of measured values to the reliability of the VNF implementation. The update of such a mapping should rather happen for the element(s) at the beginning of the failure process.

Additional relevant information to estimate the reliability of a network service (NS) is its composition in terms of VNFs used, their number and application of resilience patterns, or their logical sequence. This information should also be part of the NS description in the NS catalogue, at least with respect to the sequence and type of VNFs used. If NFV-MANO is not used to deploy resilience patterns by itself, the configuration of these patterns for individual NFs should also be part of the NS description.

If sufficient input parameters or estimations for these parameters with a fairly high level of confidence exist, NFV-MANO would be able to compute the reliability and availability of a NS deployment based on these values and on the current placement of the NS elements, using, e.g. the models described in clause 5.2.1 and clause 7. The opportunity to optimize this placement, keeping in mind other performance requirements, is thus given.

A more sophisticated use of NFV-MANO and reliability estimation is offered if NFV-MANO is also given the task to manage and orchestrate a NS with a minimum level of reliability and availability. In this scenario, another input to NFV-MANO would be these minimum values for a given NFV-based NS (as described in Req. 4.2.2 and 4.2.3 of ETSI GS NFV-REL 001 [i.2]). This would then be part of the NS data repository.

Some degree of freedom might exist with respect to the manner in which such a target threshold might be achieved by NFV-MANO. Depending on the level of detail of the NS description, it might already contain instructions about the resilience patterns to use and parameters like, for instance, the number of redundant paths. In this case, NFV-MANO might only be able to try to select and place the individual components in order to achieve the target value. If the NS description is more general, NFV-MANO might contain functionality like a deployment function that instantiates resilience patterns as necessary to achieve the prescribed target (e.g. instantiate more backup instances or larger load-sharing groups) - cf. clauses 6.2 and 7.1. Resilience patterns can only be instantiated according to the supported functionality of the VNFs, i.e. the mechanisms for failure detection and failover procedures need to be available to or instantiable by NFV-MANO. As an example, the state synchronization between stateful Active-Standby instances is typically VNF-dependent and needs to be provided by the VNFs themselves.

NFVO would then, based on the target reliability from the Network Service Descriptor and on the information of the VNFFGD, find a deployment and placement of the graph and applicable resilience patterns on the NFVI that satisfy the reliability requirements. It should be able to find this placement based on the reliability information about the NFVI and the VNFs, which would allow estimating the reliability for any given placement.

6.1.2 Network service elements

An NFV deployment generally consists of a composition of individual elements that work together to provide a network service. As principally shown in clause 5.1, these elements have to be taken into account when estimating the reliability of the service. The elements that are used in the reliability estimation are either hardware, virtualisation software, i.e. elements enabling virtualisation, or virtualised elements themselves (Table 1). For completeness, elements that are part of a network service, but are independent of the NFV deployment are also added to this list. From a modelling perspective, they can be treated like hardware in the sense that they are individual boxes with hidden internal dependencies

Table 1: Groups and types of network service elements for reliability estimation

Element group	Element type	Description
Hardware	Compute node	Hardware platforms, i.e. servers on which VNFCs (VMs or containers), controllers or NFV-MANO can run
	Storage node	Hardware platforms for storage solutions
	Network node	Hardware platforms for virtual networks, e.g. running software switches to manage networks between compute nodes and external networks
	Networking element	E.g. physical switches and routers, physical load balancers
	Physical link	LAN, MAN or WAN connections between physical networking elements, e.g. Ethernet
Virtualisation software	Hypervisor	Virtualisation environment running on a host
	Software switch	Switch implementations running on compute as well as networking nodes, e.g. vSwitch
	Network controller	Software responsible for the logically centralized network control if used for (virtual) network management, e.g. SDN controller
	NFVO	NFV-MANO part involved in all lifecycle operations of a NS
	VNFM	NFV-MANO part involved in all lifecycle operations of a VNF
	VIM	NFV-MANO part involved in all lifecycle operations of a virtualised resource
Virtualised element	VNFC	VNF implementations in VMs or container environment, VNF-internal load distributor
	Virtual link	Virtual connection between two VNFCs
Independent elements		Any element used which is independent from the NFV deployment, e.g. PaaS or PNFs

Each of the first three groups depends on the groups above it. While the hardware reliability does not depend on other elements (assuming no further level of detail is introduced, such as processors, memory, etc.), the virtualisation software needs to run on hardware and thus depends on such elements. Finally, virtualised elements need both the hardware they are running on, as well as the virtualisation software in order to work.

6.1.3 Characteristics of the networks adopting NFV concepts in terms of reliability and availability

As described in the clause 4.2, the traditional methods for estimating reliability can be used in the virtualised environment.

This clause shows the methods for making virtualised "chains of network functions" reliable on the COTS hardware.

A network service is composed of network functions (NFs), which are virtual network functions installed on virtualised machines over hardware resources, and of links connecting NFs (Figure 12).

NOTE 1: A network service can also include PNFs which are not represented here for the sake of clarity.

NOTE 2: Virtualised functions may run on other technologies than VMs, e.g. containers.

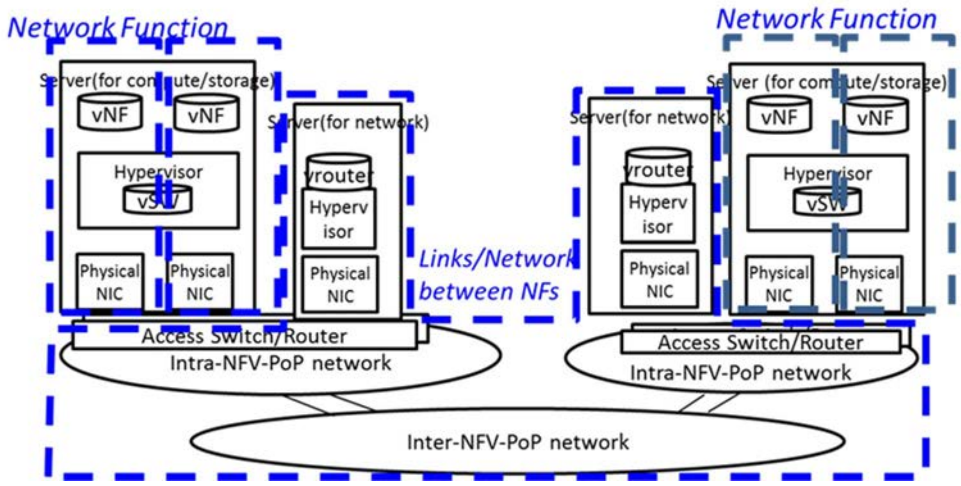


Figure 12: Network functions (NF) and links between NFs in NFV

The reliability of a chain of network functions is calculated by using the reliability of the network functions and links connecting them according to the graph structure of the network. This method is the same as for the traditional estimation of network reliability.

Reliability dependency among VNF, virtualisation layer and hardware

The difference of reliability estimation for virtualised networks and traditional networks mainly arise from the dependency of probabilities of failure for links and NFs. This is because VNFs can be implemented without knowledge of the hardware on which they are going to be deployed. For example, a failure of a physical server within the NFVI will cause service outage if an active VNF/VNFC and a standby VNF/VNFC reside on this server, even if this VNF is based on the Active-Standby redundancy (Figure 13).

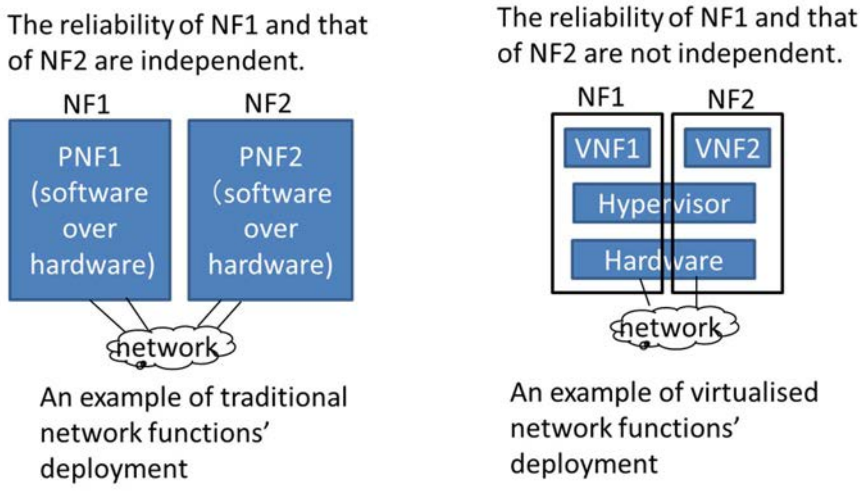

The reliability of NF1 and that of NF2 are independent.
NF1
NF2
PNF1 (software over hardware)
PNF2 (software over hardware)
network
An example of traditional network functions' deployment
The reliability of NF1 and that of NF2 are not independent.
NF1
NF2
VNF1
VNF2
Hypervisor
Hardware
network
An example of virtualised network functions' deployment

Figure 13: Difference of the situation between traditional network and virtualised network in terms of reliability

Software reliability involvement

Any NS element (i.e. NF) in an NFV deployment includes already three elements from the perspective of reliability (Figure 14): the physical hardware it is deployed on, the virtualisation infrastructure such as a hypervisor, and the VNF software implementation itself (including its VM or possibly container environment).

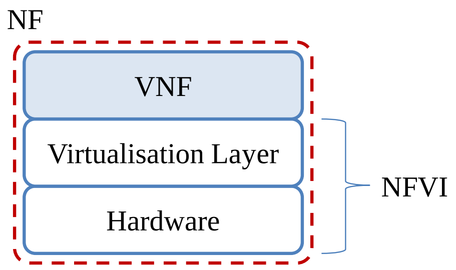


Figure 14: Elements of a functioning NF

However, since the combination of these components is only decided at the earliest during the instantiation of the VNF and may change during its runtime due to migration, reliability and availability should be treated at the level of the individual components rather than at the level of the composite. In terms of reliability block diagram modelling, this means that the three elements have a serial dependency (Figure 15).

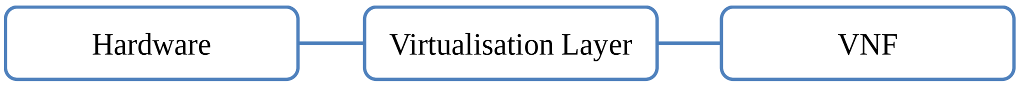


Figure 15: Serial dependence

To simplify the model, the failure causes and thus the failure probabilities for the individual elements might be treated as being independent, i.e. while a hardware failure also leads to the failure of a VNF running on that hardware, it is not included in the VNF failure causes. On the other hand, the availability of the VNF component itself is only based on the failure causes inherent to the VNF, and not on the availability of the underlying infrastructure.

Keeping in mind this simplification and carefully separating the failure causes, the availability of a NS element (network function) could theoretically be calculated as the product of the (independent) availabilities of the hardware, the virtualisation infrastructure based on that hardware and the VNF (clause 5.2.2).

NOTE 3: Once a VNF is launched, and until some particular lifecycle functions of NFV-MANO, e.g. used for scaling, migration, or failure restoration, are sought, the only NFV-MANO feature used is the one provided by the VIM for resources virtualisation, i.e. hypervisor in this example.

$$\mathrm{A_{NF}} = \mathrm{A_{Hardware}} \times \mathrm{A_{Hypervisor}} \times \mathrm{A_{VNF}} \tag{6.1}$$

However, several additional considerations should be kept in mind with respect to this formula. First of all, the availability as a single metric might not be of as much interest to cloud service providers and to infrastructure providers as some lower level metrics that contribute to it. Examples of such metrics for infrastructure providers are hardware failure rates, MTBF and MTTR - see also clause 5.2.1 for more information.

In the specific case of NFV, the Mean Time To Repair (MTTR), which is one parameter that contributes to the availability of a system, takes on a particular meaning. With the exception of a global failure of the NFVI, the repair process for the failure of hardware, virtualisation infrastructure or VNF instance begins with the instantiation of a new VNF instance on a different server (or on the same physical machine in case only the VNF instance has failed). If the instance is stateful, the restoration of the lost state would also be part of the repair process if supported (see e.g. clause 6.2.2.1.7). MTTR may also include the time to detect and isolate the failure, to trigger and to complete the restoration. MTTR depends on whether the VNF itself does implement failure detection, e.g. a heartbeat mechanism between Active-Standby redundant instances, or if NFV-MANO implements such mechanisms. Thus, MTTR is basically the sum of the failure detection time, the instance creation and configuration time. Depending on the type of instance, this MTTR can be very short (in the range of seconds), and thus the availability, as well as simplex exposure, for an Active-Standby instance should be by default better in a virtualised environment than in a traditional one (where repairs may take minutes to hours, or even days), assuming the same number of elements and the same MTBF. Further discussion of the MTBF, MTTR and their effect on availability and reliability in the relevant redundancy scenarios can be found in clause 7.1.

The failure rates, especially the hardware failure rates, are mainly of interest to the infrastructure provider to estimate how much hardware is needed in a pool in order to be able to have sufficient capacity for running services at any point in time. They also provide information on how often failover procedures are to be expected, and thus, the type and implementation of resilience patterns to be deployed.

In summary, MTTRs and failure rates are indicators for the ability of the infrastructure provider to prevent service outages, and, if they happen, to minimize service outage times and to provide backup capacities in the case of failed elements. This ability impacts directly service guarantees and potential SLA features the infrastructure operator could offer to a service operator.

An indicator that is more directly relevant for service operators is the reliability, i.e. the probability that an element or service works without failure for a time t. By a similar logic as for the availability, the reliability of a network function $R_{NF}(t)$ for a time t is given by

$$R_{NF}(t) = R_{Hardware}(t) \times R_{Hypervisor}(t) \times R_{VNF}(t) \qquad (6.2)$$

In a practical deployment, the highest variability of these three different components is caused by the variability of VNFs in term of reliability. The combination of hardware and virtualisation layer tends to be much more stable. Thus, the reliability of the virtualised infrastructure

$$R_{NFVI}(t) = R_{Hardware}(t) \times R_{Hypervisor}(t) \qquad (6.3)$$

might be considered, for practical reasons, jointly in many cases.

Similar considerations and variations of this basic concept apply for other combinations of components, e.g. having more than one dependent VNF on the same NFVI or introducing redundancy. These more complex cases and their effects are discussed in detail in clause 7.1.

Use of COTS hardware

Another difference is the use of COTS hardware which has a reliability lower than that of carrier grade servers, such as ATCA.

Reliability/availability methods, e.g. Active-Standby, Active-Active, etc. for NFs and link aggregation for links are studied, together with the factors and elements which are to be considered when the reliability of a virtualised network is estimated.

Lifecycle operations

Since what seems to be related to the reliability is "scale," "heal," and "software upgrade" from the viewpoint of lifecycle management, the procedures and mechanisms for these phases, such as failover, restoration and other functionalities, together with the impact on the NFV architecture, are investigated and recommendations or requirements are given.

NOTE 4: Software upgrade is mainly discussed in clause 8.

6.2 Network function

6.2.1 Introduction

As described in clause 6.1, the reliability and availability of an NF are dependent on the NFVI environment (Hypervisor and Hardware) where the VNF is instantiated. Since the reliability of COTS hardware is not very high, e.g. around three-nines, the reliability of the NF cannot be higher than that. Hence, protection schemes on the hypervisor or VNF level are needed.

Several reliability/availability methods that improve the reliability of network functions and how these methods relate to reliability and availability of the network function throughout its lifecycle are described in the following clauses.

Since the time required to recover from a failure state and to restore the redundant state is one of the important parameters to calculate the availability of network functions, the description from the fault management cycle's viewpoint will also be included.

Lifecycle operations

The lifecycle operations of a VNF are listed in ETSI GS NFV-MAN 001 [i.5]. These operations are: instantiate a VNF, query information on a VNF, scale a VNF, heal a VNF, update a VNF's software, modify a VNF, upgrade a VNF's software and terminate a VNF. The focus is on lifecycle operations that affect the end-to-end service reliability and availability.

Fault management cycle

Fault management phases, as described in ETSI GS NFV-REL 001 [i.2], are shown in Figure 16. In the following clauses about protection schemes, the healing (remediation and recovery) phase is investigated along this flow. This clause describes reliability features related to the reliability of VNF and NFVI in the context of fault management.

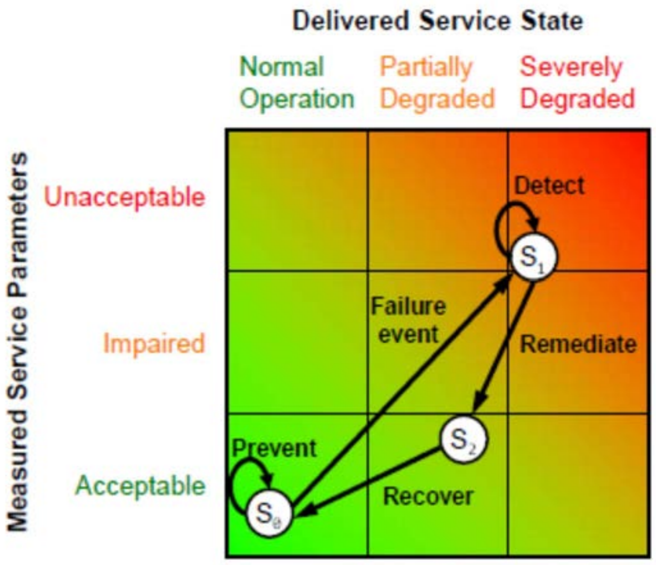


Figure 16: Fault management phases as described in [i.2]

For the following discussion, it is assumed that some protection schemes are provided to assure a certain level of reliability or availability.

Protection schemes

As protection schemes, [i.7] describes several methods listed in Table 2

Table 2: Protection schemes defined in SA Forum [i.7]

Scheme	Description
2N	One active service unit and one standby unit.
N+M	Multiple (N) active service units and one or more (M) standby service units that can take over from any of the active service units.
N-way	Service units can be active for some service instance and standby for other service instances.
N-way Active	Several service units can be simultaneously active for the same service instance.
No redundancy	Only a single service unit is assigned for service instance and no failover is provided.

In NFV environments, these protection schemes also work, and there are several implementations to realise these configurations. Several redundancy models that have different characteristics as examples are describved in the present document, and the procedures clarifying the relationship between the functional blocks within the NFV architecture from the standpoint of reliability and availability are investigated.

The selected examples are as follows.

- 2N: Active-Standby protection scheme. This is one of the most basic configurations for the redundancy.
- N-way: Load balancing protection scheme.
- N-way Active: Active-Active protection scheme.
- NONE: No protection is provided.

6.2.2 NFVI and NFV-MANO support for VNF reliability and availability

6.2.2.1 Mechanisms overview by fault management cycle phase

6.2.2.1.0 General

This clause provides a brief introduction of the mechanisms required for the NFVI fault management, arranged by the fault management cycle phase, and an overview of the roles and services that are expected from the NFVI and associated NFV-MANO management functions in the context of the NFV reference architecture.

The roles of the infrastructure services and their use with respect to redundancy modes and state protection mechanisms are also discussed for each phase.

6.2.2.1.1 Affinity aware resource placement

VM placement (or, in the context of NFV, VNFCI placement) is an infrastructure service provided by associated NFV-MANO entities (specifically the placement is done by VIM internal components, but under control of higher level information as specified through the associated VNF descriptors and/or APIs).

From the resiliency perspective, the relevant placement constraints include affinity- and anti-affinity rules, which specify the placement constraints with respect to the expected common mode failures in the underlying infrastructure. Affinity placement constraints generally dictate the placement of VMs (VNFCIs) associated with an affinity group onto same physical nodes, therefore placing them on the same fault domain (i.e. node in this case). Anti-affinity placement constraints seek to place the VMs (VNFCIs) of the anti-affinity group in a manner that reduces or eliminates common mode failures, e.g. to different compute nodes. Anti-affinity rules form the basic mechanisms for enforcing the placement of redundant entities in the NFVI such that the redundancy mechanisms can be expected to ensure that the redundancy is effective against failures in NFVI components. It should be noted that anti-affinity constraints could include more than just node level separation requirements (and being able to specify this is a requirement unless the design of the infrastructure guarantees that nodes do not have common failure modes that can affect more than single node at a time). Further anti-affinity constraints which seek to reduce the common failure modes would include placement constraints at enclosure, rack, room, or even facility levels (for geo-redundancy).

Both affinity-and anti-affinity constraints need to be retained and used beyond the initial placement decisions. Examples of the lifecycle operations that need to take these constraints into account beyond the initial placement include recovery operations, scaling operations for the VNFCIs that are part of the scale in/out groups, and any re-placement operations during the VM lifecycle for any cause (particularly migration related operations, which may be for operational reasons such as for node software upgrades, or management/optimization reasons such as for performance or power management). In addition to the on-demand (whether by VNF, NFV-MANO or orchestration processes) lifecycle management actions, the affinity constraints also need to be met with the deferred placement decisions associated with the resource reservation mechanisms for the future.

In addition to the VM resource placement, affinity rules may need to be extended to the placement of other resources, including the network or even storage resources.

An example where network level anti-affinity service would be required by a VNF (or associated VNFCI) is a situation where the application wants to ensure the availability of a critical communication path between the two application components through redundant application level interfaces to the underlying NFVI network. In such case, unless the network path anti-affinity can be provided as a service for the associated redundant channels, the channels would not result on truly redundant independent channels, but could be subjected to same common mode failures in the underlying infrastructure. Note that this case is expected to not be required if the underlying NFVI network infrastructure and its associated management (particularly fault management) processes are sufficiently reliable and fast to provide the associated protection services as infrastructure service, which is the state architectural goal of the NFVI infrastructure implementation in NFV phase 1 documents.

While affinity related rules are key placement constraints from the resiliency point of view, it should also be noted that there are other potential constraints that are required in the placement decisions, including but not limited to, node resource constraints (in terms of types and amounts in addition to, e.g. instance memory capacity requirements needed for acceleration functions, instruction set supports, etc.), performance related constraints (with regard to compute, network, etc. resources), and infrastructure optimization related constraints, which need to be taken into account on the VM placement. All such applicable constraints need to be resolved jointly in the constraint based placement decisions by NFV-MANO placement related processes. VNFMs will require notifications from the placement operations (e.g. in terms of success and failure) including, but not limited to, availability constraints to be able to take actions (such as escalate to orchestration level) in cases of placement failures.

6.2.2.1.2 State protection

State protection mechanisms are mechanisms and services to facilitate the protection of the critical internal state of the VNFs and/or their constituent VNFCIs. State protection of "stateful" is a pre-requisite for service continuity, and some state protection is also generally required for the service availability as well (such as protection of configuration state). In traditional NF software implementations state protection is fully performed by the application software (possibly through some "middleware" services which are utilized by the application processes). A common example of the basic structure of the state-protected application process would be an Active-Standby process pair where the application processes (and their encapsulation virtualisation containers in the cloud) are instantiated on anti-affinity configuration, and the protected state is copied from the active process instance to the associated standby process instance.

The four following state protection processes are applicable in the clouds.

- Application level state checkpointing to external state repository: such external state repository can be another application component instance, e.g. application level database service.

- Application level state checkpointing to redundant application process (direct application checkpointing).

- Infrastructure level, full VM state protection service, where the infrastructure provides state protection for full VM state as a service for the application. This state protection service does not require application level code for the state replication; however, the application needs to be able to request to be instantiated in this mode from the infrastructure.

- Infrastructure level application aware partial VM state protection service, where the infrastructure provides state replication service, but instead of complete application state within VM, only a specific subset is protected. The implementation of this mode requires at the minimum the availability of information from the application to describe the protected subset, and may require other services, such as entry points for the standby activation in a manner that takes into account that non-protected subsets of the replica state may not be there.

Combinations of the above mechanisms are also possible: a common case would be to combine external storage based state protection with persistent storage to support, e.g. protection of configuration state elements with checkpointing mechanisms to protect more transient states.

All of the state protection mechanisms listed above will at least partially rely on the NFVI services (and respectively, their associated NFV-MANO provided management services and APIs) as follows:

- The application level state protection of external repository will use the NFVI network services and resources for the communication between the application instance and the external repository. This is not necessarily unlike any other VNF internal communication channel, and requires minimum capabilities for the establishment of such virtual communication channel (or network) as well as NFVI and NFV-MANO processes for performance monitoring and fault management functions for the associated channel(s), notification of the channel related events, including the fault management notifications. If the repository is implemented using NFVI storage resources, then associated storage interfaces and services (including storage state replication services) and associated network communication channels will be provided by the NFVI and NFV-MANO services, which are also expected to be responsible for monitoring and fault management actions with respect to such services, and notifications of the fault events through the VNFM.

- The application level state protection for redundant application process instances will utilize the NFVI network service as a communication channel for the state checkpointing process. Network level fault management services and notifications are expected to be provided as NFVI and NFV-MANO services as above.

- The full VM level state protection service is performed by the infrastructure as a service for the requesting application VM. This service utilizes node level resources for state replication between the active and standby instances and, like other state protection services, also utilizes underlying NFVI network services to establish and maintain the associated communication channels between the protected and protecting entities. "Checkpointing with buffering" mechanism as described in [i.8] is an example of transparent, full VM level state protection mechanism, and "checkpointing with replay", also described in [i.8], is an example of VM state protection mechanism which improves latency properties of most of the through-traffic, at the expense of recovery time and not being fully application transparent (i.e. like for application level state protection mechanisms, source level access to application code is required to implement this mode).

- The partial VM level state protection service is similar to full VM level state protection service, with the exception that this service only replicates a subset of the VM state. Partial replication service requires information to specify the state subset (e.g. memory areas) subject to protection. It also requires that an associated standby instance can support activation process (failover or switchover) starting from the partially replicated VM state.

The state protection service(s) used affect the associated remediation and recovery actions, and the associated impacts are discussed in the subsequent sections for the respective phases.

6.2.2.1.3 Failure detection

Failure detection mechanisms in traditional designs rely heavily on the direct access to the underlying hardware resources. Hardware related failure detection in traditional systems implement various detection mechanisms at the device driver layer, generally implemented as "hardened" device drivers.

This is not possible in NFV deployments, as the application processes are decoupled from the underlying hardware. This implies that the direct access to the hardware is encapsulated by the hypervisor. In addition, the underlying hardware resources are expected to be shared by many different VNF instances. Even in case of hardware acceleration, the direct hardware access from the VNFs is effectively hidden. This abstraction has several implications for the VNF application failure detection design. Most importantly, if a VNF requires indications of failures of the underlying hardware, it needs to subscribe to fault management notifications from the NFVI. The NFVI will provide the associated failure indications by standardized means within the context of specific resource functionality and failure modes. In addition, there will be failure detection mechanisms internal to the VNF/VNFC instances. Generally, these are associated with mechanisms that are application design/application function specific, such as protocol mechanisms related to the exogenous failures from the VNF/VNFC perspective, as well as VNFC internal health state monitoring mechanisms that are implementation specific (such as underlying process state monitoring services).

6.2.2.1.4 Localization

Localization operations are a collection of the infrastructure services (performed jointly by the NFVI and NFV-MANO components), which have the goal to rapidly determine the underlying fault cause and/or scope to the extent required to determine the fault management action to be taken on subsequent fault management processes, especially in isolation and remediation phases.

Since the goal of the cloud infrastructure is to isolate the guests from the infrastructure, these guests (i.e. VNFCIs composing a VNFI), or their associated VNFM will not have neither access to all of the fault indications for the resources impacted by specific failures, nor do they have topological view of the infrastructure to support cause determination: localization functions will need to be implemented as infrastructure services in combination with the NFVI and NFV-MANO functions. Localization processes typically involve passive mechanisms such as fault correlation from various infrastructure (NFVI) and NFV-MANO fault detectors in the context of the known infrastructure topology and other attributes, and may also include active mechanisms (such as active probing for resource states) to support localization processes.

Localization results are expected to be made available to the VNFMs of the VNFs that are determined to be affected through fault management notification mechanisms that the VNFMs can subscribe to. This is an essential requirement for all failure modes that are expected to involve VNF level remediation/recovery mechanisms in the later stages of the fault management cycle. If the fault management remediation and recovery actions can be taken solely as autonomous infrastructure service (i.e. masked from the VNFIs or its constituent VNFCIs), it is not necessary to make localization results available to VNFMs.

The localization phase can be skipped when the associated underlying detection mechanisms are able to unambiguously determine the nature and the scope of the fault to support the actions to be taken in later fault management cycle phases, such as a node-local resource fault. Localization processes are commonly required to determine the cause of more complex faults involving multiple co-operating subsystems, particularly with respect to the physical and logical interfaces spanning the node boundaries, such as network, storage services and other common shared services faults.

6.2.2.1.5 Isolation

Fault isolation mechanisms are used to ensure that the faults and/or erroneous system state do not propagate across the subsystem boundaries, as well as to prevent simultaneous activation of the active and standby processes (also referred as "split-brain avoidance").

Fault isolation mechanisms in the clouds include virtual machine isolation, reset and "virtual power off", node level power controls, and VM isolation from the NFVI infrastructure networks. While certain fault isolation mechanisms that are limited to the scope of individual VMs can be used through a request of the VNFM, provided that they do not have capability to unintentionally affect other VMs or other resources that may share the same physical node, some of the mechanisms are only available as infrastructure initiated actions (e.g. node power off cannot be initiated by individual VNF/VNFM, as it would not meet the constraint that the action may not affect VMs that are not part of the initiating VNF task). As such, fault isolation mechanisms need to be either implemented as infrastructure fault management service initiated actions, with fault management notifications to the affected VMs, or as a combination of the infrastructure initiated and VNF initiated actions (constrained to mechanisms limited to VNFs resources).

Regardless of the implementation of the associated control logic for the fault isolation process, the mechanisms themselves are implemented in the NFVI components, and are exposed through the NFV-MANO APIs available to the infrastructure fault managers, with potential subset available to the VNFMs.

The on-demand isolation processes implicitly rely on the availability of the communication channels to perform isolation operations (such as node power off). As these channels or associated resources/supporting software processes in the NFVI components used to perform fencing operations may fail (either as a symptom of the same underlying failure or separately), there should be supplementary fail-fast implementation that performs the complementary actions to isolation processes, such as node level watchdog timers causing resets. In addition, the reliable isolation mechanisms should be able to utilize multiple layers of complementary isolation mechanisms, such as node power off first and upon failure of this process attempt to use next level mechanisms such as isolation of the node from the system network(s) in associated switches.

6.2.2.1.6 Remediation

Remediation actions are a collective set of processes to get the service provided by failed entities back to normal, in-service state. Remediation actions typically involve failover of the affected services to protecting resources in the case redundancy is utilized or, in the non-redundancy configurations, re-instantiation or restart of the failed VNFCI.

The set of remediation processes utilized and location of the associated services are primarily determined by the following three key aspects of the VNFC configuration:

- VNFCIs association with system network, including network interfaces, virtual network topology, pre-computed communication paths (if used), entities such as load balancers and other such elements required to (re)direct the load to protecting VNFCI(s) from the failed VNFCI.

- Redundancy model utilized, e.g. no redundancy, Active-Standby, Active-Active, n actives with m standbys (generic case for describing active standby topology configurations), n-way active, n+m active (generic case to describe Active-Active configurations).
- State protection model utilized, e.g. no state protection "stateless", externalized state protection (protection to external state repository, including persistent storage), direct Active-Standby checkpointing, and direct VM level checkpointing.

The generic high level sub-flow used in the remediation steps for redundancy modes with standby entities is as follows:

- Activate redundant standby VNFCI, if not already running.
- Move network and storage connections to the new active VNFCI.
- Retrieve instance state from external repository if applicable (for "externalized state" VNFCIs).

6.2.2.1.7 Recovery

The recovery processes will work to restore the intended system configuration after the occurrence of faults and failures, including the full redundancy configuration, when applicable.

Unlike in the physical systems, recovery operations in the cloud can take advantage of the resource pooling properties of the cloud to decrease substantially the recovery time, especially as compared to recovery operations requiring replacement of physical subsystems. For example, upon determination that a compute node has failed, the recovery mechanism does not need to wait for the replacement of the physical node, but can request a new node be allocated from the cloud resource pool.

Like for the remediation operations, the generic flow for the recovery operations depends on the network configuration, redundancy model, and state protection models utilized.

The generic high level sub-flow used in the recovery steps for redundancy models with standby entities is as follows:

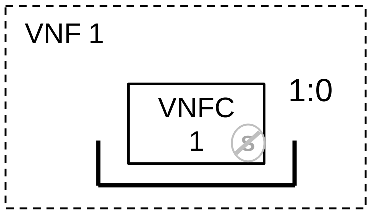
- Locate or request a new redundant resource to be allocated for hosting each affected VNFCI.
- Establish network and storage connectivity to new redundant VNFCIs through the NFVI network, as required to support VNFCI operation in standby state.
- Synchronize the redundant instance states with the active resource states, or establish the connectivity to the external state repository, if applicable (for "stateful" VNFCIs).

6.2.2.2 Non-redundant/on-demand redundant VNFC configurations

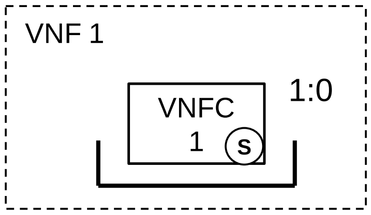
The non-redundant configuration is the simplest configuration with respect to the redundancy. In this configuration, there is no pre-established redundant entity to fail to.

The availability performance of the non-redundant configurations can be improved by fault detection, combined with automatic re-instantiation of the failed entity upon failure. Since there is no pre-instantiated and active redundant entity in place, direct state replication for stateful functions is not possible and, therefore, only state protection mechanisms that can be used in these configurations would require the use of external state repository.

The following configuration examples focus on the configurations where the remediation actions are taken as an infrastructure service. To distinguish from the typical Active-Standby schemes where the standby is pre-established, 1:(1) notion is used here to indicate the on-demand nature of the standby method (vs. 1:1 notation for warm standby).



Stateless VNFC with on-demand standby



Stateful VNFC with on-demand standby

Figure 17: 1:0 Active - On-demand standby with no VNFCI state protection; stateless re-instantiation remediation

Two scenarios are depicted above, with "stateless" and "stateful" VNFCs. As there is no state protection in either of these cases, any state that may be established during the VNFC operation will be lost upon VNFC failure. When the VNFC fails, and after its re-instantiation is completed, the operation of the new instance starts from the pre-established "reset" state, which is independent of the state of the failed instance (beyond the instantiation-time initial state).

The different characteristics of this configuration are as follows:

Placement: VNFCs can be freely placed on the NFVI by NFV-MANO without the concern of anti-affinity constraints as they are considered to be independent entities because there is no concept of VNFCI protection in this mode.

State protection: the VNFC state protection is not applicable.

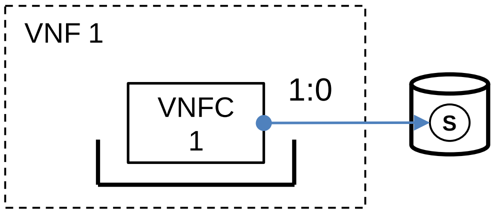
Fault detection: NFVI detects the faults on the NFVI.

Fault localization: NFVI and NFV-MANO perform the required localization actions.

Fault containment: NFVI and NFV-MANO perform the required containment actions. Depending on the specific failure mode and its associated scope, containment may include powering off the failed nodes and/or network reconfiguration actions.

Fault remediation: NFV-MANO performs VM re-instantiation on failure. Supplementary actions may be required, e.g. network and/or storage association reconfiguration.

Fault recovery: it is equivalent to remediation (see above) in this mode.



VNFC with on-demand standby and externalized state protection

Figure 18: 1:0 Active - On-demand standby with externalised state protection; stateful re-instantiation remediation

In this configuration, the VNFC internal state can be protected by storing it in an external state repository. When the VNFC fails and after the instantiation of a new instance is completed, the operation of the new instance starts from the pre-established "reset" state and, after that, VNFCI can retrieve any critical state stored by the failed instance up to the point of its failure from the external state repository.

The different characteristics of this configuration are as follows:

Placement: VNFCs can be freely placed on the NFVI without the concern of anti-affinity constraints when they are considered to be independent entities, as there is no concept of VNFCI protection in this mode. Depending on the nature of the externalized state repository, its placement may be subject to explicit or implicit (e.g. externalized protection that uses external storage nodes) anti-affinity requirements with respect to the VNFCI placement.

State protection: VNFC state protection is the responsibility of the VNFCI. State protection is done by an externalised entity, which may be (or utilizes) storage service provided by the NFVI, or another VNFC that is provided by application (or combination thereof).

Fault detection: NFVI detects the faults on the NFVI.

Fault localization: NFVI and NFV-MANO perform the required localization actions.

Fault containment: NFVI and NFV-MANO perform required containment actions. Depending on the specific failure mode and its associated scope, containment may include powering off failed nodes and/or network reconfiguration actions.

Fault remediation: NFV-MANO performs VM re-instantiation on failure. Supplementary actions may be required, e.g. network and/or storage association reconfiguration. Restoration of the state from the externalised state repository before transitioning to operational state is the responsibility of the re-instantiated VNFCI.

Fault recovery: it is equivalent to remediation (see above) in this mode.

6.2.2.3 Active-Standby VNFC redundancy configurations

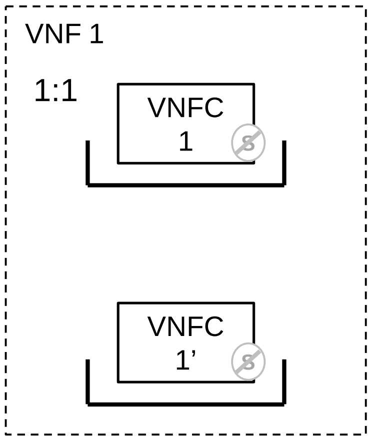
Active-Standby VNFC redundancy configurations are configurations where active entities are running the application component software and providing application specific service, and these active elements are protected by one or more standby entities, which are not actively providing service. These configurations have a number of general basic properties, such as:

- There are standby resources, typically one for each active instance, but relationship can also be multiple actives protected by one or more standby instances, which reduces the amount of required resources that need to be committed for redundancy support.
- As standby resources are not performing application load processing, they may require implementation of additional exercise processes and specific software for the latent fault detection purposes to ensure that the standbys are ready to transfer to service upon the failure of the associated active resource.
- In the 1:1 case, failures affect the full processing capacity associated with the active entity, which implies that during the failover, the service provided is out of service. This requires the minimization of the fault management cycle time for maximizing the service availability and continuity.
- Failures do require system network reconfiguration to direct traffic to standby entities.
- The 1:1 Active - Standby configuration does not require a load distribution function in front of the pooled active resources. Simple network capability to switch the traffic from the active entity to the standby entity is sufficient.

The following common cases can be identified:

- 1:1, single active protected by single standby, no performance degradation for a single fault.
- N:1, N actives protected by single standby, no performance degradation (<N) for a single fault.
- N:M, N actives protected by M standbys, no performance degradation (<N) for up to M simultaneous faults.

N:M is the generic configuration which can be used to describe and instantiate any of the specific case configurations listed. However, for stateful entities with direct state replication, this reverts back to multiple n+1 configurations, as in such case, there generally needs to be predetermined state protection sender-receiver relationships.



Stateless VNFC with warm standby

Figure 19: 1:1 Active-Standby with no VM state replication; stateless failover remediation

The different characteristics of this configuration are as follows.

Placement: VNFCs of the redundant pair need to be placed on different hardware servers with no or limited common failure modes.

State protection: VNFC state protection is not applicable.

Fault detection: NFVI detects the faults on the NFVI.

Fault localization: NFVI and NFV-MANO perform required localization actions.

Fault containment: NFVI and NFV-MANO perform required containment actions. Depending on the specific failure mode and its associated scope, containment may include powering off failed nodes and/or network reconfiguration actions.

Fault remediation: NFVI performs VM failover on the hypervisor layer. Supplementary actions may be the responsibility of NFV-MANO (e.g. network reconfiguration).

Fault recovery: NFV-MANO assigns the replacement of the failed node from the cloud resource pool as a new standby entity. NFV-MANO is then responsible for the on-demand diagnosis of the candidate failed entities, and initiation of any subsequent physical recovery request actions for entities with confirmed persistent faults.

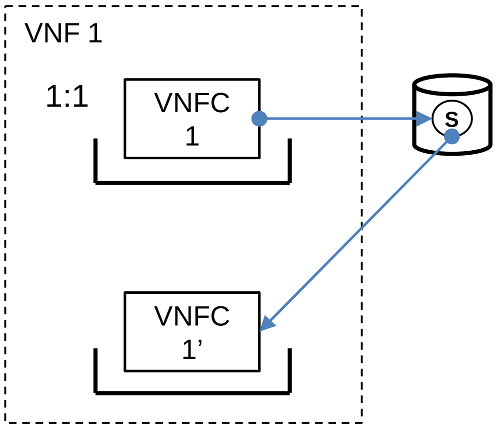


Figure 20: 1:1 Active-Standby with externalised VM state replication; stateful failover remediation

The different characteristics of this configuration are as follows:

Placement: VNFCs of the redundant pair need to be placed on different hardware servers with no or limited common failure modes. Depending on the nature of the externalized state repository, its placement may be subject to explicit or implicit (e.g. externalised protection that uses external storage nodes) anti-affinity requirements with respect to the VNFCI placement.

State protection: VNFCI performs partial VM state replication for its critical state to external state replica repository. This state replication may be VNFC vendor proprietary or utilize 3rd party or open source middleware services. From the NFVI perspective, the state replication process is opaque, and the NFVI involvement in the replication process is limited to provisioning of the associated NFVI network domain resources for the state replication communications and the fault management of the provided underlying network connectivity elements.

Fault detection: NFVI detects the faults on the NFVI.

Fault localization: NFVI and NFV-MANO perform required localization actions.

Fault containment: NFVI and NFV-MANO perform required containment actions. Depending on the specific failure mode and its associated scope, containment may include powering off failed nodes and/or network reconfiguration actions.

Fault remediation: VNFCI performs VM failover to standby. Stateful fault remediation requires that the standby node is brought to the state that is consistent with the state of the external state repository. This process may be either reactive (the state is retrieved only during the failover) or proactive (the state is retrieved during operational phase and failover can commence when the whole state is available in the standby). Supplementary remediation actions may be the responsibility of NFV-MANO (e.g. network reconfiguration).

Fault recovery: NFV-MANO assigns the replacement of the failed node from the cloud resource pool as a new standby entity. NFV-MANO is then responsible for the on-demand diagnosis of the candidate failed entities, and initiation of any subsequent physical recovery request actions for entities with confirmed persistent faults.

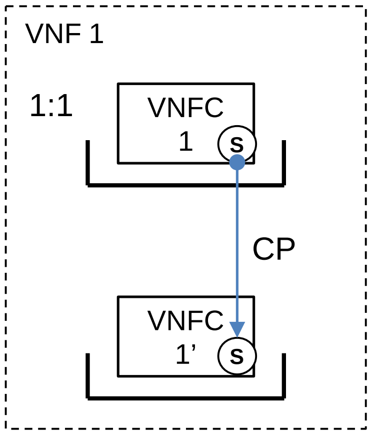


Figure 21: 1:1 Active-Standby with direct partial VM state replication; stateful failover remediation

The different characteristics of this configuration are as follows:

Placement: VNFCs of the redundant pair need to be placed on different hardware servers with no or limited common failure modes.

State protection: VNFCI performs partial VM state replication for its critical state. This state replication may be VNFC vendor proprietary or utilize 3rd party or open source middleware services. From the NFVI perspective, the state replication process is opaque, and the NFVI involvement in the replication process is limited to provisioning of the associated NFVI network domain resources for the state replication communications and the fault management of the provided underlying network connectivity elements.

Fault detection: NFVI detects the faults on the NFVI.

Fault localization: NFVI and NFV-MANO perform required localization actions.

Fault containment: NFVI and NFV-MANO perform required containment actions. Depending on the specific failure mode and its associated scope, containment may include powering off failed nodes and/or network reconfiguration actions.

Fault remediation: VNFM or VNF performs VM failover initiated by application. Remediation actions can also be split between the NFVI and VNF, e.g. NFVI may be fully responsible for the network fault remediation, while VNF may use network APIs at the NFV-MANO layer to request network reconfiguration as part of specific VNFC failure remediation cases. VNF/VNFM is responsible for starting the application in the state reflecting the replicated protected state.

Fault recovery: NFV-MANO assigns the replacement of the failed node from the cloud resource pool as a new standby entity. VNFC is responsible for the state replication to bring the new standby up to date with the active state, which restores the redundancy configuration. NFV-MANO is then responsible for the on-demand diagnosis of the candidate failed entities, and initiation of any subsequent physical recovery request actions for entities with confirmed persistent faults.

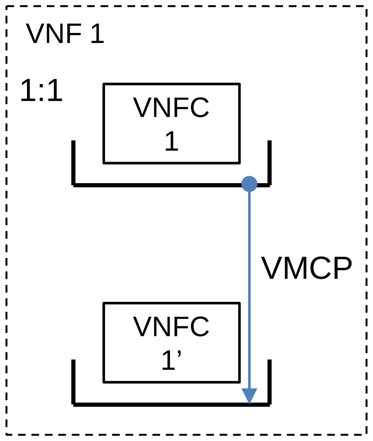


Figure 22: 1:1 Active-Standby with full VM state replication; fully stateful failover remediation

The different characteristics of this configuration are as follows:

Placement: VNFCs of the redundant pair need to be placed on different hardware servers with no or limited common failure modes.

State protection: NFVI (specifically hypervisor) performs full VM state replication, including full VM execution state replication as a platform service.

Fault detection: NFVI detects the faults on the NFVI.

Fault localization: NFVI and NFV-MANO perform required localization actions.

Fault containment: NFVI and NFV-MANO perform required containment actions. Depending on the specific failure mode and its associated scope, containment may include powering off failed nodes and/or network reconfiguration actions.

Fault remediation: NFVI performs VM failover by hypervisor layer. Supplementary actions may be the responsibility of NFV-MANO (e.g. network reconfiguration).

Fault recovery: NFV-MANO assigns the replacement of the failed node from the cloud resource pool as a new standby entity. The NFVI layer is responsible for the state replication to bring the new standby up to date with the active state, which restores the redundancy configuration. NFV-MANO is then responsible for the on-demand diagnosis of the candidate failed entities, and initiation of any subsequent physical recovery request actions for entities with confirmed persistent faults.

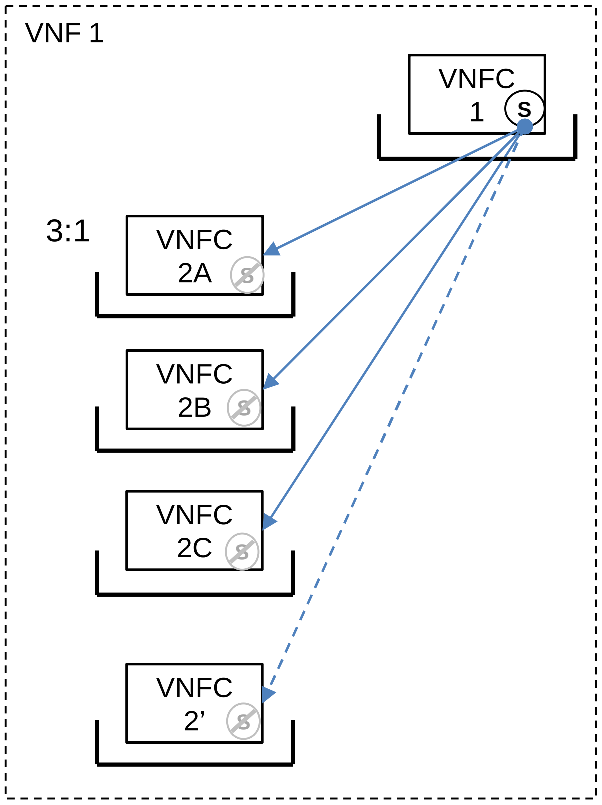


Figure 23: n:1 (3:1 shown) Active-Standby with externalised state push/pull (no state replication of underlying VNFCIs); remediation to state determined by controlling VM

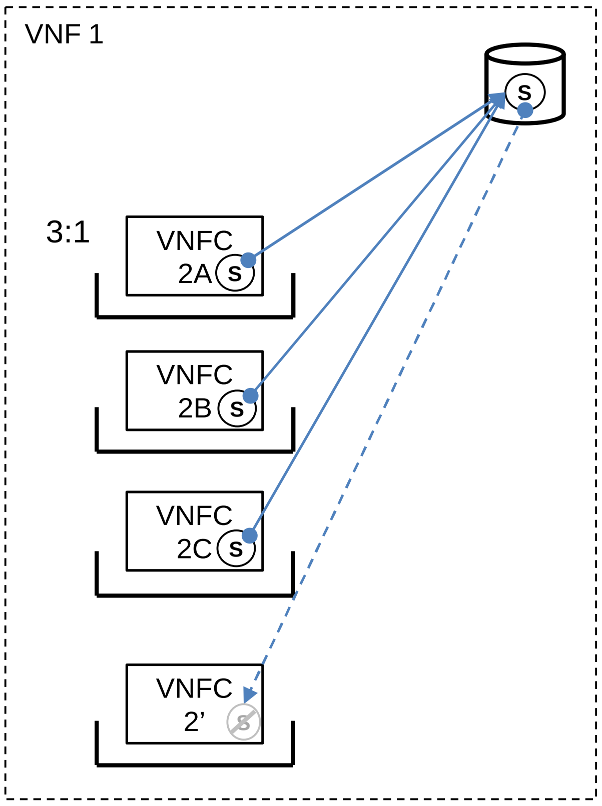


Figure 24: n:1 (3:1 shown) Active-Standby with externalised partial VM state replication; stateful remediation

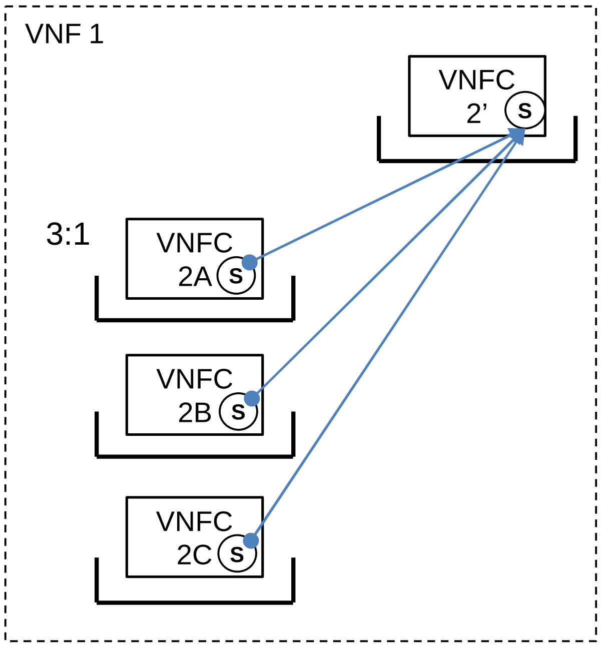


Figure 25: n:1 (3:1 shown) Active-Standby with direct partial VM state replication; stateful remediation

In this configuration, there are n actives (three shown for 3:1), each of which replicates their internal critical state to a single shared standby instance. The potential drawback of this configuration is that the standby instance needs to have sufficient memory capacity to hold n times the critical state of the active instances, which may be problematic especially for the configurations with large n and/or when combined with large critical state blocks.

A generalized version of this configuration would be n:m, where there are n active entities, and m standby instance entities. Such configuration can be useful for, e.g. dealing with potential memory capacity issues associated with the baseline n:1 configuration. However, in the cases where the state replication relationship is always n:1 (i.e. the specific state is only replicated to one entity), this essentially transforms to m independent configurations of n:1, possibly with differing 'n' for each of m configurations (esp. for cases where n is not evenly divisible by m).

6.2.2.4 Active-Active VNFC redundancy configurations

Active-Active VNFC redundancy configurations are configurations where all the entities are actively running the application component software and, therefore, are considered to be active with respect to the application processing. As compared to Active-Standby configurations, these configurations have number of attractive properties, such as:

- There are no dedicated standby resources, all nodes are running active - this implies that the subsystems on the nodes are effectively exercised by the application processing and, therefore, active mode fault detection mechanisms can be employed for the fault detection tasks without the need to rely on the additional exercise processes and specific software for the latent fault detection purposes.
- Actives can be pooled to perform more processing, and aggregate processing capacity can be dynamically scaled out/in based on the load variation.
- When active elements are used in N+M configurations (esp. N+1), the amount of committed resources associated with the redundancy can be substantially reduced as compared to common 1:1 configurations.
- Failures only affect fraction of the processing capacity, which can be small for larger N (in generalized N+M case).
- Failures do not require system reconfiguration to direct traffic to standby entities, instead the failure remediation relies directing traffic out from the failed instances and to remaining operational nodes. These processes can in many cases be faster than comparable redirection mechanisms for Active-Standby configurations, especially when a dedicated load balancer entity is used for the traffic distribution.

However, Active-Active redundancy configurations also represent certain specific challenges as compared to the 1:1 configurations:

- Active-Active configuration requires load distribution function in front of the pooled active resources. This load distribution function can be any set of network mechanisms of varying complexity, such as semi-static association of the subscribers to processing pool resources, L2/L3 address based associations, hash and other ECMP style load distribution mechanisms, L4 and higher layer load balancers/application delivery controllers.
- The load distribution function itself needs redundancy and may also require state replication in the case of session-stateful load balancing, to avoid load distribution becoming a single point of failure.
- The internal state replication for the Active-Active pools is more complex, and may require state partition that is coordinated with the properties of the load distribution function to ensure that the stateful sessions are associated with the active resource(s) containing the state associated with the failed entity.
- The fault detection for active pool members requires direct fault detection from the load distribution function or, alternatively, load distribution function needs to be reconfigured based on the detected faults.

The following common cases can be identified:

- 1+1 Active-Active
- N+0 active, performance degradation (<N) for a single fault
- N+1 active, no performance degradation (<N) for a single fault
- N+M active, no performance degradation (<N) for up to M simultaneous faults

N+M is the generic configuration, which can be used to describe and instantiate any of the specific case configurations listed. N+M is also common target for such configurations, typically with larger N and small M to minimize the cost of meeting the associated availability/reliability targets. Common configurations and their relations to pooled VM's state replication methods are outlined below, using N+M=4 as illustrative example configuration for each specific case. Depending on the nature and protection requirements of the associated internal state, some of the protection and replication mechanism combinations may have limitations or restrictions on their applicability.

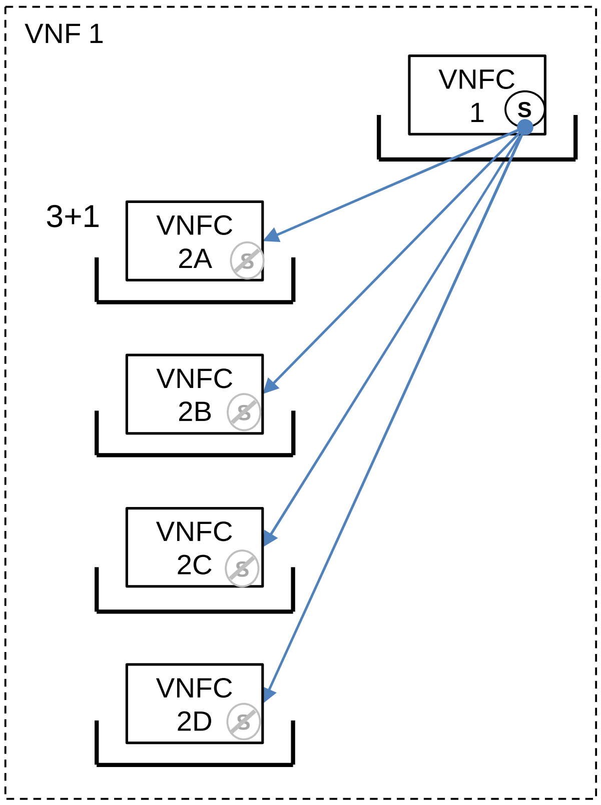


Figure 26: N+M example, 3+1, no pool member VM state replication

In Figure 26, during normal operation state is pushed from the associated controlling element (VNFC1) to the replicas of VNFC2. In case of a VNFC2 failover all associated state is retrieved from VNFC1.

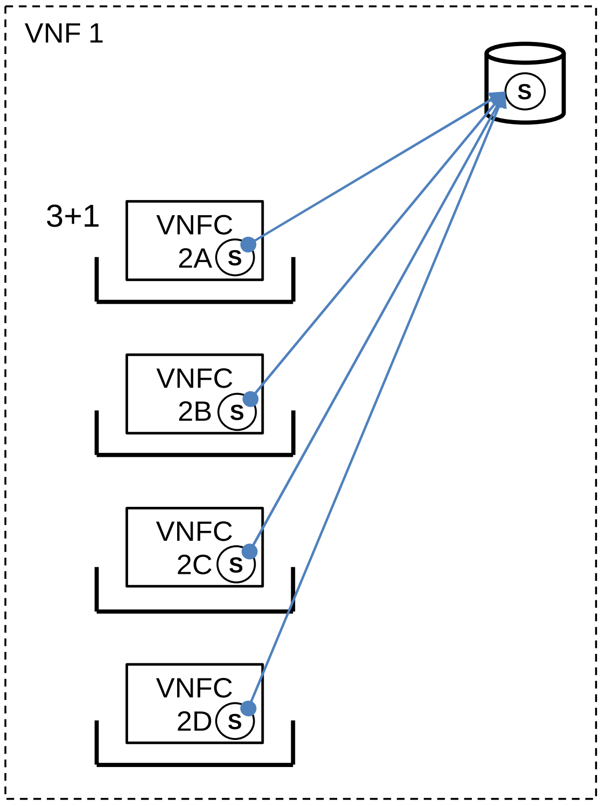


Figure 27: N+M example, 3+1, pool partial VM state replication to external state repository; stateful remediation

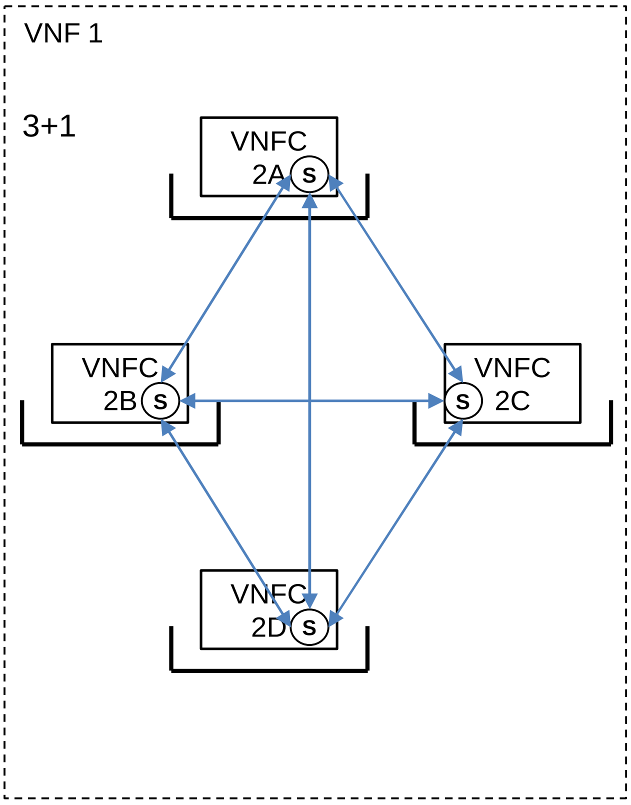


Figure 28: N+M example, 3+1, working with direct partial VM mesh state replication; stateful remediation

6.2.3 VNF protection schemes

6.2.3.1 Introduction

In ETSI GS NFV-REL 001 [i.2], mechanisms for detecting failures and for restoring services are investigated mainly from the viewpoint of the interaction between VNFs, NFVI, and NFV-MANO. ETSI GS NFV-MAN 001 [i.5] and ETSI GS NFV-SWA 001 [i.6] refer to scaling-up/down and scaling-out/in phases and a VNF upgrade phase as parts of the VNF lifecycle. However, mechanisms to enhance the VNF reliability are not studied very much in these documents.

This clause extends the investigation area to the VNF protection schemes, with which the reliability of NFs is realized. VNF protection schemes investigated include traditional Active-Standby method, Active-Active method and load balancing method with state transfer among VNFs.

6.2.3.2 Active-Standby method

The Active-Standby method is one of the popular redundancy methods adopted in many high availability systems. This configuration is depicted in Figure 29. Note that the Active-Standby method is referred to as one of the 2N redundancy model (N=1) in "Service Availability Forum Application Interface Specification (SA Forum AIS)" [i.7].

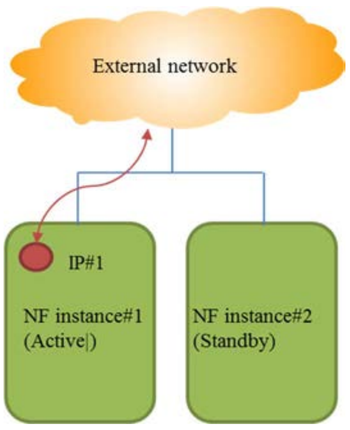


Figure 29: An Active-Standby configuration of NFs

- State protection

  Some NFs, such as stateful SIP proxies, use session state information for their operation. In this case, the active NF shares the state information with its standby NF to enable service continuity after a switch-over. The same principle applies for VNF Active-Standby protection schemes.

  There are two methods to store state information. The state information can be externalized or stored_at/shared_among peer mates. Restoration mechanisms with externalised state data have been described in ETSI GS NFV-REL 001 [i.2]. Hence, this clause focuses on the latter case where state information is stored at peer mates. Since shared state information needs to be kept consistent within the system, checkpointing or other methods will be used as described in ETSI GS NFV-REL 002 [i.8].

- Recovery and remediation phase

  When an active NF instance fails, the standby NF instance takes over the active role so that the availability of the system is maintained. The possible main steps are as follows:

  1) Failure detection of the active NF instance.

  2) Failover from the former active NF instance to the former standby NF instance. The former standby instance becomes the new active instance.

  3) Replacement of the failed NF instance with a new one which becomes the new standby NF instance.

  4) Start of the state information replication to the new standby NF instance.

Transparent failover in traditional deployments is enabled by assigning one logical IP address that the clients use to communicate with the system. In addition, each instance has its own fixed IP address used for designating each NF instance distinctively.

During normal operation, the active NF instance serves the logical IP address. If the active NF instance fails and the failure is detected, the former standby NF instance will become active, and starts serving packets destined to the logical IP address. Hence, the system can continue providing the service. This is depicted in Figure 30 and Figure 31.

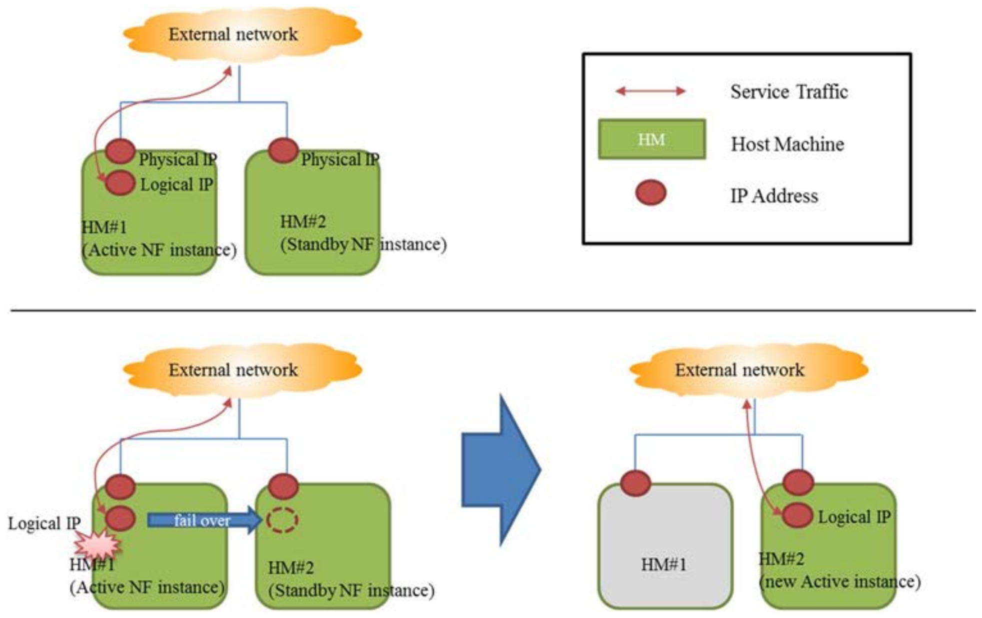


Figure 30: An example of the Active-Standby method in traditional environments

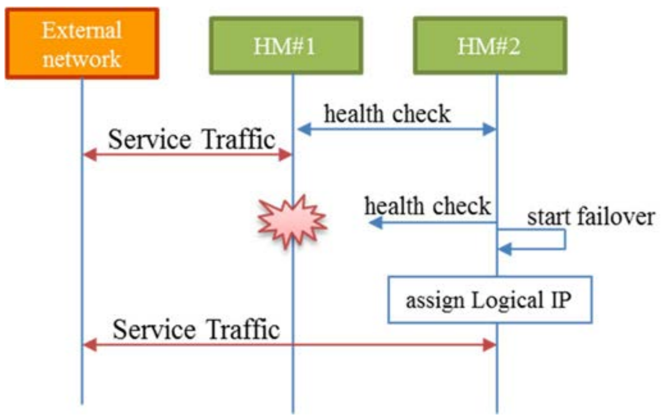


Figure 31: Failover flow for a traditional system with the Active-Standby method

In NFV deployments, the active VNF and clients on the external network are typically connected via NFVI (network infrastructure domain) as shown in Figure 32. This makes failover mechanism of virtualised systems different from that of the traditional ones. Since the interconnection between the active VNF and the next NF in the service chain is managed by NFV-MANO, NFV-MANO's availability influences the availability of the NS.

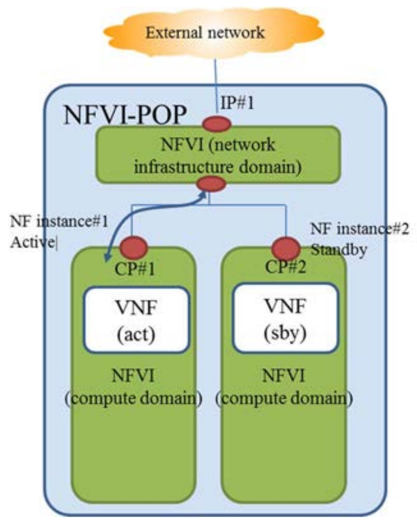


Figure 32: A typical Active-Standby configuration of VNFs in the NFV architecture

In this configuration, the assumption is that the active VNF and the standby VNF are deployed in a location-disjoint mode. For this, NFV-MANO need to support anti-affinity rules for deploying VNFs so that each VNF instance is installed on a different compute domain.

Each VNF instance has an internal connection point (CP), with an associated IP address. The two internal connection points are aggregated to a single connection point that is visible to systems on the external network. For example, a virtual router connects the IP address assigned to the external CP with the internal CP of the active VNF. Thereby, the active VNF can communicate with clients on the external network and can provide services.

Like in traditional systems with Active-Standby protection, the standby VNF periodically checks the operational status of the active VNF, e.g. using a heartbeat mechanism, which enables failure detection of the active system within the required time for the service. Though NFV-MANO may also perform health check of the VNFs, it increases dependency on NFV-MANO to propagate information concerning the failure of an active VNF to the standby VNF via NFV-MANO.

In case the active VNF (VNF#1 in Figure 33) fails, the standby VNF (VNF#2 in Figure 33) will initiate the fail-over procedure. It is realised by reassigning the external CP to the internal CP of the former standby VNF.

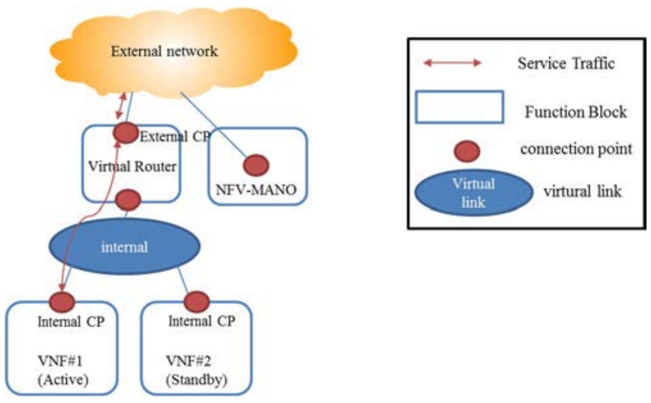


Figure 33: Active-Standby method in the virtualised environment

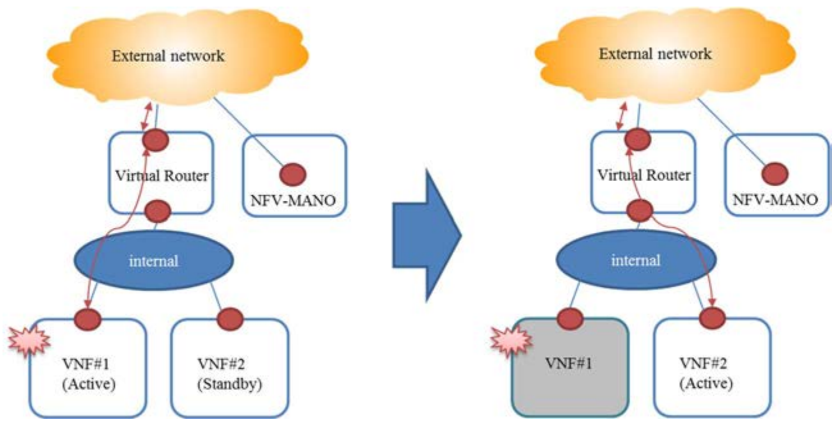


Figure 34: Active-Standby failover in the virtualised environment

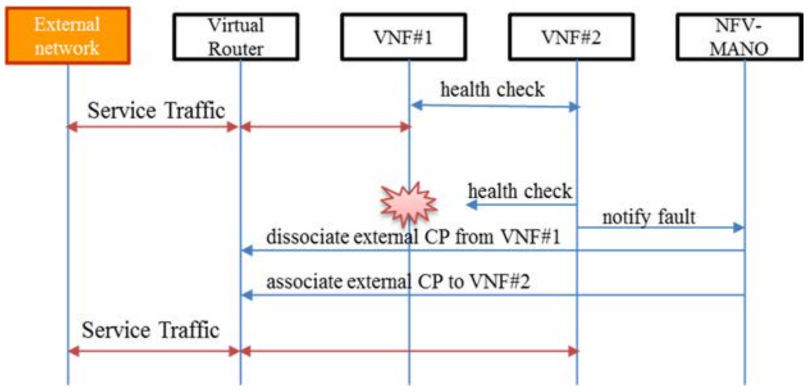


Figure 35: A failover procedure of VNFs in the Active-Standby configuration

In NFV environments, the mapping of the external CP to the internal CP is configured on the virtual router. This mapping cannot be reconfigured without NFV-MANO. Therefore, NFV-MANO should be informed about the failure of the active VNF, and associate the external CP with the internal CP of the former standby VNF instance (Figure 35). Moreover, it is also expected that NFV-MANO instantiates a new standby VNF. Note that the time required for changing the CP mapping affects the availability of the system, since the service is unavailable before reconfiguration, and that the time needed for instantiating a new standby VNF is related to the remediation period's duration.

In addition to reconfiguring the CP mapping of the system, further consideration to keep the system's state information consistent is required. Figure 36 shows VNFs configured with the Active-Standby method with state replication between VNFs. The active VNF stores session state information in a storage within itself, and replicates the information to the storage within the standby VNF. Figure 37 shows the restoration procedure for this setup.

Since the new active VNF has to check the consistency of the backup state information to judge whether the service can be continued before receiving the subsequent service traffic flow, the route change request sent to NFV-MANO should be triggered by the new active (i.e. the former standby) VNF.

For the remainder of this clause, only Active-Standby configurations with state synchronization are described since they are a superset of those configurations without state synchronization.

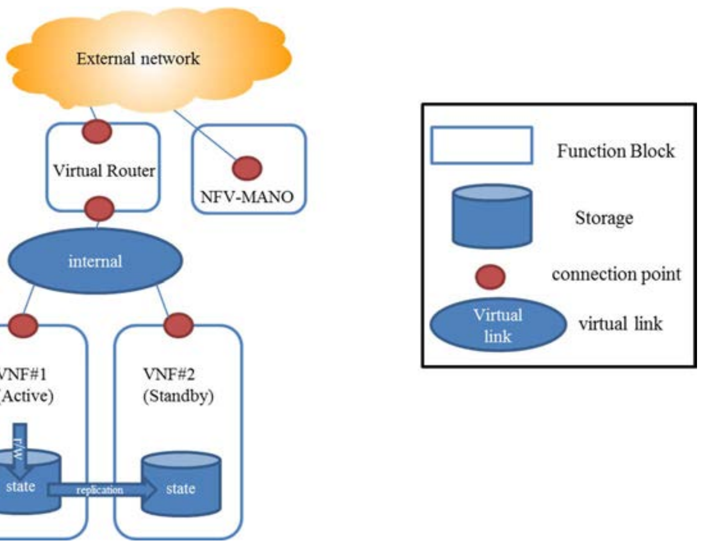


Figure 36: Stateful VNFs with internal storage for states in the Active-Standby configuration

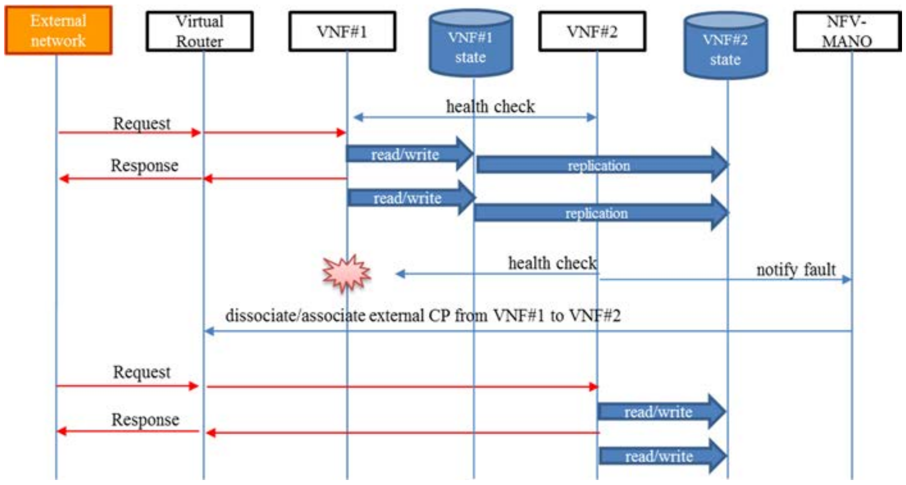


Figure 37: A failover procedure of VNFs synchronizing state information in the Active-Standby configuration

- Scaling phase

  Dynamic scale out and scale in requires the configuration of a NF group. This group can contain multiple sets of Active-Standby pairs. Clients have either to select a designated NF by its logical IP address or a load balancer is inserted in front of the NFs to provide a single IP address for the NF group to the clients.

  The configuration with a load balancer is described in clause 6.2.3.3. Alternatively, a service can be scaled-up or scaled-down by modifying existing VMs with more or less resources. Note that the difference between these methods in terms of reliability and availability will be discussed in clause 7.2. A procedure for scaling-up/down is as follows:

  1) Prepare (create/instantiate) a new standby NF instance with required infrastructure resource (memory, CPU, etc.) and activate it.

  2) Stop replication of state information from the active NF instance to the current standby NF instance.

  3) Start replication from the active NF instance to the new standby NF instance, which is prepared in step1.

  4) After the replication is completed and both are synchronized, switchover the Active-Standby NF instances.

  5) Terminate/remove the former standby NF resources.

6) As an alternate NF instance for the former active NF, prepare (create/instantiate) a new NF instance with required infrastructure resource (memory, CPU, etc.) and activate it.

7) Stop replication of state information from the new active NF instance to the standby NF instance.

8) Start replication of state information from the active NF instance to the new standby NF instance prepared in step 6.

9) After the replication is completed and both are synchronized, terminate/remove the former active NF resources.

Figure 38 shows a possible flow diagram of scaling-up/down operation of stateful VNFs Note that a heavy line in this figure means that the corresponding VNF instance is active during this period, and a dotted line means that the VNF instance is inactive.

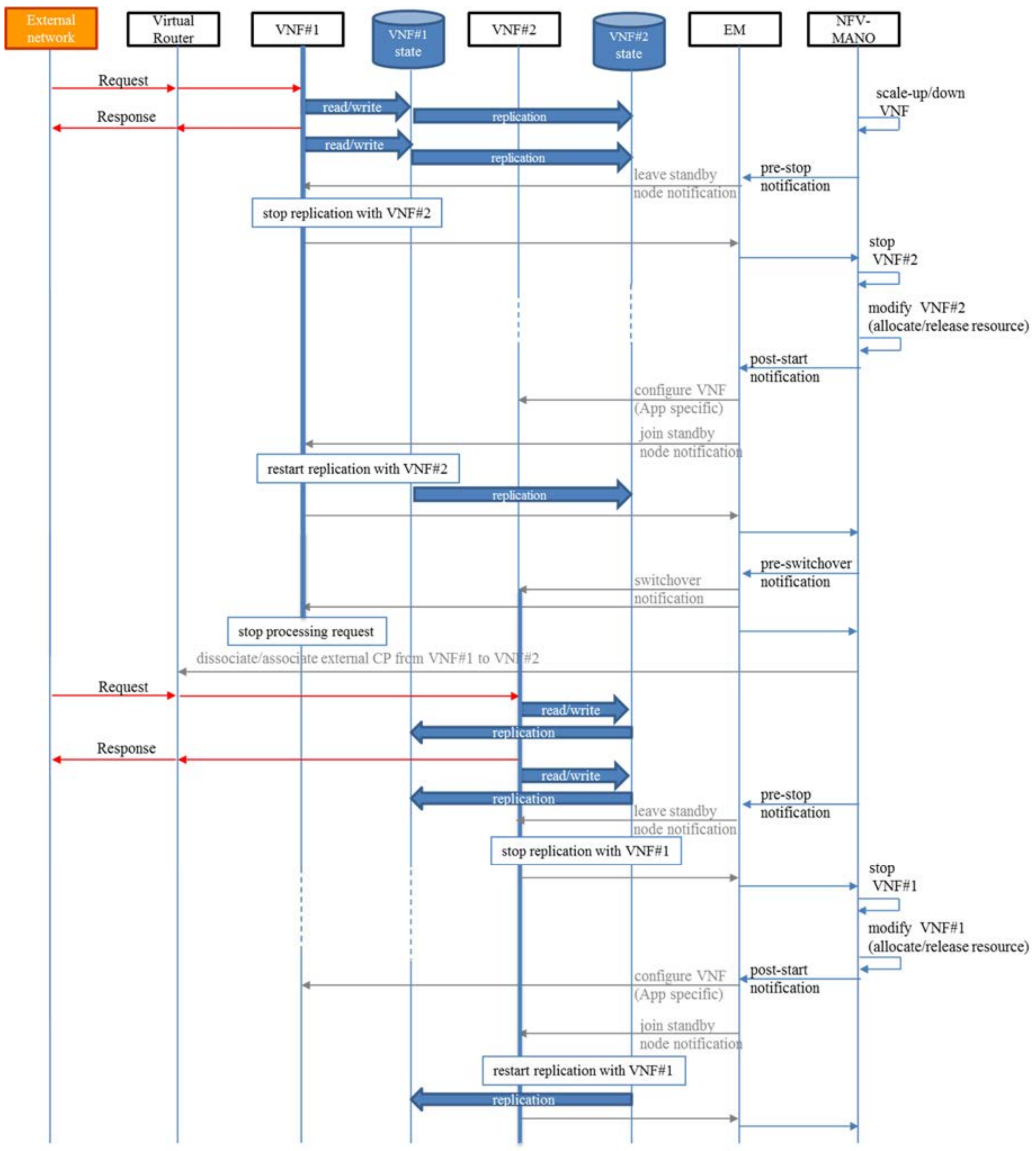

External network
Virtual Router
VNF#1
VNF#1 state
VNF#2
VNF#2 state
EM
NFV-MANO
Request
Response
read/write
replication
scale-up/down VNF
leave standby node notification
pre-stop notification
stop replication with VNF#2
stop VNF#2
modify VNF#2 (allocate/release resource)
post-start notification
configure VNF (App specific)
join standby node notification
restart replication with VNF#2
pre-switchover notification
switchover notification
stop processing request
dissociate/associate external CP from VNF#1 to VNF#2
stop replication with VNF#1
stop VNF#1
modify VNF#1 (allocate/release resource)
restart replication with VNF#1

Figure 38: A scale up/down procedure of VNFs synchronizing state information in the Active-Standby configuration

When NFV-MANO terminates or deactivates a standby VNF instance, some preparation for termination or deactivation should be done. For example, the standby VNF instance should be de-correlated from the active one, so that the active VNF does not interpret, e.g. the lack of keep-alive messages, as a failure of the standby VNF instance. Since this preparation is likely to be application specific, NFV-MANO should inform the corresponding active VNF and/or EM that the standby VNF instance is going to be terminated or inactivated (pre-stop notification in Figure 38).

Similarly, on switchover, an active VNF instance stops processing new requests from external NFs and completes undergoing processing before it hands over the active role to the standby VNF instance. The standby instance begins processing events (becomes active) only after the former active VNF instance becomes ready for termination or deactivation. Therefore, NFV-MANO should notify VNFs and/or EM that the switchover is requested (pre-switchover notification in Figure 38).

- Software upgrade phase

  The software upgrade procedure for systems configured with the Active-Standby method is similar to the scale-up/down procedure.

  A possible software upgrade procedure is as follows:

  1) Stop replication of state information from an active NF to the corresponding standby NF, and deactivate or terminate the standby NF.

  2) Upgrade software of the standby NF or prepare/instantiate a new standby NF with the new software.

  3) Activate the standby NF and start replication from the active NF to the standby NF with necessary conversion of state information so as the new software to handle it correctly.

  4) Switchover the Active-Standby NF.

  5) Stop replication of state information from the active (former standby) NF to the standby (former active) NF, and deactivate or terminate the standby NF.

  6) Upgrade software of the standby (former active) NF or prepare/instantiate a new standby NF with the new software.

  7) Activate the standby NF and start replication from the active NF to the standby NF.

Figure 39 shows a possible flow diagram of software upgrade of stateful VNFs. It is assumed that a new image of the VNF is stored in the repository before upgrading the VNF to the new software. It is also assumed that the VM is terminated instead of deactivated in this example. Again, note that the heavy line in this figure means that the VNF is in an active state.

Note that this procedure rather follows a traditional software upgrade of PNF, but detailed discussion of VNF software upgrade in NFV is described in clause 8.

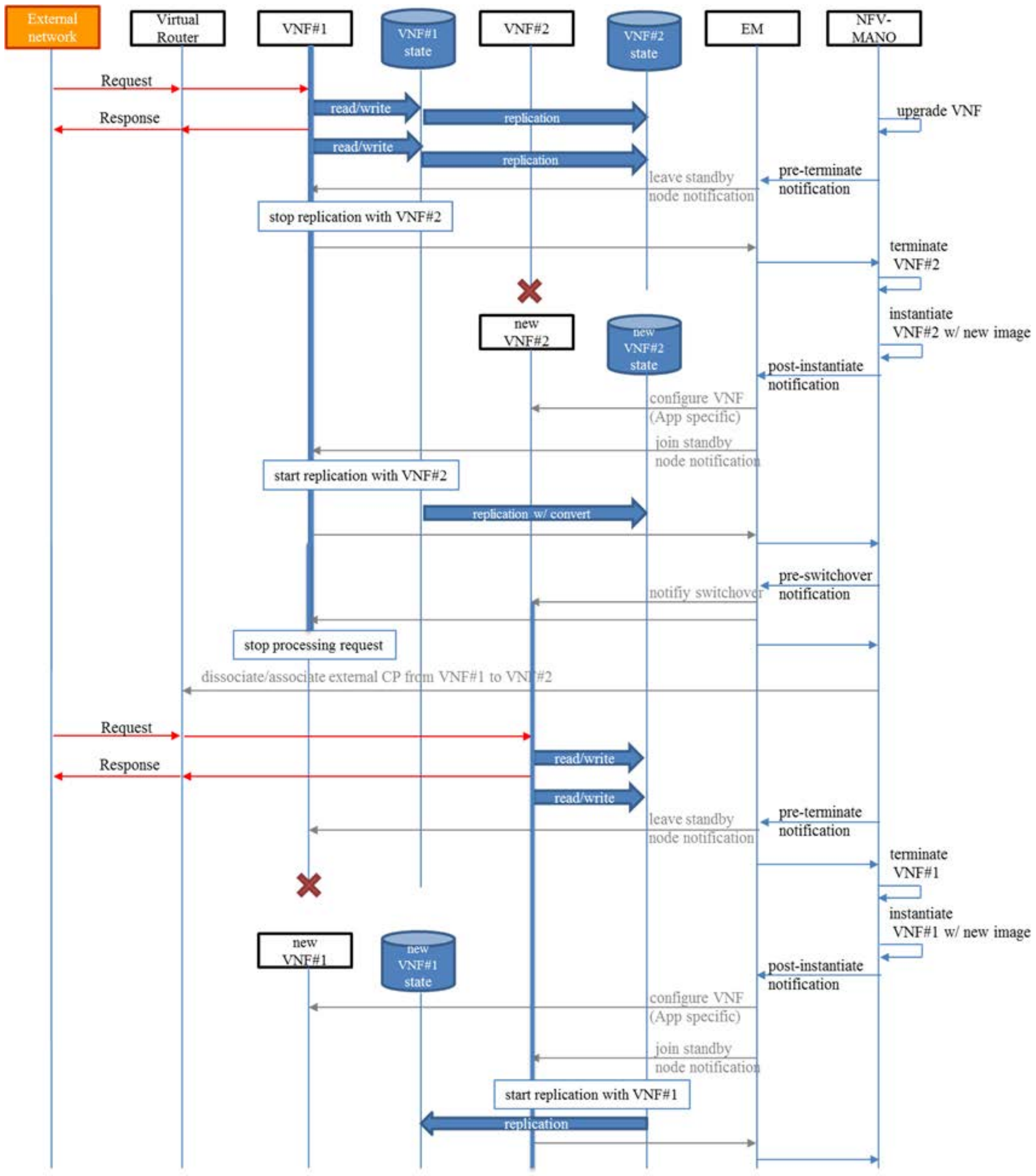

External network
Virtual Router
VNF#1
VNF#1 state
VNF#2
VNF#2 state
EM
NFV-MANO
Request
Response
read/write
replication
upgrade VNF
pre-terminate notification
leave standby node notification
stop replication with VNF#2
terminate VNF#2
instantiate VNF#2 w/ new image
new VNF#2
new VNF#2 state
post-instantiate notification
configure VNF (App specific)
join standby node notification
start replication with VNF#2
replication w/ convert
pre-switchover notification
notifiy switchover
stop processing request
dissociate/associate external CP from VNF#1 to VNF#2
terminate VNF#1
instantiate VNF#1 w/ new image
new VNF#1
new VNF#1 state
start replication with VNF#1

Figure 39: A possible procedure of the software upgrade for the Active-Standby model with state information synchronization

Similarly to the scaling-up/down procedure, NFV-MANO should inform the corresponding active VNF instance and/or the EM that the standby VNF instance is going to be terminated or inactivated (pre-terminate notification in Figure 39.)

On switchover, NFV-MANO should also notify VNFs and/or the EM that the switchover is requested (pre-switchover notification in Figure 39).

6.2.3.3 Active-Active method

The Active-Active method is a configuration where two NF instances are simultaneously active for the same service (Figure 40). The external network can receive the service by accessing either IP addresses of these NF instances. This configuration is referred to as an Active-Active redundancy configuration (N-way active redundancy model N=2) in the SA Forum AIS document [i.7].

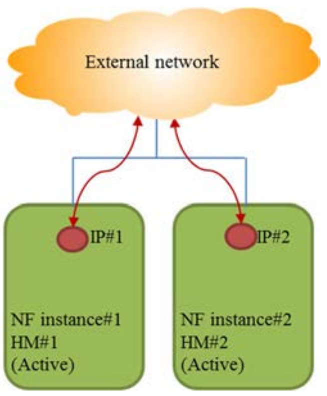


Figure 40: An Active-Active configuration of NFs

- State protection

  This configuration does not contain a standby system. Therefore, externalisation should be used to maintain the state information for service continuity in case of system failure. The same principle applies for VNF Active-Active protection schemes, and external storages provided by NFVI may be used for the state protection. State protection using NFVI storage is described in clause 6.2.2.4.

- Recovery and remediation phase

  Figure 41 depicts a configuration change when NF instance#1 fails in a traditional system. External NFs can receive services from NF instance #2 when no answer is received from the NF instance #1, by changing the destination of requests for the NF from IP#1 to IP#2, which is usually done by the DNS in which the NFs are assigned to a primary NF and to a secondary NF. Note that when a load balancer is placed in front of the NF instances, external NFs can get services without knowing the failure of NF instance#1, since the load balancer can deliver all the requests to the NF instance #2 when the NF instance #1 fails.

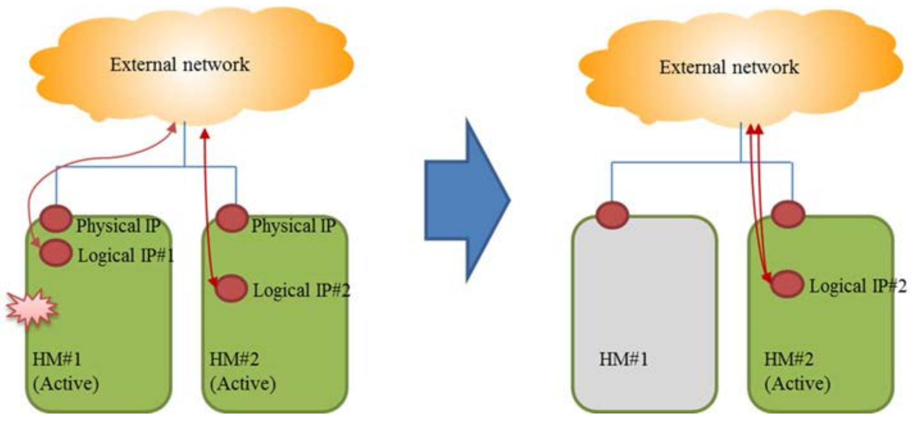


Figure 41: An example of remediation mechanism with Active-Active method in traditional physical environments

In NFV deployment, the active VNF and clients on the external network are connected via NFVI (network infrastructure domain) instead of using a logical IP address assigned to a virtual machine where the VNF is deployed.

In this case, two VNFs are deployed on different compute domains according to anti-affinity rules. Two external CPs are assigned to the virtual router in a network infrastructure domain. Each external CP is connected respectively to an internal CP of each VNF.

Figure 42 shows an example of Active-Active configuration in NFV environments. In this figure, NFV-MANO does not have to reconfigure the mapping of external CP and internal CP, since the clients in the external network change the destination of the request message so as to be able to get services from the VNF instance #2.

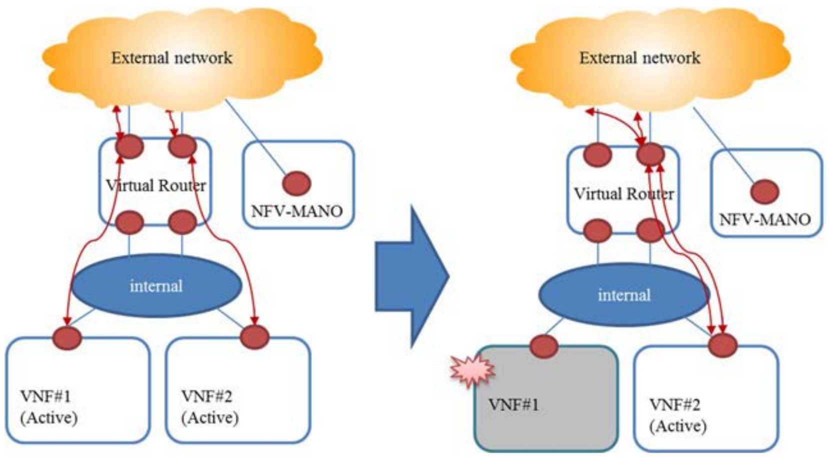


Figure 42: An example of Active-Active method in the NFV environment

- Scaling phase

  Similar to the Active-Standby method, dynamic scale out and scale in require the configuration of a NF group. Clients on the external network have either to select a designated NF that is added by its logical IP address with the help of address resolution service like DNS, or a load balancer has to be inserted in the network infrastructure domain in front of the NFs by NFV-MANO when scaling out happens. The load balancer delivers traffic from the external CP to each set of VNFs so that the traffic is fairly distributed. The configuration with a load balancer is described and investigated in clause 6.2.3.4. Alternatively, a service can be scaled-up and scaled-down by replacing the existing VMs with those with more or less resources. Note that the difference between these methods in terms of reliability and availability is discussed in clause 7.2.

  Figure 43 shows a possible flow of scaling-up/down of VNFs lead by NFV-MANO. In this figure, active service instances provided by VNF #1 are depicted with a blue line, and active service instances provided by VNF #2 are depicted with a green line.

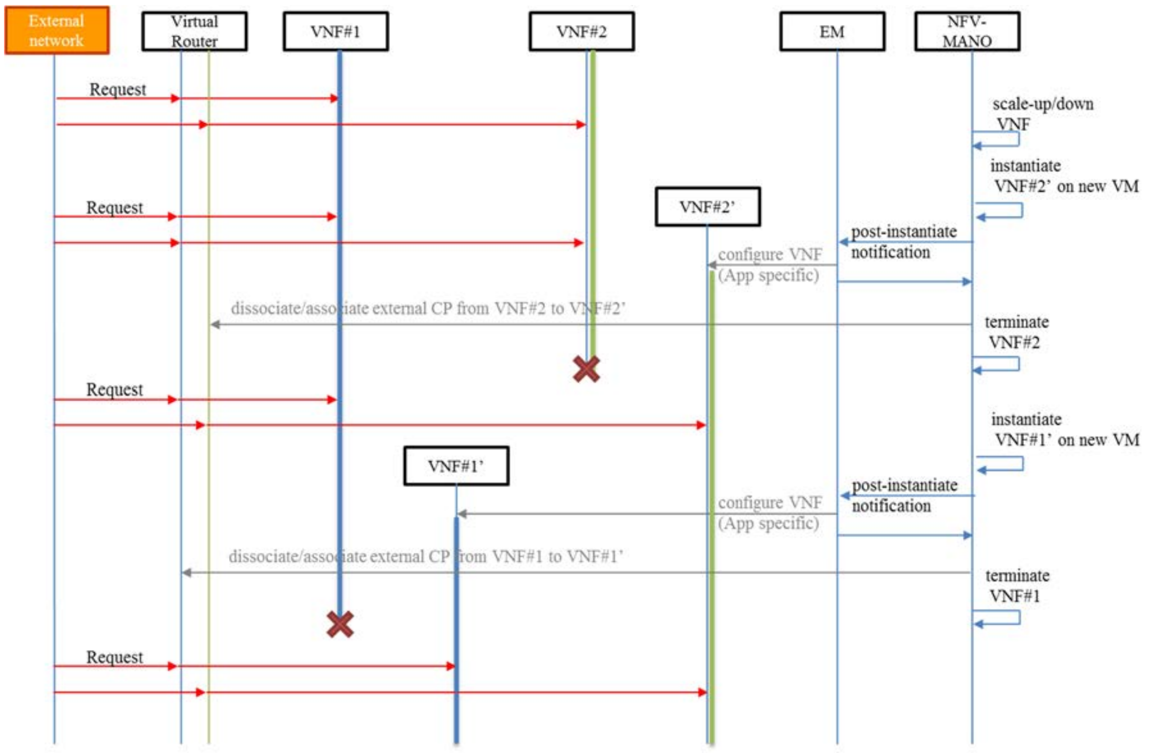

External network
Virtual Router
VNF#1
VNF#2
EM
NFV-MANO
Request
scale-up/down VNF
instantiate VNF#2' on new VM
VNF#2'
Request
post-instantiate notification
configure VNF (App specific)
dissociate/associate external CP from VNF#2 to VNF#2'
terminate VNF#2
Request
instantiate VNF#1' on new VM
VNF#1'
post-instantiate notification
configure VNF (App specific)
dissociate/associate external CP from VNF#1 to VNF#1'
terminate VNF#1
Request

Figure 43: A scale up/down procedure of VNFs synchronizing state information in the Active-Active configuration

- Software upgrade phase

  Figure 44 shows a possible flow of software upgrade procedure for VNFs performed by NFV-MANO. This figure assumes that a new VNF image is stored in a repository before software upgrade. It is also assumed that the VM is terminated instead of deactivated in this example.

  When the VNF is stateful, the old VNF has to exist until all the ongoing services (transactions) are terminated for the service continuity. Detailed discussion on software upgrade with service continuity will be discussed in clause 8.

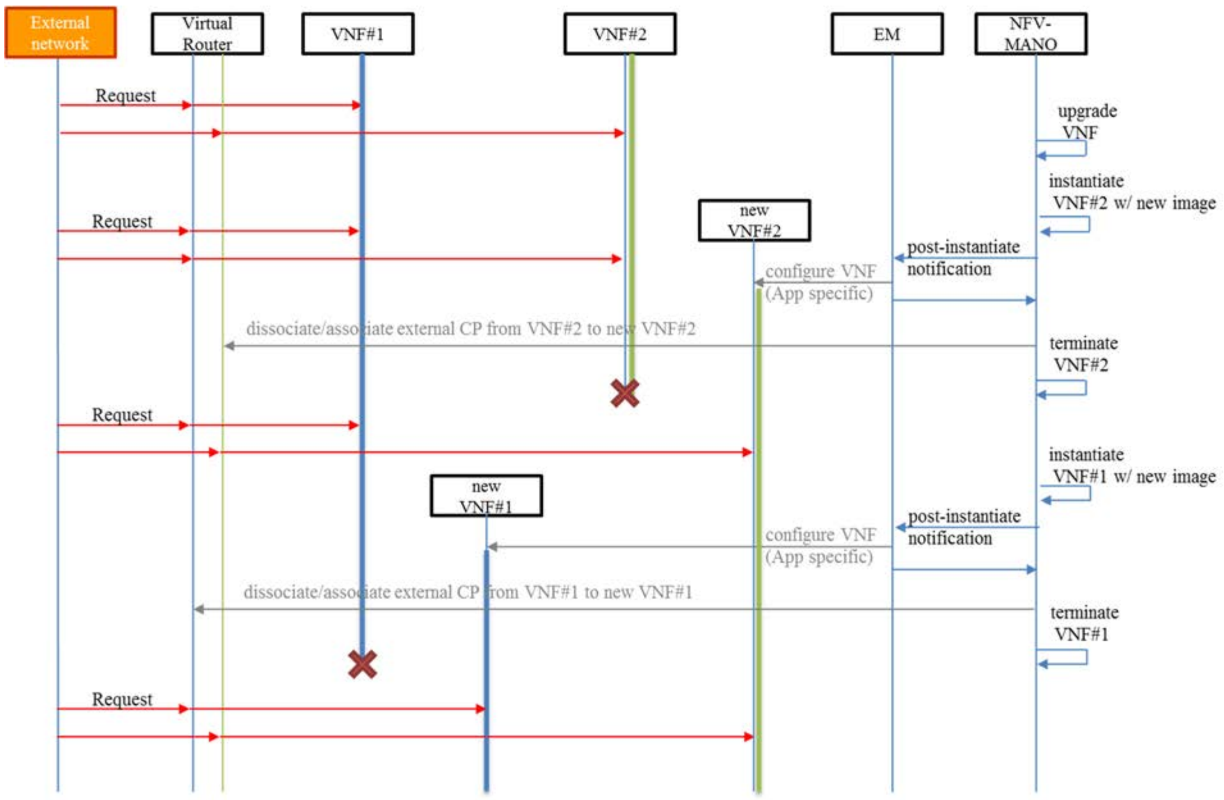

External network
Virtual Router
VNF#1
VNF#2
EM
NFV-MANO
Request
upgrade VNF
instantiate VNF#2 w/ new image
new VNF#2
Request
post-instantiate notification
configure VNF (App specific)
dissociate/associate external CP from VNF#2 to new VNF#2
terminate VNF#2
Request
instantiate VNF#1 w/ new image
new VNF#1
post-instantiate notification
configure VNF (App specific)
dissociate/associate external CP from VNF#1 to new VNF#1
terminate VNF#1
Request

Figure 44: A possible procedure of software upgrade for the Active-Active method with state information synchronization in the NFV environment

6.2.3.4 Load balancing method

Load balancing is a popular method used for Web servers. In cloud environments, a load balancer is deployed in front of several VNFs. The service traffic goes through the load balancer and is distributed to several VNFs that handle the whole traffic (Figure 45). A load balancing method that corresponds to the N-way redundancy model of the SA Forum AIS document [i.7] is investigated in this clause.

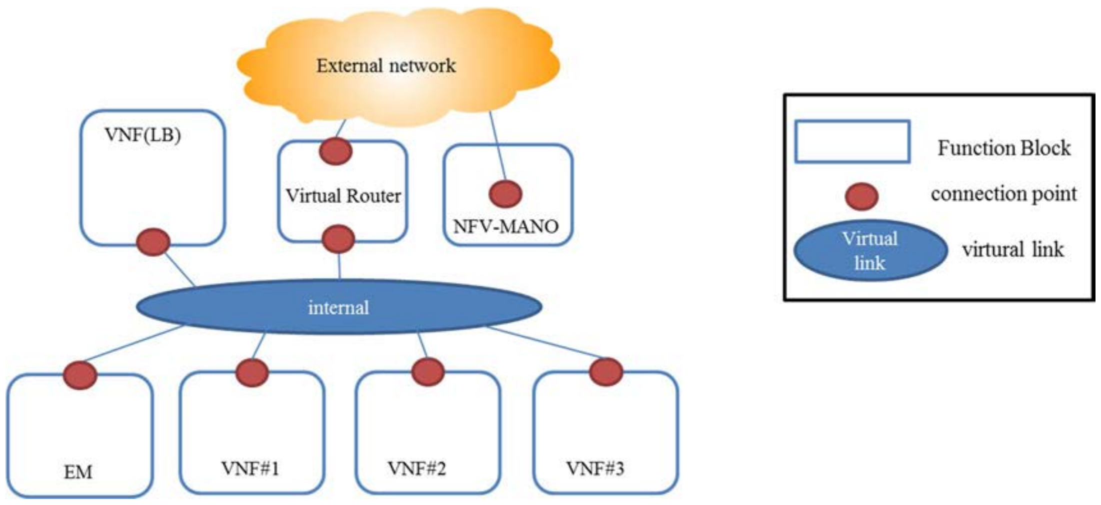


Figure 45: A load balancing model in virtualised environments

- State protection

  When a VNF is stateful, e.g. SIP proxies, there are two methods to store state information. One method is to externalize the state information. The other is to share the state information among peer mates, which means that the state information of the service instances within VNFs have their standby information in other VNFs as shown in Figure 46. In this figure, the state information of the service instances running on VNF#1 is replicated in VNF#2 as a backup. Similarly, the one of service instances on VNF#2 (resp. #3) is replicated on VNF#3 (resp. #1).

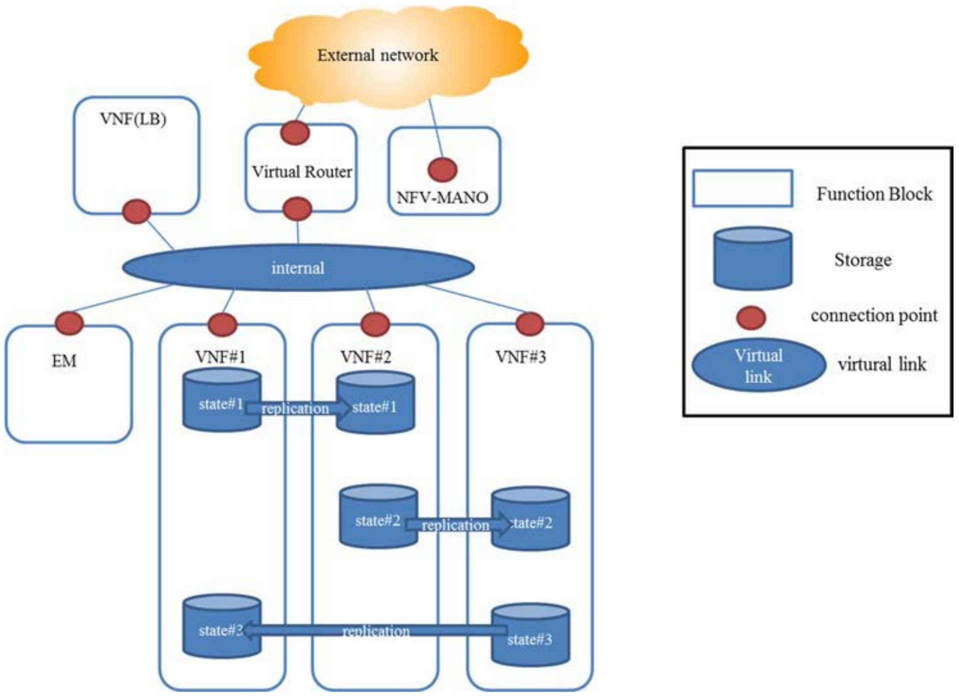


Figure 46: Load balancing model with internal storage

Restoration mechanisms with externalized state data have been described in ETSI GS NFV-REL 001 [i.2]. Hence, this clause mainly focuses on the latter case where state information is stored at peer mates. Since shared state information is to be kept consistent within the system, checkpointing or other methods will be used as described in ETSI GS NFV-REL 002 [i.8].

- Remediation and recovery phases

  As an assumption, a load balancer periodically checks the operational status of VNFs to which service traffic is delivered to avoid delivering traffic to the failed VNF. When the load balancer detects the failure of a VNF, it stops providing traffic to the failed VNF and delivers it to other VNFs so that service continuity is maintained (Figure 47). When VNFs are stateful, the traffic to the failed VNF has to be delivered to the VNF that stores the state information of the failed service instances as a backup. The standby service instances of the VNF continue the service with the backup information. In some cases, the standby service instances recognize the failure of the corresponding active service instances so as to take over the ongoing services. A possible way to do this is that VNFs check one another, so that the standby service instances within the VNF are aware of the failure of their corresponding active service instances.

  Since the external CP that is visible to clients on the external network is only assigned to the virtual load balancer, NFV-MANO does not have to explicitly change the configuration between the load balancer and VNFs for remediation. For recovery, NFV-MANO should deploy a new VNF that compensates the performance and the availability lost due to the failure.

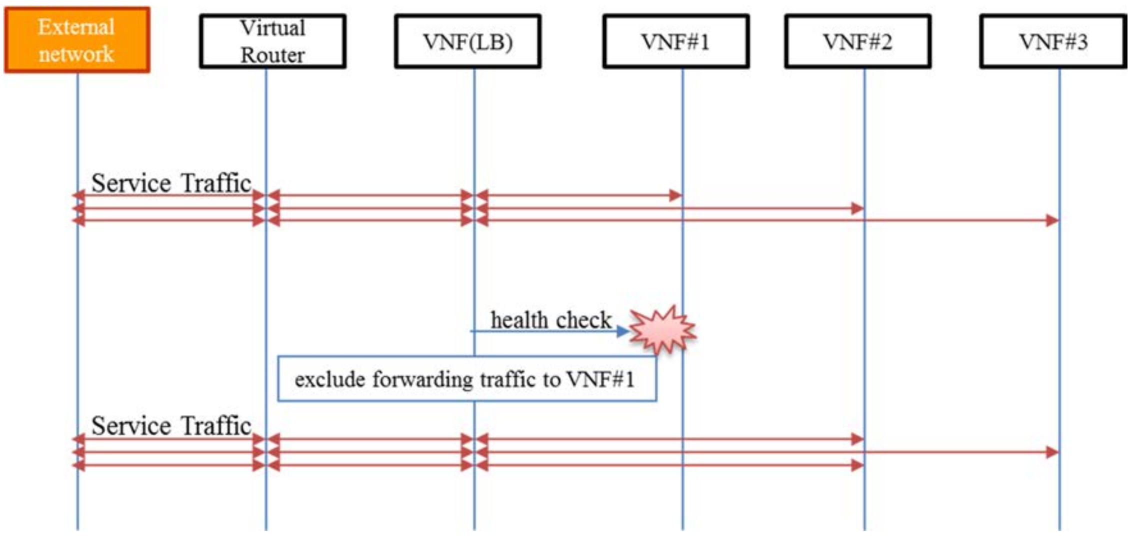


Figure 47: A sample procedure that will occur when a VNF with the service application fails

Note that a virtual load balancer should also be redundant, e.g. Active-Standby, to avoid single point of failure. For Active-Standby virtual load balancers, the same requirements for NFV-MANO as those derived from the investigation of the Active-Standby method are imposed as shown in clause 6.2.3.2.

- Scaling phase

  With a load balancing model, auto scaling-in/out will be performed as described in Figure 48 and Figure 49. The performance of a group of VNFs can increase/decrease according to the traffic load increase/decrease. Three deployments options are considered: first scaling of a stateless service with a load balancer, second scaling of a stateful service with a load balancer, and third scaling of a stateful service with a load balancer and externalized state.

  An example of an auto scaling-out procedure for a stateless service is as follows (Figure 48). NFV-MANO monitors the condition of the VNF instances and when it meets the predefined condition for auto-scaling-out, e.g. when the load of each VNF instance exceeds a pre-defined threshold, NFV-MANO prepares a new VNF instance and requests the EM to set it up for starting services, or requests the VNF instance to prepare to provide services. After the VNF is ready for service, the EM or the VNF itself requests the load balancer to deliver service traffic to it.

  An auto scaling-in procedure for a stateless service is as follows. NFV-MANO monitors the condition of the VNF instances and when it meets the predefined condition for auto-scaling-in, e.g. when the load of each VNF instance falls below a pre-defined threshold, NFV-MANO requests the EM to stop a VNF instance, or requests a VNF instance to stop itself. The traffic to the terminating VNF instance (VNF#3 in Figure 49) should stop before NFV-MANO terminates VNF#3, and NFV-MANO should also tell the EM/VNF to prepare to terminateVNF#3 for the graceful termination.

Since the sequence for terminating a VNF instance is an application specific procedure as shown above, the EM will be able to take care of or conduct the procedure. The EM requests the load balancer (LB) to stop delivering service traffic to VNF#3 (Figure 49). When all events in VNF#3 are processed and all the traffic to VNF#3 is stopped, the EM lets NFV-MANO know that VNF#3 is ready to be terminated.

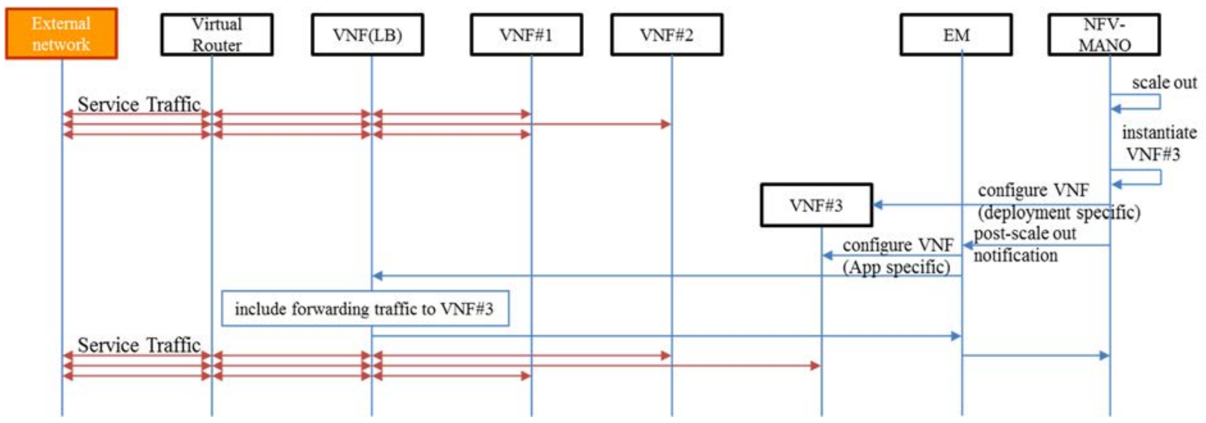

External network
Virtual Router
VNF(LB)
VNF#1
VNF#2
EM
NFV-MANO
scale out
Service Traffic
instantiate VNF#3
VNF#3
configure VNF (deployment specific)
post-scale out notification
configure VNF (App specific)
include forwarding traffic to VNF#3
Service Traffic

Figure 48: A possible scale-out procedure of a VNF in the load balancing model (stateless)

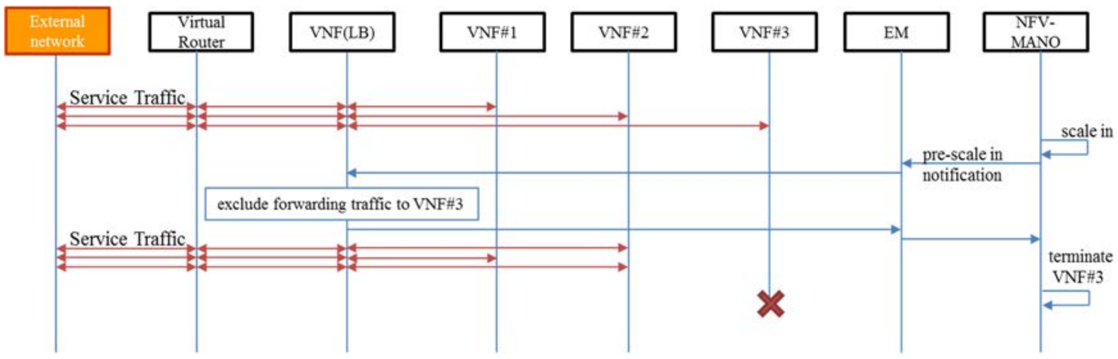


Figure 49: A possible scale-in procedure of a VNF in the load balancing model (stateless)

For stateful services, a load balancing deployment where state information resides in the VNF instances is shown in Figure 46. The information is transferred to the VNF instance that acts as a backup of that session. In this figure, when a VNF instance (VNF#3) is to be terminated, the sessions processed in VNF#3 will be taken over by VNF#1, which has state information of the session as a backup. The replicated data of the sessions processed in VNF#2, which are stored in the terminating VNF instance (VNF#3), are transferred to a new backup VNF instance, such as VNF#1, by VNF#2. This configuration change should be managed by the EM.

Figure 50 depicts the flow diagram for this procedure.

When VNF#3 becomes ready to be terminated, the EM lets NFV-MANO know that VNF#3 and the corresponding VM are ready to be deleted.

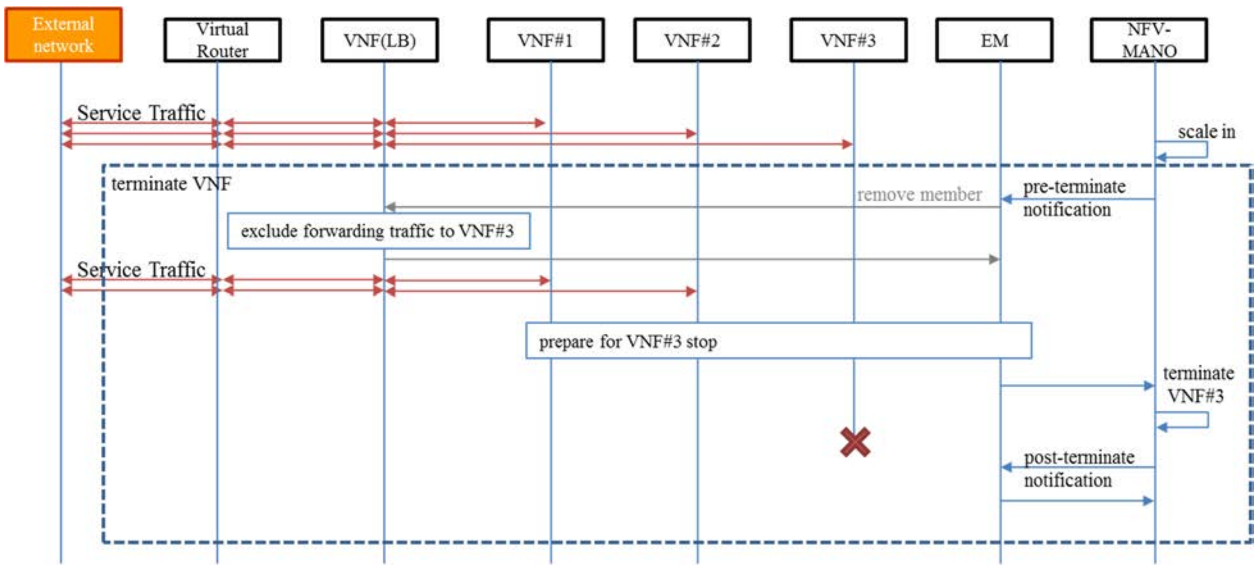


Figure 50: A possible scale-in procedure of a VNF in the load balancing model (stateful)

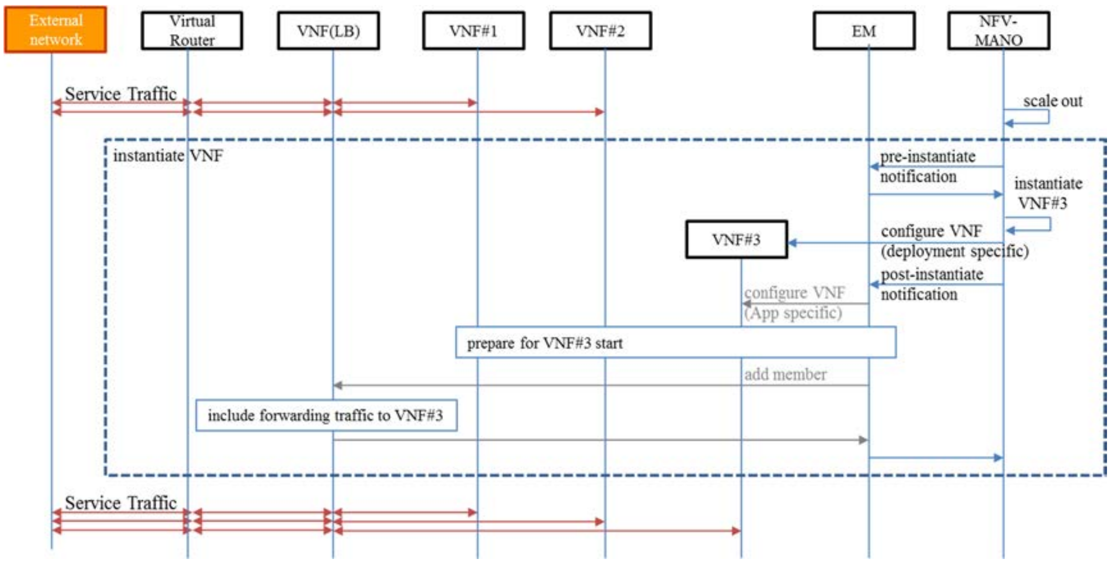


Figure 51: A possible scale-out procedure of a VNF in the load balancing model (stateful)

Scaling out procedure is similar to that of scaling in. The configuration of VNF instances, such as the backup relationship, should be reorganized. On scaling out, the required configuration of a new VNF instance may be determined based on the condition where the scaling-out requirement is met. Therefore, the EM may decide the configuration of the new VNF instance and prepare the information that will be used in NFV-MANO, as well as reorganize the configuration between VNF instances, since this procedure is thought to be VNF specific.

As described in ETSI GS NFV-REL 001 [i.2], the session state can be stored in an external storage. Figure 52 shows an example of a load balancing model with an external virtual storage. When a VNF instance (VNF#1) is to be terminated, another VNF instance (VNF#2) takes over the sessions processed by VNF#1 based on the information stored in the virtual storage. Auto scaling-in and scaling-out procedures are similar to those depicted in Figure 50 and Figure 51 despite that the preparation procedures for the termination and the initiation of a VNF instance (VNF#3) differ from those of the load balancing model with internal storage.

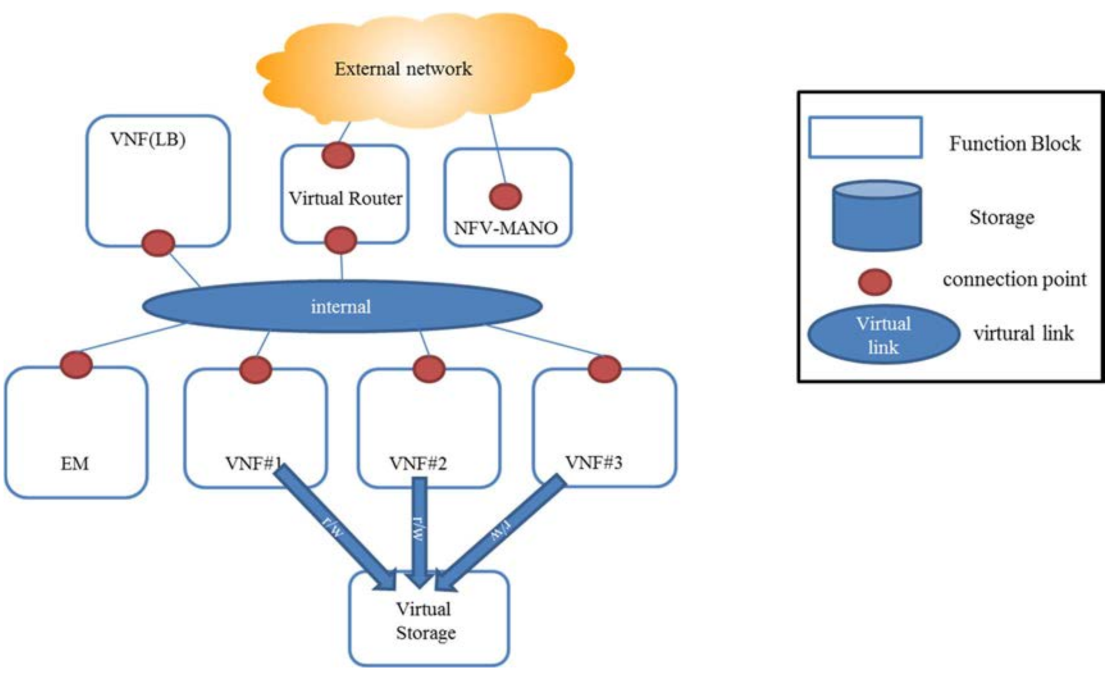


Figure 52: Load balancing model with virtual storage

When this model is applied, NFVI should make the virtual storage reliable and avoid it being a single point of failure.

- Software upgrade phase

  This clause describes software upgrade/update where VNFs are gradually replaced by instances with newer version software. VNF instances with the older software version are removed one by one. This mechanism is called the rolling update method. Figure 53 shows a possible flow of rolling update controlled by the EM.

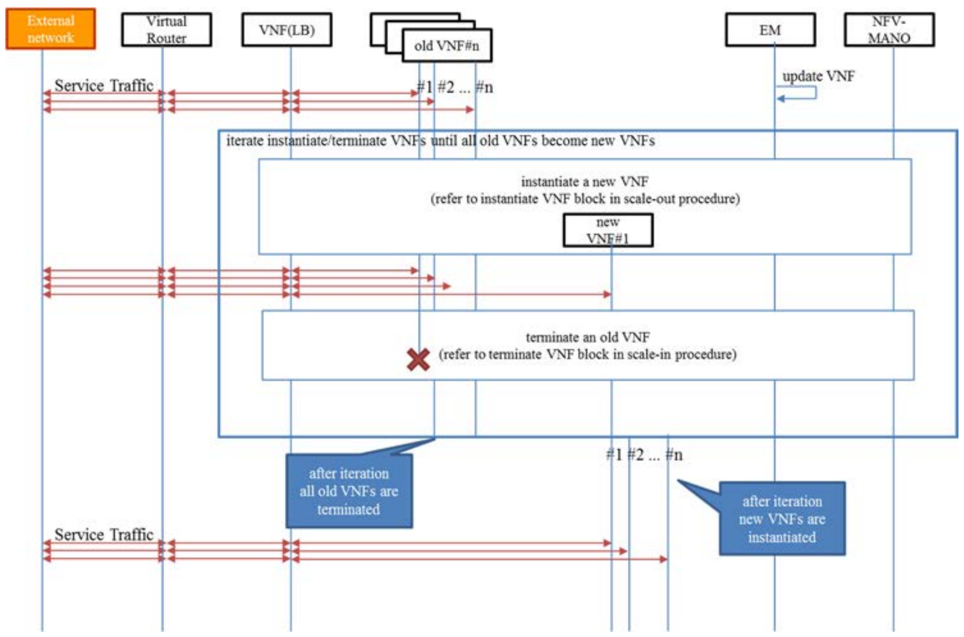

External network
Virtual Router
VNF(LB)
old VNF#n
EM
NFV-MANO
Service Traffic
#1 #2 ... #n
update VNF
iterate instantiate/terminate VNFs until all old VNFs become new VNFs
instantiate a new VNF
(refer to instantiate VNF block in scale-out procedure)
new VNF#1
terminate an old VNF
(refer to terminate VNF block in scale-in procedure)
#1 #2 ... #n
after iteration all old VNFs are terminated
after iteration new VNFs are instantiated
Service Traffic

Figure 53: A possible procedure of the rolling update

When no state information is shared among VNF instances, rolling update can be done by simply creating VNF instances with the new version software and removing VNF instances with the old version software. Some preparation is needed before terminating these VNF instances. In contrast, if VNF instances share state information, e.g. when VNF instances mutually store backup information, VNF instances with the new software version are required to take over the state information from those with the old software. Since the structure of the state information can be in a different form according to the software version, new VNF instances should have the capability to transform, or to convert, the format of the state information transferred to/from the old VNF instances.

Figure 54 shows a possible procedure controlled by the EM, where traffic is distributed among VNF instances.

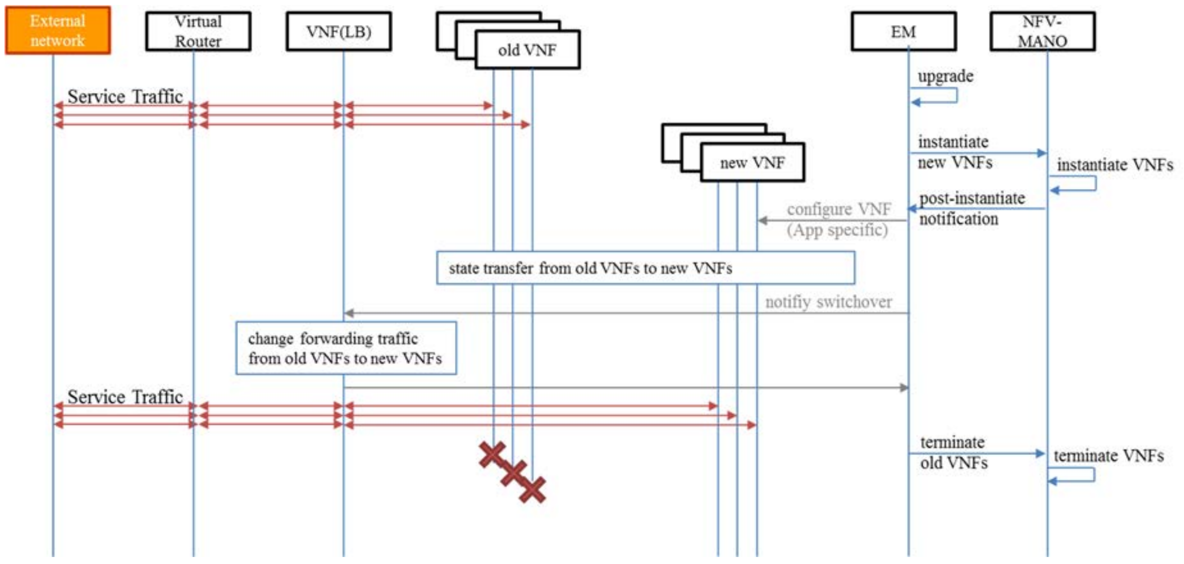


Figure 54: A possible procedure of the software upgrade for load balancing model with state information synchronization in the NFV environment

6.3 Reliability methods for virtual links

6.3.1 Introduction

There are several interconnection possibilities between VNFs, see clause 8.2 of ETSI GS NFV-INF 001 [i.11]. As the virtual network is an overlay on top of physical network(s), the reliability of virtual networks/links is dependent on the availability and functionality of the underlay or infrastructure network resources. This dependency can in general be reduced by avoiding spreading VNFC instances connected by a virtual network over different physical nodes. A placement of such connected instances on the same physical hosts saves physical network capacity and increases performance by having a higher share of connections just using virtual switches instead of physical ones. However, such a policy could violate affinity rules or other placement restrictions, and also reduces the flexibility to migrate instances. It would be the task of NFV-MANO to find a trade-off between the conflicting objectives.

This clause investigates failures occurring in physical links and their impact across NFVI nodes. Examples of link failures within NFVI nodes are described in ETSI GS NFV-REL 001 [i.2].

6.3.2 Failure of physical links across NFVI nodes

Virtual networks have to use physical networks if they are to span different physical hosts. Thus, virtual links may span one or several physical links and interconnecting network elements, any of which may fail and thus interrupt the virtual connection. In order to increase the availability of an underlay connection, redundant links can be provided, either using load sharing between the links or with dedicated backup links (analogous to the Active-Active and Active-Standby patterns for VNFCs).

Load sharing here means that the flows that are to be transported between the endpoints connected by the redundant connections are distributed among them, using, e.g. round robin or hash-based mechanisms. As a consequence, in the case of a failure of one physical link, only a share of the traffic from a VNFC to another might be affected. In contrast, if dedicated backup links are used, the complete traffic is routed over the active link(s) if no failure occurs, and is redirected via the backup link in case of failure.

These mechanisms can be implemented on the data link layer or the network layer.

- For a single underlay hop, layer 2 link protection based on NIC bonding may be used. For example, a number of Ethernet connections between two switches can be aggregated using the Link Aggregation Control Protocol (LACP) (IEEE 802.1ax) [i.12], resulting in a single logical connection. With Linux bonding drivers, different load balancing and fault tolerance modes can be used. The individual physical links do not necessarily need to have the same physical remote endpoint (i.e. switch); solutions exist for aggregating physical switches as well into a logical switch (IEEE 802.1aq) [i.13].
- On layer 3, different network paths might be used for resilience. These alternative paths can be used to share the network load, e.g. by distributing traffic flows across them, or they can be configured as active and backup protection paths. If an active path fails, its protection path can be activated and the traffic routed around the failed part of the network, such as known from MPLS fast reroute. Here, the use of SDN enables the setup of fast failover between paths, i.e. the pre-establishment of the different routes and the rerouting of the flows originally being forwarded via the failed path.
  For this option, alternative paths through the network need to exist that show the same performance characteristics, e.g. delay or possible throughput, in order to avoid affecting the service quality after a failover occurs. In data centre environments, this is typically given via the redundant connectivity between ToR, aggregation and core switches.

While these solutions can re-establish connectivity on short timescales, in general the failure of a physical link or the ports it is attached to means the loss of all data in flight over the link at the moment of the failure until the uplink node recognizes it. Thus, some of the application connections using the link are affected, even if only by having to retransmit data over a TCP connection. This can only be avoided by transmitting the same packets redundantly over several links in parallel, causing significant overhead and additional resource consumption.

7 Reliability estimation models

7.1 Basic redundancy methods and reliability/availability estimation

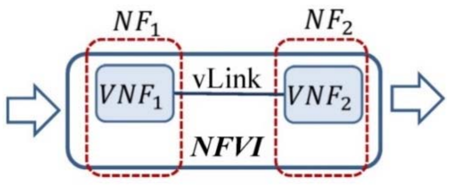
7.1.1 Sub network (or a chain of NFs)

As described in clause 4, an end-to-end path is a chain of NFs that includes multiple VNFs and links in-between. A sub network can be defined as a subset of the end-to-end path and is also a chain of NFs which includes some VNFs and links. Thus, the smallest sub network would include two NFs and a link between them. It is assumed that a network service (NS) can go through different sub networks and the end-to-end reliability of this NS is estimated by combining the reliability of each sub network. A NF is composed of one (or several) VNF running on an NFVI (hardware, virtualisation layer) in the NFV environment.

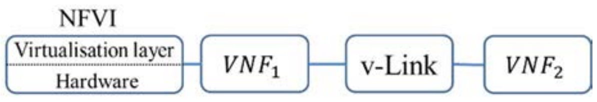
NOTE: The case of NS comprising PNFs is not treated here for the sake of clarity.

The links refer to virtualised or/and non-virtualised network resources that support communication facilities between two NFs (e.g. NFVI (network) represented in Figure 56). The reliability of a sub network is thus defined by including the different element reliability, i.e. VNFs and NFVIs.

7.1.2 Reliability/availability of a sub network



(a)



(b)

Figure 55: (a) A sub network SN1 using a single NFVI and (b) its reliability block diagram

How to estimate the reliability of a NF and its associated NFVI was presented in clause 5. Figure 55(a) shows an example of a sub network in the NFV environment. In this scenario, two NFs (composed of two different VNFs) are deployed on a single NFVI and there is a virtual link (v-Link) for communication between the two NFs. In order to provide the service, all the elements in this sub network need to work simultaneously and properly, as shown in Figure 55(b). The sub network reliability is the probability that all elements execute correctly and it can be estimated by:

$$R_{SN1} = R_{NFVI} * R_{VNF1} * R_{VNF2} * R_{v\text{-}Link} \qquad (7.1)$$

If the VNFs of a sub network are deployed on separate NFVIs as shown in the Figure 56, the reliability of the link, (i.e. NFVI (network)) needs to be considered.

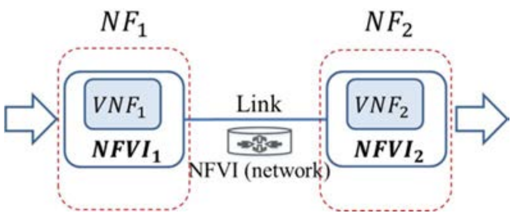


Figure 56: A sub network SN2 using different NVFIs

Then, the reliability of sub network for this scenario can be:

$$R_{SN2} = R_{NFVI\text{-}1} * R_{NFVI\text{-}2} * R_{VNF1} * R_{VNF2} \qquad (7.2)$$

7.1.3 Sub network with redundancy configurations

Redundancy is widely used to improve the reliability and availability of a system. The most common approaches are Active-Active and Active-Standby configurations. This clause describes various redundancy configurations for sub networks rather than VNFs and how these configurations provide/improve the overall reliability/availability.

a) Reliability/availability estimations with Active-Active redundancy configurations

The common sense of Active-Active configurations is that multiple functionally equivalent units run in parallel and are highly synchronized. When a unit fails, the other unit(s) continues (continue) serving, thus no service interruption occurs. If ***n*** units are running in parallel, the system can withstand up to (***n-1***) units failures. There are multiple options for Active-Active configurations and each provides a different availability/reliability level.

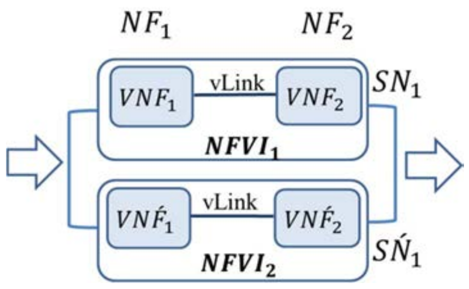


Figure 57: An example of 1:1 Active-Active configuration: two sub networks SN1 and $S\acute{N}_1$ are running in parallel

Figure 57 shows an example of Active-Active configuration of a sub network. As seen in the figure, two sub networks SN1 and $S\acute{N}_1$ are running in parallel and providing the same service, i.e. 1:1 Active-Active configuration (let this configuration called AA1 for future references). When a sub network, or any element of a sub network, fails, the other active unit can continue to provide the service. Service interruption occurs when both sub networks fail, or at least one element fails in each sub network. The unreliability or the probability that both sub networks fail at the same time is:

$$F_{AA1} = F_{SN1} * F_{SN'1} \qquad (7.3)$$

where $F_{SN1} = F_{SN'1} = 1 - (R_{NFVI} * R_{VNF1} * R_{VNF2} * R_{vLink})$. And the overall reliability becomes:

$$R_{AA1} = 1 - F_{AA1} \qquad (7.4)$$

If VNFs of a sub network are deployed on different (separate) NFVIs, it is also possible to deploy two active VNFs on the same host NFVI (AA2), as shown in Figure 58. Then, the reliability of the VNFs is improved by $R_{VNFi} = 1 - F_{VNFi} * F_{VNF'i}$, $i = \{1,2\}$. The reliability of each NF becomes:

$$R_{NFi} = R_{NFVI} *(1 - F_{VNFi} * F_{VNF'i}), i = \{1,2\} \qquad (7.5)$$

The overall reliability of the sub network is:

$$R_{AA2} = R_{NF1} * R_{Link} * R_{NF2} \qquad (7.6)$$

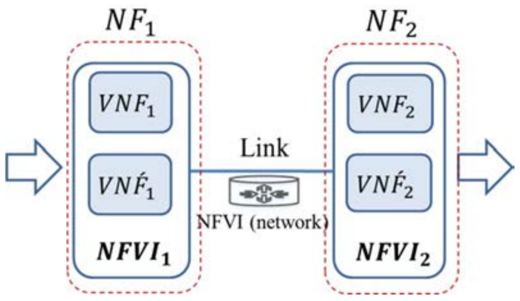


Figure 58: Active-Active redundancy for VNFs on the same NFVI

Note that, in this configuration, according to equation (7.5), the reliability of the NF is dependent on the NFVI and limited by that of the NFVI. In other words, the reliability of a NF is never greater than or equal to that of the host NFVI because $0 \leq R \leq 1$. This kind of configuration is applicable when the NFVI reliability is reasonably high.

By following the anti-affinity rule, active VNFs could be deployed on independent (physically separated) NFVIs, as shown in Figure 59 (this configuration is called AA3). In this configuration, the reliability of each NF is:

$$R_{NFi} = 1 - (F_{NFi} * F_{NF'i}), \; i = \{1,2\} \qquad (7.7)$$

where, $F_{NFi} = F_{NF'i} = 1 - (R_{VNFi} * R_{NFVIj})$.

And the reliability of the sub network can be estimated by:

$R_{AA3} = R_{NF_1} * R_{link} * R_{NF_2}$

$$R_{AA3} = R_{NF1} * R_{Link} * R_{NF2} \qquad (7.8)$$

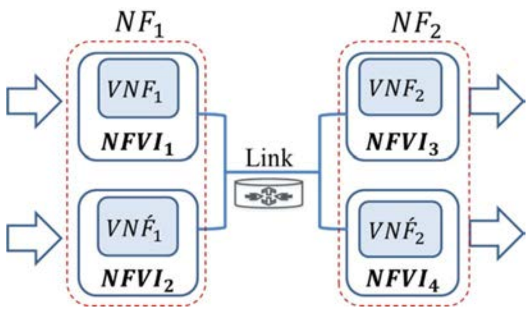


Figure 59: Active-Active configuration for VNFs on different NFVIs.

Here again, the reliability of the sub network is limited by that of the link in-between see equation (7.8). It can be referred as weakest-link-in-the-chain.

However, redundancy for the vLink/links may also be configured. For example, AA redundancy not only for the NFs but also for the link is shown in Figure 60 (this configuration is called AA4). Then the overall sub network reliability of the SN can be estimated as:

$$R_{AA4} = (1 - \prod F_{NF1}) * (1 - \prod F_{Link}) * (1 - \prod FR_{NF2}) \qquad (7.9)$$

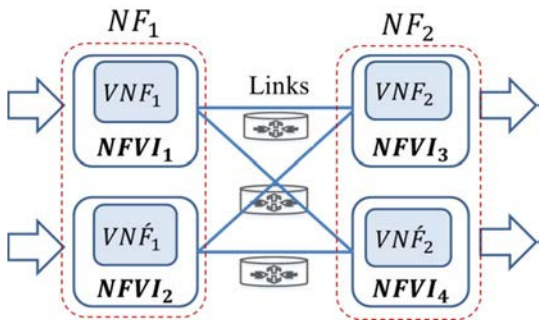


Figure 60: Active-Active redundancy for all elements of a sub network

The Active-Active configuration performance is dependent on the number of independent active units running in parallel. For instance, if *n* active units are implemented in parallel, the system reliability will be:

$$R = 1 - F^n \qquad (7.10)$$

Thus, $\boldsymbol{R \rightarrow 1}$ when $\boldsymbol{n \rightarrow \infty}$**;** the more active units run in parallel, the more reliable the system is. However the cost would be the major concern for this kind of configuration.

b) Reliability/availability with Active-Standby redundancy configurations

The availability of a system is defined as equation (5.1) and can be improved by either increasing the MTBF or decreasing the MTTR. Generally, Active-Active redundancy increases the reliability and, it thus increases the MTBF. The Active-Standby redundancy helps reducing the MTTR significantly. Either way, the availability is improved.

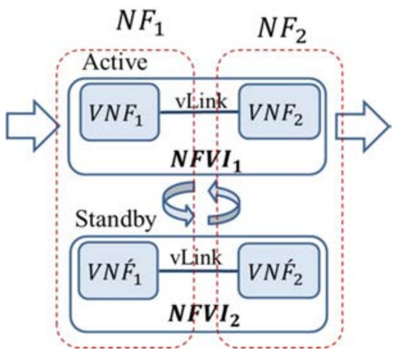


Figure 61: 1:1 Active-Standby configuration

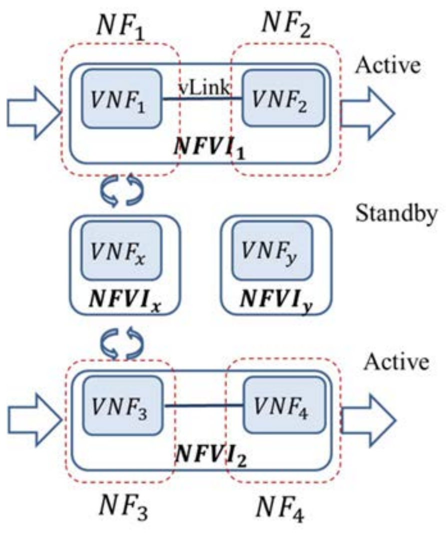


Figure 62: N:M Active-Standby configuration (N=2, M=1)

In the Active-Standby configuration, only one unit (active) is in operation and the other unit (standby) sits idle. When the active unit fails, the standby unit assumes the role of active unit. Therefore, the operation cost is less than that of Active-Active configuration. The active and standby units could be deployed on the same (different) NFVI(s) and various Active-Standby configurations are also possible: one standby unit for one active unit as shown in Figure 61 (1:1 Active-Standby), and M standby units for N active units (N: M Active-Standby) as described in Figure 62.

The performance of Active-Standby configurations depends on the MTTR values, i.e. how fast the failed unit can be recovered. Therefore, according to equation, A= MTBF/(MTBF+MTTR), ***A →1*** if ***MTTR →0***.

The NFV-MANO system reliability will play a vital role in this configuration. With the help of NFV-MANO system, the failure detection and failover mechanisms might be more efficient. Note that the Active-Standby redundancy cannot improve the reliability (or MTBF) of a system.

7.1.4 Reliability/availability of end-to-end paths

An end-to-end path is composed of multiple sub networks (or chain of sub networks) with their corresponding redundancy configurations, as shown in Figure 63. The reliability/availability of the sub networks and the elements can be estimated as mentioned above. Consequently, the reliability of an end-to-end path could also be estimated as:

$$R_x = \prod_i R_{SNi} * \prod_i R_{Link} \qquad (7.11)$$

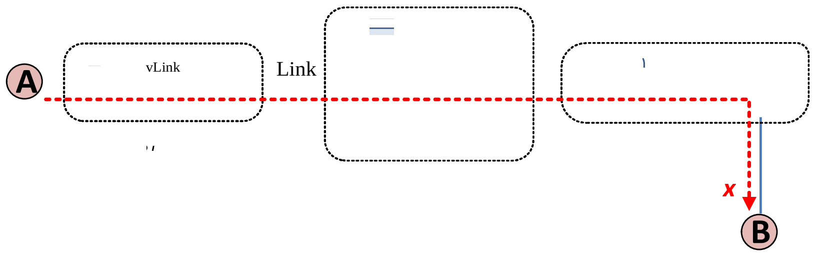


Figure 63: An end-to-end path

Summary

Generally, the performance of AA redundancy is defined by the number of active units running in parallel. The performance of Active-Standby configurations depends on the total recovery time which includes the failure detection time, the repairing/replacing time and so on. The NFV-MANO system availability will play a vital role in both Active-Standby and Active-Active configurations. The reliability/availability of an end-to-end path could be estimated by combining the reliability/availability of sub networks and the links along the path, and is limited by the least reliable components, i.e. the weakest-in-the-chain.

7.2 Lifecycle operation

7.2.1 Introduction

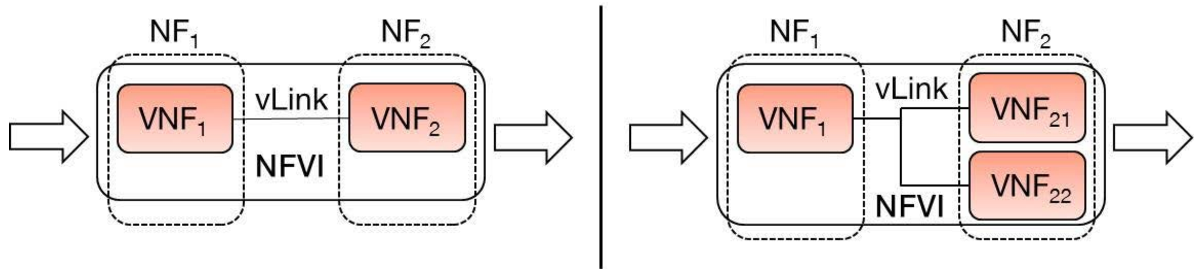
Predicting standardized, objective and quantitative performance of key quality measurements like availability (based on service outage downtime) or reliability (based on metrics such as call completion rate, dropped call rate) on the basis of estimated values of standardized, objective quality performance measurements enables service providers to better engineer the quality of their network services. This clause represents a basic approach to start the discussion. Much more study needs to be done to solve the estimation problem, including what are the right standardized parameters for the inputs needed to derive the estimates for the feasible availability (e.g. measurement of service uptime) and reliability (e.g. measurement of the system's ability to perform its key functionality such as processing and preserving calls). This clause illustrates one way reliability and availability can be estimated for a network service (NS) comprising different VNFs which are designed in a redundant way, or not. Besides exploiting representations and formulas introduced in previous clauses, the consideration of NS lifecycle operations, e.g. VNFs scaling, is the essential new aspect introduced here.

7.2.2 Network service without redundancy

Based on the representations and hypotheses provided in clause 7.1 (e.g. NFVI comprises all the hardware equipment useful for running NSs), this simple case helps to express the basic assumptions and equations needed for the estimation:

- a NS consists of VNF_1 and VNF_2 and is designed without redundancy;
- both VNFs are deployed on the same NFVI;
- VNF_1 is not impacted by the traffic load, while VNF_2 needs to be scaled out (one time) if the input load is over a certain pre-defined threshold (Figure 64).

NOTE 1: It is assumed that a load balancing function is embedded in VNF_1 allowing to distribute the traffic to the two VNF_2 instances after the scaling out process.



a) Input load below the scaling threshold b) Input load over the scaling threshold

Figure 64: A network service without redundancy before and after the scaling process

Migration - see equation (7.13) - for maintenance purposes, or due to hardware (e.g. blade) failure restoration, consists of moving virtual resources (e.g. CPU) of a VNF from one physical resource to another.

NOTE 2: For the sake of simplicity, the migration is considered to happen in the same infrastructure, e.g. from one blade to another.

Three functional components of NFV-MANO are needed to manage the operations (start, run, and stop) of an NS: NFVO, VNFM, and VIM. The model considered here is the one where NFVO is used to launch the network service and to supervise, a posteriori, the resource allocation, i.e. finding/allocating resources for VNFs is done directly by the VNFM. For the sake of simplicity, it is also assumed that granting of lifecycle management requests is not performed. In other words, NFVO's reliability is not considered in what follows, while VNFM and VIM are useful for scaling and migration.

Degradation of the VNFs service reliability or availability can result from two main types of faults:

- intrinsic VNF software: this is modelled by the VNF software reliability (R_{VNF});

- VNF running environment (e.g. VM, container): this is due to the virtualisation layer, expressed by R_{VIM}, and/or the hardware on which the VNFs run, i.e. infrastructure represented by R_{NFVI}.

NOTE 3: The use of R_{VIM}* R_{NFVI} to express the VNF running environment is a rough approximation - further studies are needed to characterise more precisely this contribution to the overall reliability.

Failures detection and service restoration mechanisms have to be taken into account in these reliability estimations inputs (i.e. R_{VNF}, R_{VIM}, R_{VNFI}) in order for the models to evaluate the defects' impact (e.g. call failures) and the system availability. In the cases in which scaling of the VNFs is an integral part of the service availability, the probability of each of the required components to successfully complete the scaling will need to be included in the models. This reasoning also holds for service restoration, e.g. using migration, in the event of failure, including the time required to do so.

Following the equations expressed in clause 7.1, the NS' reliability is thus:

- before the scaling:

$$R_{NS} = R_{VIM} * R_{NFVI} * R_{VNF1} * R_{VNF2} * R_{vLink} \qquad (7.12)$$

$$R_{NS} = R_{VIM} * R_{NFVI} * R_{VNF1} * R_{VNF2} * R_{vLink} * R_{VNFM} \text{ (if migration is involved)} \qquad (7.13)$$

where R_{VIM} and R_{VNFM} represent the probability that operations useful for the migration (e.g. detection of the migration need, e.g. blade breakout, migration launching, …) are successful.

when scaling out is needed:

$$R_{NS} = R_{VIM} * R_{NFVI} * R_{VNF1} * R_{VNF21} * R_{VNF22} * R_{vLink} * R_{VNFM} \text{ (migration involved, or not)} \qquad (7.14)$$

NOTE 4: The assumption used in (7.14) is that the input traffic needs to be integrally serviced, i.e. both VNF_{21} and VNF_{22} are needed in the reliability/availability estimation - the case of input traffic partially serviced, e.g. customers treated by VNF_{22}, can be easily deducted, i.e. removing R_{VNF22} from (7.14).

NOTE 5: For the sake of simplicity, the probability to get new resources needed for the migration or the scaling out process is considered as part of R_{NFVI}.

NOTE 6: NFV-related operations such as NS/VNF on-boarding are not taken into account here, as reliability and availability start to be considered only after the service has started running at time t_0.

NOTE 7: The previous equations do not express that the VNFM and VIM services are only requested during a short period (for scaling or migration). Further studies are needed to characterize this real situation, although in the PNF framework, this practice is common, e.g. for an active-standby system, the modelling includes the probability that the switch over (from active to standby) is reliable all the time, even if the switching is not required continuously.

As the previous equations (7.12), (7.13) and (7.14) are all applicable (the correct use only depending on the phase of the lifecycle operations), one solution is to choose the minimum reliability over all three, i.e. a worst-case assumption:

$$R_{NS} = \min \{ (7.12), (7.13), (7.14) \} \qquad (7.15)$$

NOTE 8: Although scaling and migration times obey to random variables, further studies are needed to find better solutions than this worst case one.

NOTE 9: The scaling out process can be performed within a VNF, e.g. VNF_2 comprises two VNFCs ($VNFC_1$ and $VNFC_2$) with scaling possibility for one VNFC (e.g. $VNFC_2$). In this case, equations (7.12) and (7.13) are unchanged, while equation (7.14) can be written:

$$R_{NS} = R_{VIM} * R_{NFVI} * R_{VNF1} * R_{VNF2} * R_{vLink} * R_{VNFM} \text{ (migration involved, or not)} \qquad (7.16)$$

NOTE 10: This R_{VNF2} value is related to another type of VNF_2 and should not be mistaken with the one used in previous equations.

As indicated in clause 5.2.1, for any system, once R is obtained, if the exponential law is held for true, the MTBF can be computed using: $R(t) = e^{-t/MTBF}$.

If the MTTR is known or estimated, the system's availability A can be calculated: $A = \frac{MTBF}{MTBF+MTTR}$

As the availability of an architecture composed of elements in series can be computed the way it is done for its reliability, i.e. by multiplying all elements' availability, A_{NS} related to equations (7.12), (7.13) and (7.14) can thus be obtained, e.g. the availability related to (7.12) is:

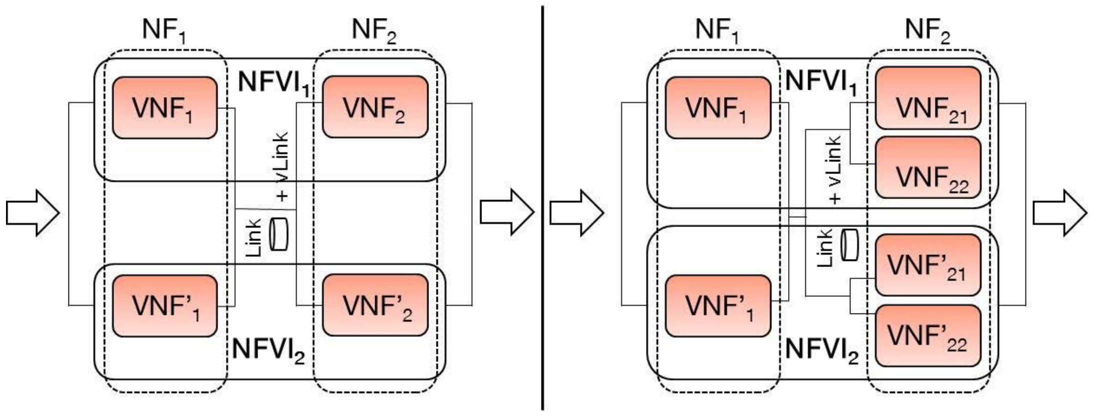
$$A_{NS} = A_{NFVI} * A_{VNF1} * A_{VNF2} * A_{vLink} \qquad (7.17)$$

NOTE 11: As for (7.15), a solution for the final network service availability is to choose the smallest value of all A_{NS} from the scenarios of equations (7.12), (7.13) and (7.14).

7.2.3 Network service with redundancy

The following second example of a network service is more resilient:

- the NS is made of VNF_1 and VNF_2;
- its design incorporates full redundancy;
- while the 1st instances (VNF_1 and VNF_2) are deployed on $NFVI_1$, the redundant ones (VNF'_1 and VNF'_2) are deployed on $NFVI_2$;
- VNF_1 is not impacted by the traffic load, while VNF_2 needs to be scaled out (one time) while respecting the initial redundancy if the input load is over a certain pre-defined threshold (Figure 65).



a) Input load below the scaling threshold b) Input load over the scaling threshold

Figure 65: A network service with redundancy before and after the scaling process

In the same way as above, the NS's reliability and availability can be computed taking into account the critical components and the probability of each contributing to successful service fulfilment, including service restoration in the event of failure and the time required to do so. With a more resilient architecture that includes redundancy, service restoration is generally much faster as the redundant components can recover the service faster and more reliably than a service restoration obtained through migration to take over the service:

- Before the scaling:

  $R_{NS} = R_{NF1} * R_{NF2} * R_{Link} * R_{vLink}$

  Using:

  $F_{NF1} = F_{VNF1} * F_{VNF'1} = (1 - [R_{VIM1} * R_{NFVI1} * R_{VNF1} * R_{vLink}]) * (1 - [R_{VIM2} * R_{NFVI2} * R_{VNF'1} * R_{vLink}])$

  $F_{NF2} = F_{VNF2} * F_{VNF'2} = (1 - [R_{VIM1} * R_{NFVI1} * R_{VNF2} * R_{vLink}]) * (1 - [R_{VIM2} * R_{NFVI2} * R_{VNF'2} * R_{vLink}])$

  the reliability of NF_1 and NF_2 can be written:

  $R_{NF1} = 1 - F_{NF1} = 1 - (1 - [R_{VIM1} * R_{NFVI1} * R_{VNF1} * R_{vLink}]) * (1 - [R_{VIM2} * R_{NFVI2} * R_{VNF'1} * R_{vLink}])$

  $R_{NF2} = 1 - F_{NF2} = 1 - (1 - [R_{VIM1} * R_{NFVI1} * R_{VNF2} * R_{vLink}]) * (1 - [R_{VIM2} * R_{NFVI2} * R_{VNF'2} * R_{vLink}])$

Then:

$$R_{NS} = \{ 1 - (1 - [R_{VIM1} * R_{NFVI1} * R_{VNF1} * R_{vLink}]) * (1 - [R_{VIM2} * R_{NFVI2} * R_{VNF'1} * R_{vLink}]) \} *$$
$$\{ 1 - (1 - [R_{VIM1} * R_{NFVI1} * R_{VNF2} * R_{vLink}]) * (1 - [R_{VIM2} * R_{NFVI2} * R_{VNF'2} * R_{vLink}]) \} * R_{Link} * R_{vLink} \quad (7.18)$$

This formula becomes as follows if migration is involved:

$$R_{NS} = \{ 1 - (1 - [R_{VIM1} * R_{NFVI1} * R_{VNF1} * R_{vLink} * R_{VNFM}]) * (1 - [R_{VIM2} * R_{NFVI2} * R_{VNF'1} * R_{vLink} * R_{VNFM}]) \} *$$
$$\{ 1 - (1 - [R_{VIM1} * R_{NFVI1} * R_{VNF2} * R_{vLink} * R_{VNFM}]) * (1 - [R_{VIM2} * R_{NFVI2} * R_{VNF'2} * R_{vLink} * R_{VNFM}]) \} * R_{Link} * R_{vLink} \quad \mathit{(7.19)}$$

- When scaling out is needed (migration involved, or not):

$$R_{NS} = \{ 1 - (1 - [R_{VIM1} * R_{NFVI1} * R_{VNF1} * R_{vLink} * R_{VNFM}]) * (1 - [R_{VIM2} * R_{NFVI2} * R_{VNF'1} * R_{vLink} * R_{VNFM}]) \} *$$
$$\{ 1 - (1 - [R_{VIM1} * R_{NFVI1} * R_{VNF21} * R_{VNF22} * R_{vLink} * R_{VNFM}]) * (1 - [R_{VIM2} * R_{NFVI2} * R_{VNF'21} * R_{VNF'22} * R_{vLink} * R_{VNFM}]) \} * R_{Link} * R_{vLink} \quad \mathit{(7.20)}$$

As well as indicated in the previous case, a solution to combine (7.18), (7.19) and (7.20) is:

$$R_{NS} = \min \{ (7.18), (7.19), (7.20) \} \quad \mathit{(7.21)}$$

NOTE: Scaling out within a VNF can be treated the way equation (7.16) was established.

The availability can also be determined by A = $\frac{MTBF}{MTBF+MTTR}$, with R(t) $= e^{-t/MTBF}$. As this example deals with redundancy, the formula to be used for multiple components in parallel is given in clause 5.2.2, i.e. the availability related to (7.17) is:

$$A_{NS} = A_{NF1} * A_{NF2} * A_{Link} * A_{vLink} \quad \mathit{(7.22)}$$

with:

$$A_{NF1} = 1 - ([1 - A_{VIM1} * A_{NFVI1} * A_{VNF1} * A_{vLink}] * [1 - A_{VIM2} * A_{NFVI2} * A_{VNF'1} * A_{vLink}])$$

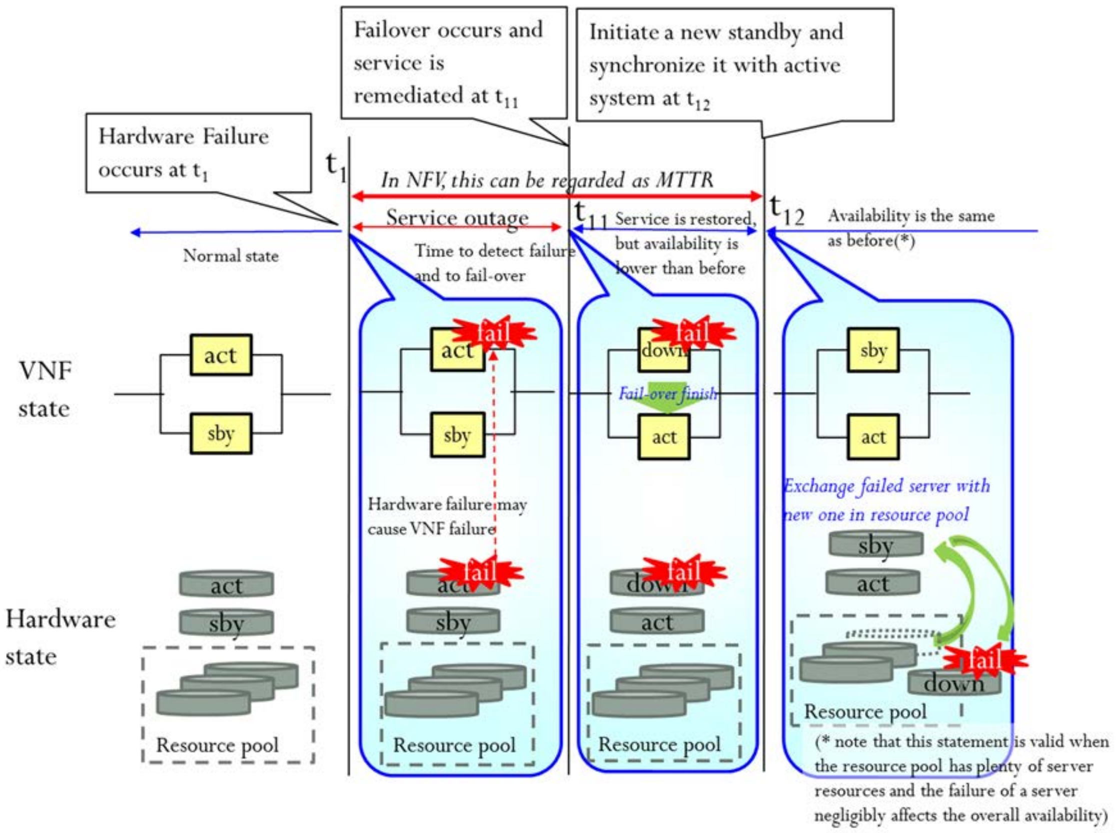
$$A_{NF2} = 1 - ([1 - A_{VIM1} * A_{NFVI1} * A_{VNF2} * A_{vLink}] * [1 - A_{VIM2} * A_{NFVI2} * A_{VNF'2} * A_{vLink}])$$

Standardized, objective and quantitative NFV service quality metrics are a key input into modelling both service reliability as well as service availability. Metrics such as premature VM release ratio and VM stall are possible inputs to the contributions of the NFV infrastructure to VNF service quality performance. Individual telecom providers and their suppliers will likely have proprietary reliability and availability modelling techniques that are used to produce useful estimates of the end user service qualities delivered by their offerings. However, it is important that the models are generated using consistent definitions of the inputs and outputs measurements to assure that standard parameter estimates and predictions are used as modelling inputs, and later those predictions can be validated and calibrated against actual performance to continuously refine those values. Standardization of important objective and quantitative quality metrics will continue to be worked by standards development organizations such as TM Forum, QuEST Forum, NIST and ETSI NFV.

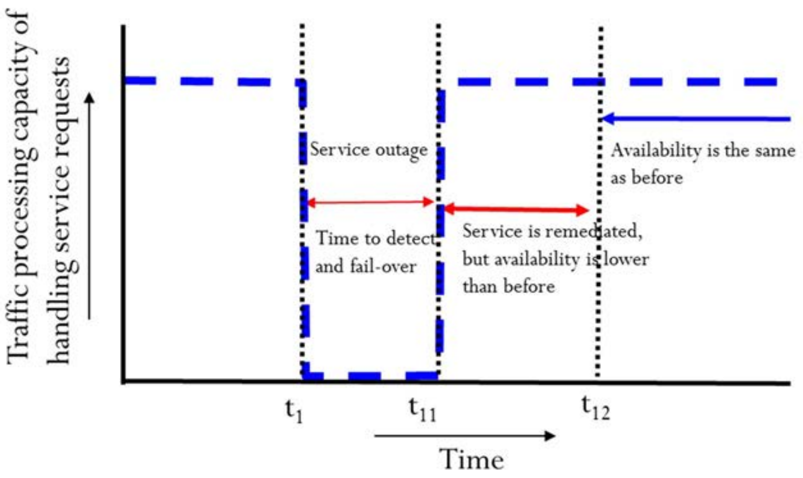
7.2.4 Reliability and availability owing to the protection schemes

As shown in clause 6.1, failure detection and remediation/recovery procedures are required in order to achieve high reliability and availability. This clause describes the dependency of the elements in terms of reliability and availability during the lifecycle of NFV environments.

Figure 66 and Figure 67 show the investigation of (a) the situation after a hardware failure occurs at a server on which active VNF resides, and (b) the change of traffic processing capacity to handle service requests after a failure occurs for Active-Standby and Active-Active configurations, respectively. In both cases, it is assumed that after the detection of a failure, the traffic destined to the failed VNF goes to the other active VNF at t_{11} thanks to, e.g. a switch or a load balancer, etc. placed in front of the system.

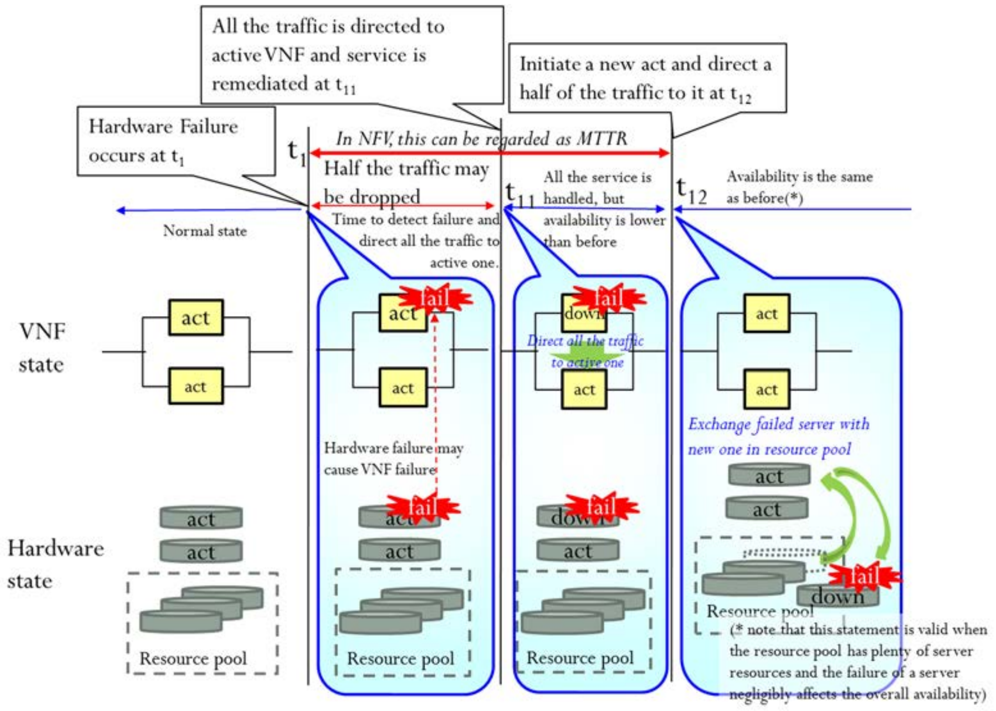

Failover occurs and service is remediated at t11
Initiate a new standby and synchronize it with active system at t12
Hardware Failure occurs at t1
t1
In NFV, this can be regarded as MTTR
Service outage
t11 Service is restored, but availability is lower than before
t12 Availability is the same as before(*)
Normal state
Time to detect failure and to fail-over
VNF state
act
sby
fail
down
Fail-over finish
Hardware failure may cause VNF failure
Exchange failed server with new one in resource pool
Hardware state
Resource pool
(* note that this statement is valid when the resource pool has plenty of server resources and the failure of a server negligibly affects the overall availability)

(a) Situation after a hardware failure occurs at a server on which the active VNF resides.

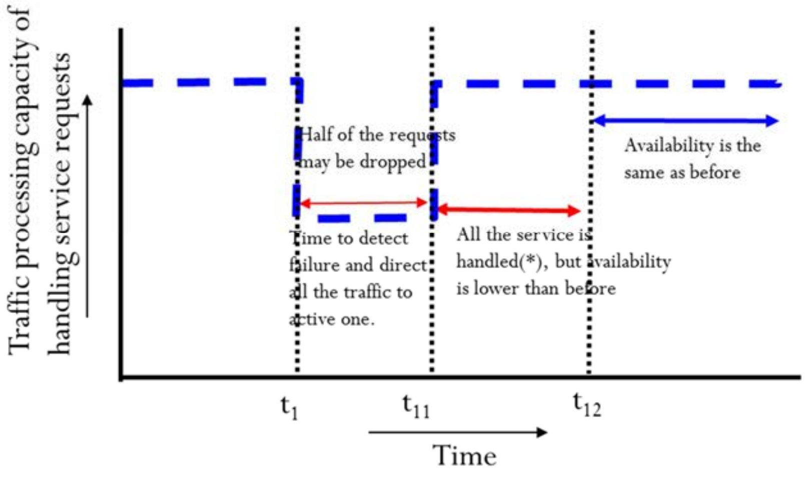


(b) An example of traffic processing capacity change for providing service in remediation/recovery after a failure occurs (Active-Standby configuration).

Figure 66: Investigation on Active-Standby protection scheme after a hardware failure occurs


All the traffic is directed to active VNF and service is remediated at t_{11}
Initiate a new act and direct a half of the traffic to it at t_{12}
Hardware Failure occurs at t_1
In NFV, this can be regarded as MTTR
Half the traffic may be dropped
Time to detect failure and direct all the traffic to active one.
All the service is handled, but availability is lower than before
Availability is the same as before(*)
Normal state
VNF state
Hardware state
act
fail
down
Direct all the traffic to active one
Exchange failed server with new one in resource pool
Hardware failure may cause VNF failure
Resource pool
(* note that this statement is valid when the resource pool has plenty of server resources and the failure of a server negligibly affects the overall availability)

(a) Situation after a hardware failure occurs at a server on which one of the active VNFs resides.



(b) An example of traffic processing capacity change for providing service in remediation/recovery after a failure occurs (Active-Active configuration).

Figure 67: Investigation on Active-Active protection scheme after failure occurs

Note that in the Active-Active case of Figure 67, it is assumed that the traffic is distributed evenly among the two active VNF instances, each of which handles, at most, the half of its traffic processing capacity, i.e. total amount of the traffic does not exceed the traffic processing capacity of a single VNF instance. When both VNFs handle more than 50 % of their traffic processing capacity before t_1, some traffic will be dropped after t_{11} since only one active VNF handles the traffic imposed on the system.

In these figures, the time between t_1 and t_{11} depends on the failure detection and fault isolation mechanisms. Failures can be detected by the NFV elements, such as VNF, NFVI and NFV-MANO though the detectable failures may vary, but all of those elements should be available to detect the failure. In other words, the detection mechanism should be prepared in VNF, NFVI and NFV-MANO since the availability of fault detection mechanism depends on the availability of these functions. It is also better to shorten the detection time to reduce the time for service outage or service capacity degradation. To shorten the detection time usually consumes computing resources. To avoid complete service outage or to decrease the degradation level of service capacity, load balancing configuration may be effective since the fraction affected by the failure of a single VNF decreases.

NFV-MANO directs the network traffic so that it does not go to the failed VNF, thus justifying the need of a highly reliable NFV-MANO. Some mechanisms can be implemented to steer traffic from the failed VNF to active VNF(s) autonomously, for example, with the help of a load balancer.

The remediation phase, which spans from t_{11} to t_{12}, lasts until NFV-MANO prepares a new VNF and reconfigures the system putting it back in the initial configuration, except for the resource pool. Though the service is properly provided in this phase, the availability of the system is lower than that in the initial configuration, and a new standby, or a redundant, VNF is expected to be prepared rapidly. Since it takes some time to initiate a new VNF to restore the system to the initial state, it is better to prepare some redundant VNFs (e.g. VMs) and to implement a mechanism that enables the system to incorporate the redundant VNFs autonomously for reducing the time between t_{11} and t_{12}.

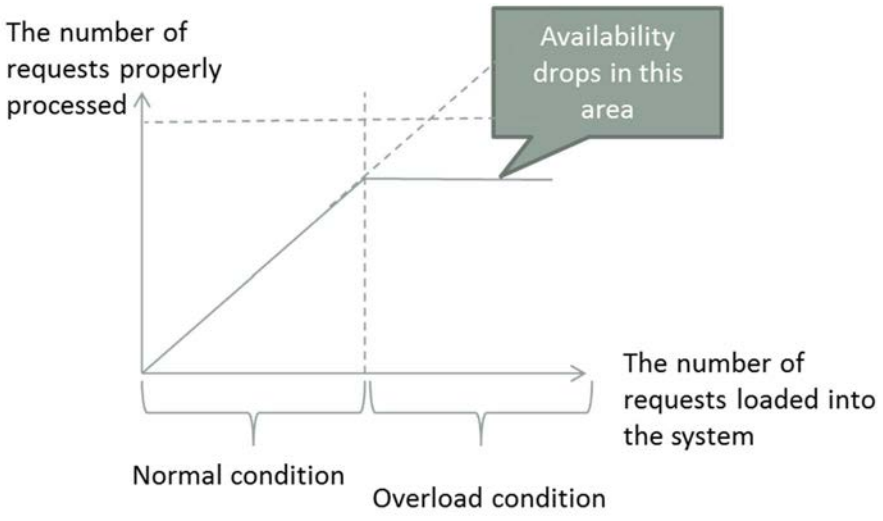
7.2.5 Reliability and availability during load fluctuation

In the VNF lifecycle, traffic load varies statistically in normal situation. The average load increases according to the number of the users who enjoy the service or to the frequency of the service requests per user. Traditionally, service providers/operators may predict the tendency on the number of service requests and prepare resources for the service in ahead.

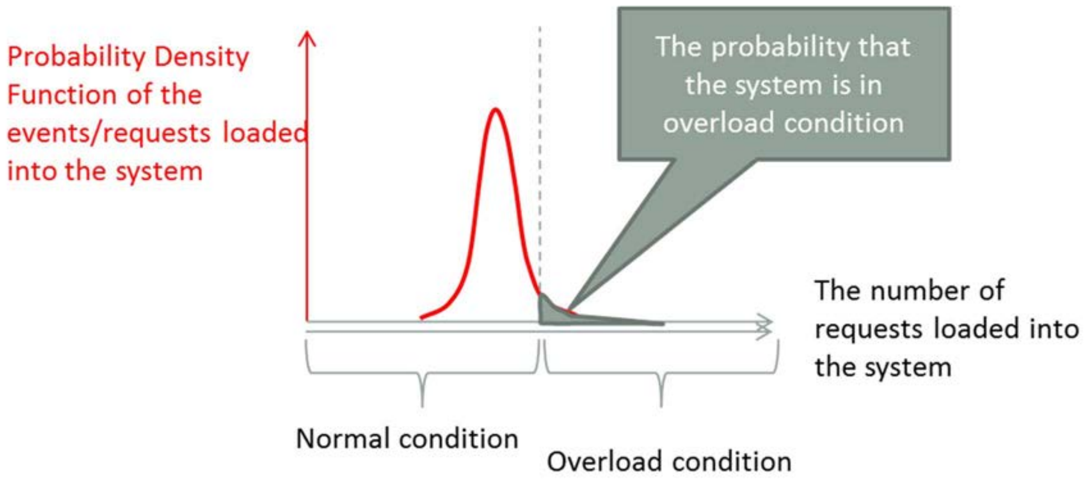
For example, the number of the service requests handled properly within a unit time is ideally the same as that of the service requests loaded into the system if the number of requests is under the threshold of congestion control as shown in Figure 68 (a). However, when the number of requests exceeds this threshold, the system will drop/reject a certain amount of events because of the congestion control mechanism's limits, in order to keep the system under control, which means that the availability falls down within this area.

Figure 68 (b) represents the probability density function, *Pr(n)*, that the number of service requests applied to the system within a unit time is *n*. When the number of requests per unit of time is large enough, it is normally-distributed (Gaussian curve). Since the integral of *Pr(n)* from the threshold to infinite, where congestion control is activated, represents the probability that the system is in the overload condition, this probability means the probability that the system cannot handle all the requests and is thus not fully available. When estimating availability, this part has to be taken into consideration.

In NFV systems, scaling-out (or scaling-up) is relatively easy and rapidly realized thanks to the virtualisation techniques and the NFV-MANO functionalities compared to the traditional PNFs, where scaling out or scaling up requires time for purchase and delivery of new hardware.



(a) Number of requests properly processed



(b) The probability density function for the number of requests loaded into the system

Figure 68: Number of requests properly processed and probability density function for the number of requests loaded into the deployed NF

Since scaling-up and scaling-out need NFV-MANO functionalities, availability of the system depends on that of NFV-MANO in this situation. However, NFV-MANO can deploy a new scaled-up VNF or a new VNF rapidly when an integral of the probability that requests cannot handled properly exceeds a certain value. Therefore, this probability can be neglected for the availability calculation if the resource pool is large enough and the NFV-MANO is reliable enough. It implies that the planning process for investment of NFs becomes easier than in traditional environment.

Note that the scaling-out may be better to increase availability, since, as shown in clause 6.2.3, scaling-up procedures might need switchover, which may cause the degradation of service during the procedure.

8 Reliability issues during NFV software upgrade and improvement mechanisms

8.1 Service availability recommendations for software upgrade in NFV

In an NFV environment, there are many software components, e.g. service software, middleware, guest operation system (OS), virtualisation or hypervisor software, firmware, device drivers and software of networking components for virtual links. Since the software will evolve over time, each of those components will require to be updated or upgraded in order to fix bugs and/or to add new features. Telco services have very strict requirements on the service availability and service continuity from the end users' point of view. It is expected that when the network functions need to be upgraded, the network service should still operate and provide communication services without any disruption detected by end users. However, the data structure of new software versions might not be compatible with the one of old software versions. This means that while new and old software versions could be running at the same time (but not in the working mode simultaneously), however they might not be able to provide the service at the same time because of the inter-operability issue between two versions. In addition, since network traffic is typically connection oriented and stateful, the state information (e.g. context information) has to be preserved during an upgrade in order to avoid disrupting existing connections, which might cause service interruption to the end users. If the data formats used by the old and new software are incompatible, data conversion from the old to the new format has to be performed to keep the service continuity. However, data conversion tends to be tedious and requires an exact knowledge of the differences between the two software versions and the conversion process is known to be error-prone. The conversion of stateful data with high dynamics could result in some unintended service disruptions.

Recommendation 1: Software upgrade operations should not cause the decrease of service availability and the interruption of network service (i.e. the disruption of ongoing connections and sessions).

Recommendation 2: The software upgrade will need to update network service and/or VNFFG. Such an update operation should not cause the interruption of network service (i.e. the disruption of on-going connections and sessions).

In an NFV environment, the different layers are decoupled from each other. The upgrade of the software of each layer should be able to be done separately without impacting service availability and service continuity.

Recommendation 3: The VNF software, and the infrastructure software modules (e.g. virtualisation layer or hypervisor, virtual switches, firmware software, etc.) should be upgraded separately without causing the decrease of service availability and the interruption of network service (i.e. the disruption of ongoing connections and sessions).

In an NFV environment, since elastic resources are available for the software upgrade process, the traditional software upgrade methods (e.g. rolling upgrade with data conversion) seems not to be very efficient to use the cloud technology. The data conversion process has been shown to be error-prone resulting in severe service outages**.** Outage resulting from software upgrade in current systems is a well-known issue for network operators. According to CRN [i.3] and [i.4], 30 % of the 10 biggest cloud outages in both 2013 and 2014 are reported to be caused by upgrade, although the main cause of the outages is not given in detail. One problem of the traditional rolling upgrade method is that the new software version and its corresponding network configuration have not been verified using real traffic before switching the users' traffic from the old software version to the new one.

Recommendation 4: The software should be upgraded in a scalable way, e.g. upgrading a fraction of the whole capacity, a certain service type or a certain user group, with the constraint of preserving the service availability and service continuity.

Recommendation 5: The upgraded software with its corresponding configurations should be verified with real service traffic while the existing network service traffic is processing normally.

According to NFV prerequisite [i.2], a failure at the NFV-MANO system should not affect any existing VNF instances. The upgrade of NFV-MANO software should not affect the service availability and continuity of the provided network service.

Recommendation 6: The upgrade of the NFV-MANO software, and an operation in NFV-MANO for supporting software upgrade as well, should not have any impact on the availability and continuity of existing network services.

8.2 Implementation of NFV software upgrade and service availability

8.2.1 VNF software upgrade

8.2.1.1 Software upgrade of traditional network functions

During the software development cycle the new software version might not be compatible with the old one. The backward incompatibility might result that the new and old software versions cannot run at the same time. Incompatible software versions in the upgrade network function might not behave as intended to communicate with other endpoints. In addition, existing traffic processing in a network function, e.g. GTP protocol processing according to specifications of 3GPP, needs to follow the affinity preferences. The switching of network function used for traffic processing might cause significant end-to-end signalling surge related to the change of the network function address. Thus, the state information (e.g. context information) need to be preserved during an upgrade in order to avoid disrupting existing connections which might cause service interruption to the end users.

In the case of mobile network systems, typically, the following three issues should be considered by software upgrade for maintaining service availability and service continuity:

- User attachment affinity: User attachment is almost a long term process as the power-off of mobile phones is very rare. When a user is attached to the network, the requested service might be assigned to a network function for providing the service. The migration to another network function for service is normally end-user initiated (e.g. power off, hand-over). A big volume of migrations might cause significant end-to-end signalling traffic surge.

- Traffic flow affinity: Traffic is typically stateful and the user traffic flow needs to be processed where the stateful data is stored.

- Control traffic and user traffic: The service needs to be supported by both the control traffic session and the user traffic session.

For traditional network functions of Telco systems, there have been attempts to resolve the problems of upgrading incompatible software between old and new software versions with minimum or no service disruption. Figure 69 shows the main steps of one example of software upgrade solutions. The first step of the software upgrade procedure is to split the system into two sides. All traffic will be processed in only one of the two sides and all remaining traffic on the other side will be migrated to the side. The new software version will be upgraded on the side that is no longer processing traffic. Since the traffic migration is done inside the same version, there is no version compatibility issue. The second step of the software upgrade procedure is to isolate the side to be upgraded and proceed with the software upgrade operation. After software is upgraded, the common data (e.g. configuration, networking parameters) needs to be converted and configured in the new software before moving on to the next step. The third step of the software upgrade procedure is to convert the user related stateful data from the format of the old version software into that of the new version software and to transfer the converted data to the new version software. After the data transfer, a switchover is done from the side running the old version to the side running the new version. The fourth step of the software upgrade procedure is to update the software on the remaining side with the old software version. This fourth step is quite similar to step 2. The final step (not shown in Figure 69) of software upgrade procedure is to merge the two subsystems into one system and the final result is similar to step 2 in Figure 69, except that the network function has been upgraded to the new version.

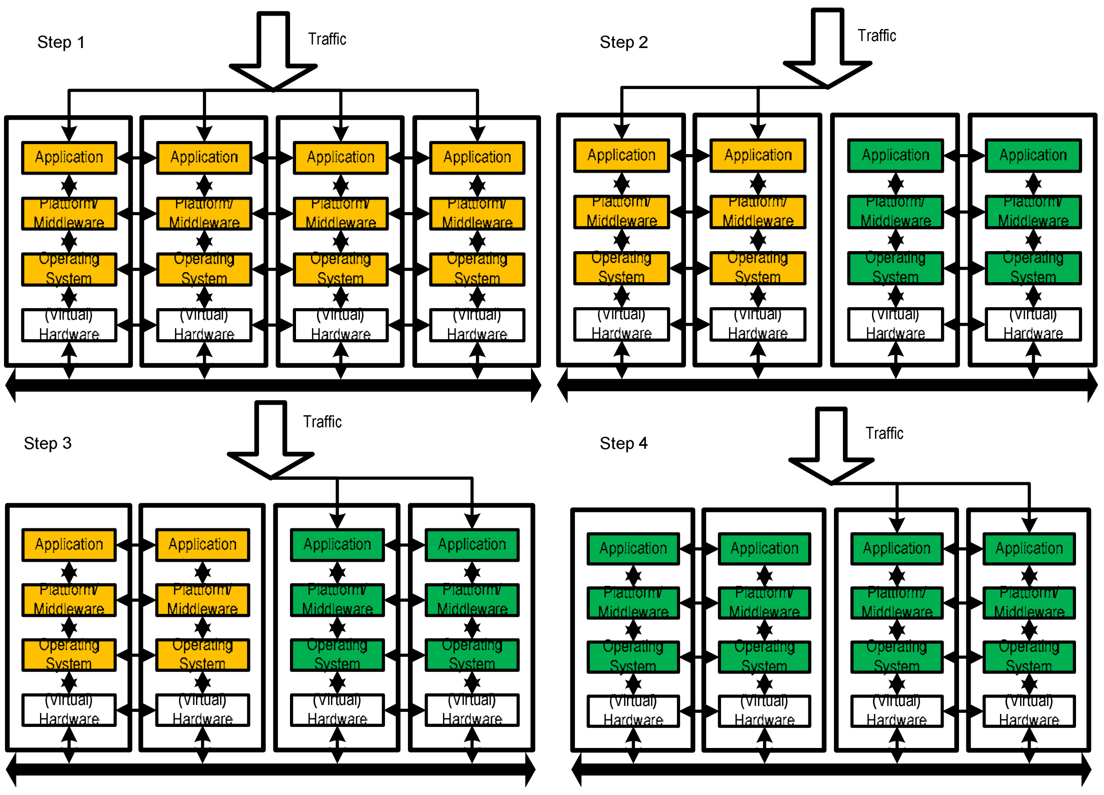


Figure 69: Main steps of software upgrade procedures (without redundancy) in a traditional network function

If the system has 2N redundancy for every service component, the standby part is updated first, followed by the update of the active part. The update procedure is quite similar to that of step 2 and step 3 in the previous (i.e. without redundancy) example.

The crucial operation of the software upgrade procedure in traditional systems is the data conversion and the switch-over between the two subsystems in step 3. Such a high volume of dynamical data conversion tends to be tedious and requires an exact description of the differences between the software versions. The conversion process is also known to be error-prone. For example, the high volume of dynamic data which needs to be converted in a typical network function is likely to bring practical difficulty practically and might result in some unintended service disruptions or outage.

In addition, each component of the system might not be always implemented with 2N redundancy, thus the serving capacity during the upgrade might be reduced to roughly half. If the system is operated with 2N redundancy, the system reliability will be decreased during the time of software upgrade because of the absence of failure protection. Such drawbacks will limit the flexibility of scheduling of software upgrade, so that an upgrade needs to be carried out only at off-peak periods to avoid overloading the system and to decrease the time for the data conversion process.

Furthermore, if the new version of software does not work properly after the switchover action in step 3, the failure recovery action is to roll back to the old software version and restart the system, which will in turn cause service interruption during the restarting period. Scheduling of software upgrade at night time could reduce the outage impact. Outages resulting from software upgrade, either in traditional systems or in cloud environments, always contribute to a big fraction of all system outages [i.3] and [i.4].

8.2.1.2 VNF software upgrade in an NFV environment

The software upgrade of IT clouds is typically using the "shutting down-upgrade" solution which is not preferred in an NFV environment. Indeed, in such environment, the challenges of VNF software upgrade are the same as those in traditional systems; however, elastic resources are available for providing new opportunities for the software upgrade process. The traditional software upgrade method of rolling upgrade could be improved with this new technology. With elastic resources, the VNF software upgrade could be done in a scalable way, e.g. upgrading a fraction of the whole capacity, a certain service type or a certain user group, with the constraint of preserving the service availability and service continuity. Thus the outages from VNF software upgrade can be reduced. For such an objective, the following principles should be followed by VNF software upgrade in an NFV environment:

1) Ongoing services/sessions/bearers are not disrupted and the new service access requests will always be served during software upgrade. The full capacity should be served and the system high availability should not be compromised during the upgrade.

2) Software upgrade can be done at any time during the day.

3) The verification of the new software version and the new configuration should be carried out with real traffic, while the old software version is still serving traffic with full capacity at the start of upgrade.

4) Software upgrade can be done in a scalable way, e.g. upgrade a fraction of the whole capacity, a certain service type or a certain user group, with the constraint of preserving the service availability and service continuity.

5) The rollback process can allow for a fast recovery without causing outage in the event of failure in the new software version and/or new configuration.

The following high level procedure (Figure 70) for software upgrade could be considered in the implementation of VNF software upgrade:

- In the first step, one or more instances of the old software version implementing the network function are providing services, while one or more instances of the new software version with limited capacity are instantiated for preparing to support the service.

- In the second step, some new service requests are routed to the new software version, while the existing service is still being supported by the old software version. Thus, the new version of software or new configuration could be verified with real traffic. User and traffic affinity should be taken into account when transferring the new service requests to the new software version. Some service requests might need to be decoded at the service layer protocol layer.

- In the third step, the serving capacity of the new software version is increased, e.g. by routing more service requests to the new software. Adding additional capacity or more instances in that part of the new software version with scaling up/out procedure might be needed. As the serving capacity of the old software version is decreased because of less new service requests and some users getting into idle, scaling in/down is executed to remove instances or decreasing capacity in the old software part. The serving capacity of the old software version could be successively decreased, while the serving capacity of the new software could be successively increased, until the fourth step.

- In the fourth step, when there is no, or very little, traffic left in the old software version instance(s), the service is fully switched into the new version of software system.

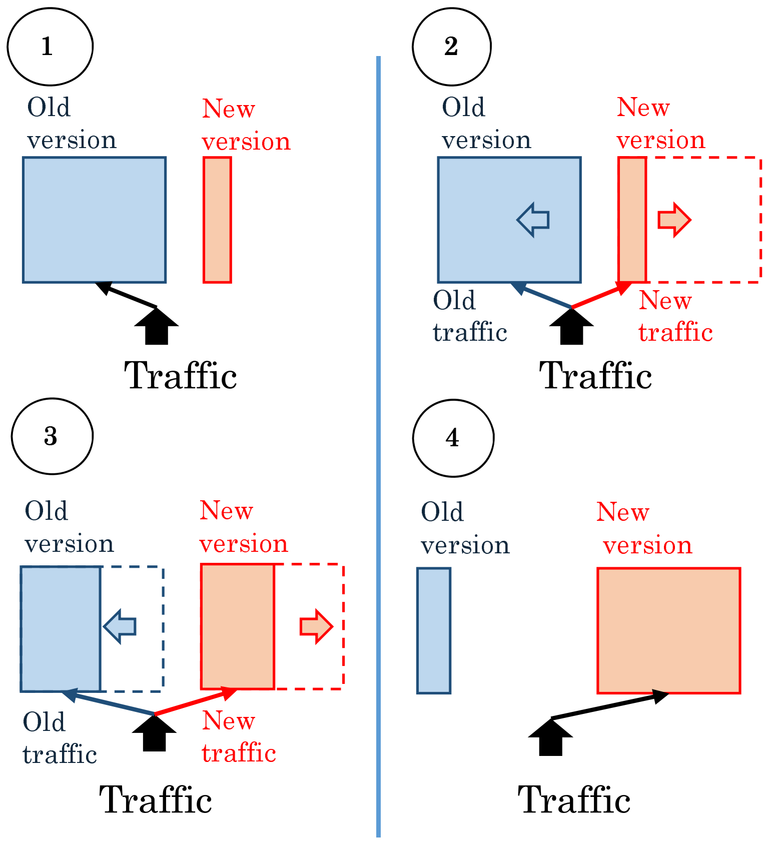


Figure 70: High level view of VNF software upgrade procedure

The similar principle has been applied in dynamic scaling with migration avoidance ETSI GS NFV-REL 002 [i.8]. Note that when moving from the third step to the fourth step, the (successive) reduction of instances/capacity running in the old software version might involve the migration of some users from the old software version instances to the new software version instances continually so that the old software instances can be removed. It could be possible to transfer the users' registration information without having any data conversion with service protocols, e.g. detaching and re-attaching or re-registration. Since there are multiple protocol layers in the Telco traffic, the routing of new service requests to the new software version in steps 2 and 3 will need the presence of a traffic decider (Figure 71) and some rules (e.g. upgrade for a fraction of the whole capacity, a certain service type or a certain user group, etc.) are needed in the traffic decider for controlling the upgrade procedure.

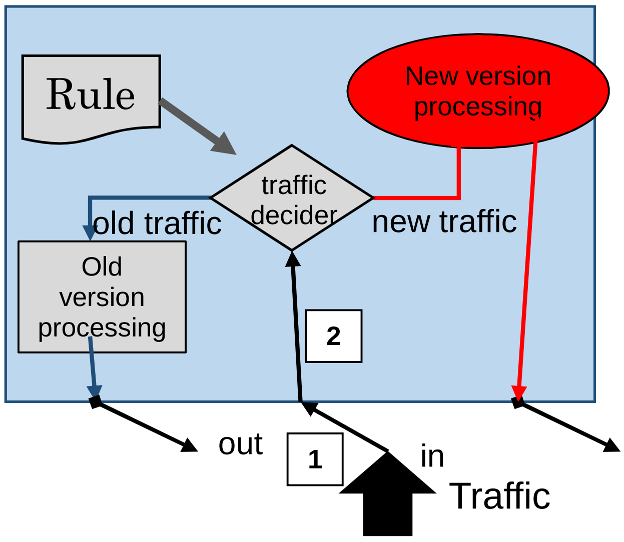


Figure 71: A diagram of traffic routing during software upgrade

The instances of the old software version and the new software version could be VNF instances or VNFC instances depending on the implementation. Since the software upgrade for a VNF should be seamless to other components of a NS chain, the traffic has to be delivered to the old version instance(s) in which the traffic decider is located. Indeed, e.g. for rollback option, this solution is more resilient than the one where the traffic decider resides in the new version instance(s). The third option of having the traffic decider outside the instance(s) is discussed in the clause concerning "software upgrade method using migration avoidance". For some Telco services, the control traffic normally initiates and controls the user plane traffic. Routing new service requests of control traffic to the instances of the new software version could indicate that the user traffic of the new established services will be processed in the instances of the new software version.

8.2.1.3 Software upgrade method using migration avoidance

The lifecycle of VNF software is expected to include:

- Updates: minor fixes in existing software such as correction of newly discovered "bugs". This results in a "point" update - for example, version X.0 is updated with version X.1.
- Upgrades: major revision of existing software such as new version of an operating system. This results in a "number" change - for example, version 0.x is upgraded to version 1.0.

The migration avoidance method defined in ETSI GS NFV-REL 002 [i.8] can be adapted for software updates/upgrades. The original intent of this method was to dynamically scale-out the number of VNFs under conditions of increased traffic load where utilization levels of existing VNFs became too high. Under such conditions, the migration avoidance method instantiates new VNFs in order to manage the increase in traffic processing in an acceptable manner.

Migration avoidance can be adapted for the VNF update/upgrade process. The architecture used for the update/upgrade process is described in ETSI GS NFV-REL 002 [i.8] -Figure 72. This architecture is illustrative - any other architecture can be considered for this purpose. It comprises the following components:

1) Servers (S1 and S2 in Figure 72) over which VNFs are instantiated - VNF updates/upgrades are instantiated by replacing existing ones over these servers. In Figure 72, VNF *A* on server S1 is replaced with an update/upgrade instance *A'* which is instantiated on server S2.
2) Software switches are run on each server; the VNFs are run over these switches.
3) A hardware switch connects the servers together and it directs flows to the servers based on which VNF the flows need to be processed by.
4) An external controller coordinates and manages the entire process.

The following assumptions apply for this process:

1) The new VNF software update/upgrade is backward compatible with the existing VNF.
2) The update/upgrade process is carried out during a period where incoming traffic loads are not at peak values. This permits some testing of the newly instantiated updated/upgraded VNF - if errors are observed with this VNF, it can be removed and traffic processing can continue with the old VNF.
3) The updated/upgraded VNF version can be tested for satisfactory performance using the range concept introduced for migration avoidance in ETSI GS NFV-REL 002 [i.8]. The assumptions for this concept can be adapted for VNF updates/upgrades as follows:
    a) Each traffic flow can be mapped into a flow ID (e.g. via a hash function applied to relevant header fields) in a specified interval *R*.
    b) The existing VNF and the update/upgrade VNF instance are associated with one or more sub-intervals (or range) within *R*. A range is described as a subset of the original interval *R*; note that ranges are non-overlapping.
    c) The hardware switch can compute the flow ID and determine which range within R it belongs to using a manageable set of rules.

d) The software switch on each server tracks the active flows that a VNF instance on that server handles. The notation $F_{old}(A)$ is used to refer to the set of flows handled by the existing VNF instance *A* prior to the instantiation of the update/upgrade *A'*.

e) The existing VNF instance *A* will have a range filter installed in the server's software switch or in the hardware switch. When updating/upgrading a VNF instance *A* with *A'*, the range previously associated with *A* is partitioned between *A* and *A'*. This partitioning allows a subset of new traffic flows to be tested over the newly instantiated VNF update/upgrade *A'*, while allowing the remaining flows to continue over the existing VNF *A*. This test period can be run until the performance of *A'* is confirmed as satisfactory; after that, all new flows can be directed over the update/upgrade VNF *A'* and the existing VNF instance *A* can be removed once all existing flows over *A* have completed. For example, if the original range for *A* is [X, Y], then the partitioning into two new ranges is done without overlap, such as:

▪ Range for *A* prior to split:	[X, Y]
▪ Range for *A* after split:	[X, M) - this notation indicates that time instance M is not part of the range for *A*
▪ Range for *A'* after split:	[M, Y] - this notation indicates that time instance M is part of the range for *A*

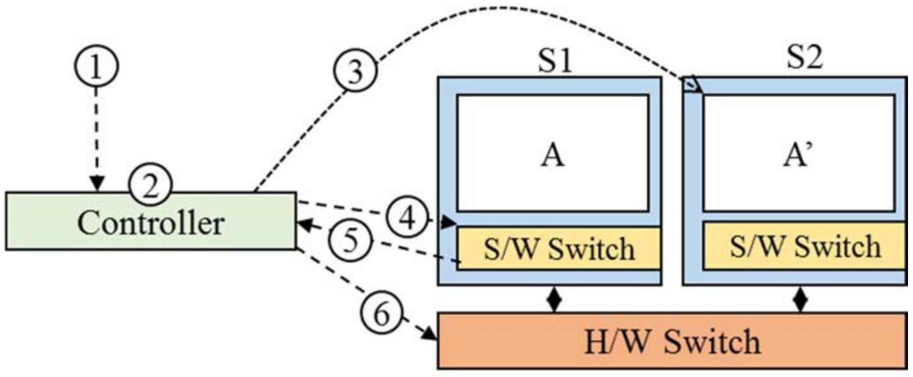


Figure 72: Migration avoidance process for VNF update/upgrade

A step-by-step process for the update/upgrade is as follows:

1) The controller receives external notification that VNF *A* needs to be replaced with an update or upgrade.

2) The controller determines where the update/upgrade VNF instance should be placed. In the example in Figure 72, the controller chooses server S2.

3) The controller instantiates the update/upgrade VNF instance *A'* at server S2.

4) As introduced above, [X,Y] denotes the range for *A* prior to the update/upgrade and [X,M) and [M,Y] the ranges for *A* and *A'* after the update/upgrade; similarly, $F_{old}(A)$ denotes the flows that were active at *A* prior to the update/upgrade. The controller's next step is to configure the software switch at S1 to send packets from *new* flows that fall in the range [M,Y] to the server S2 while continuing to send packets from flows in $F_{old}(A)$ and *new* flows that fall in the range [X,M) to VNF instance *A* at S1.

5) As flows in $F_{old}(A)$ gradually terminate, their corresponding rules are removed from the software switch at S1. Once the number of active flows in $F_{old}(A)$ that belong to the range [M,Y] drops below a configured threshold value *T*, the controller is notified.

6) The controller now configures the hardware switch to achieve the following:

a) Packets from active flows in $F_{old}(A)$ that fall in the range [M,Y] are sent to S1 (as per the above, there will be fewer than *T* of these).

b) Packets from flows that fall in the range [X,M) are sent to S1.

c) Packets from flows not in $F_{old}(A)$ that fall in the range [M,Y] are sent to S2.

d) The value of M can be selected to permit representative testing of flow processing on the update/upgrade VNF *A'*. For example, range values of [X,M) = [0,80) and [M,Y] = [80,100] allows for 20 % of new flows to be processed at *A'*, while the majority of new flows (80 %) continue to be processed at *A*. Once it is determined that *A'* processing is satisfactory, the range values can be adjusted by setting the value of M to X at some designated time value *T'* with the following implications:

   - All existing flows at *A* at time *T'* will continue to be processed by *A* until they terminate.
   - ***All new flows arriving after time T' will be directed to A'.***
   - Once all existing flows at *A* terminate, VNF *A* can be removed.

To accomplish this final action, steps 4 - 6 from the process above are thus repeated by setting the value of M to X at time *T'* resulting in:

- Range for *A'* at time *T'*: [X, Y]

It is understood that state related to a potentially large number of flows (e.g. PDP contexts) might need to be migrated over to the updated VNF resulting in a potential burst of signalling traffic in the network. Methods for minimizing such situations are for further study.

8.2.2 NFVI software upgrade

In cases in which the hardware abstraction layer is implemented by both the hypervisor and the guest OS, the guest OS might have a dependency on the hypervisor or hardware ETSI GS NFV-INF 003 [i.31]. In an NFVI environment, there are software components such as the host and guest operation system (OS) including their drivers, virtualisation layer (including the hypervisor), hardware related firmware and device drivers, networking components for virtual links and software for virtual storage, etc. Each of those components will need to be updated or upgraded to fix bugs and/or to add new features. The recommendations (see clause 8.1) for the software update process from the service availability and continuity's point of view are also applicable to NFVI software upgrade. The generic principles of NFVI software upgrade will be discussed in the following scenarios.

Scenario 1: Upgrade software of firmware and/or device driver of NFVI hardware (including the whole physical hardware)

This scenario deals with the upgrade of the server hardware-related software components or the hardware itself (Figure 73). The upgrade could be done by using the "live migration" of VMs to maintain the service availability and service continuity. For example, the VIM creates the same type of VMs (the same VM configuration and supporting files) in those servers for which the hardware has been upgraded (considering that the virtualisation layer software of those servers is compatible). Then "live migration" can be done between those VMs sequentially. It should be noted that the successful "VM live migration" is only done between VMs with compatible hardware, e.g. with compatible CPU model, communication bus and devices. The "live migration" for a VM configuration with hypervisor pass-through, e.g. VM directly connected with SR-IOV devices, is a quite challenging operation and might require PCI to be unplugged and re-plugged. Since the same network connectivity logic (e.g. IP address) should be maintained for those migrated VMs to ensure service continuity, the upgrade might require the migrated VMs to be instantiated in the same sub-network.

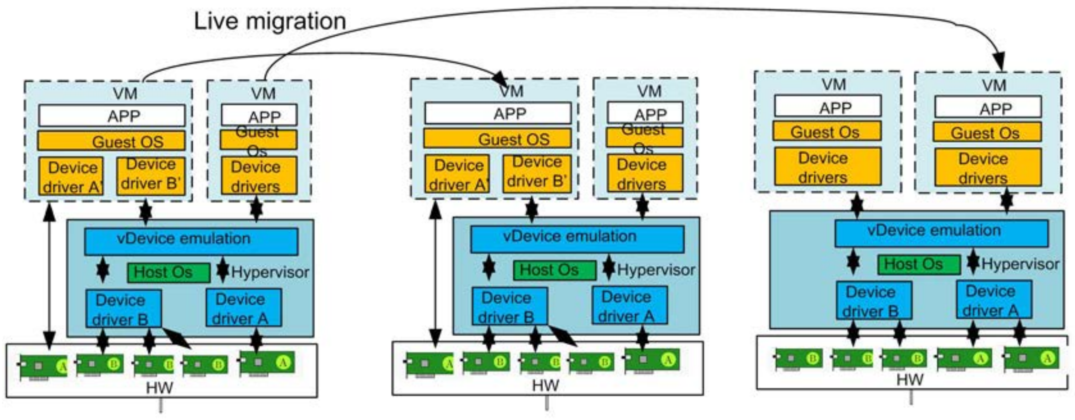


Figure 73: Diagram of main software components in physical servers

Scenario 2: Upgrade of guest OS

The guest OS may be installed separately from the virtualisation layer although there may be some dependency between them. The VNF software might also have some dependency on the guest OS. Any dependencies will have to be managed. The update or patching (minor release) of the guest OS could be done without impact on the service continuity; however, a subsequent VM restart is normally needed even with backward compatibility support in the new version of the guest OS. The upgrade of the guest OS may, or may not, require a corresponding VNF software upgrade. If the guest OS upgrade does not require a corresponding VNF software upgrade and there is backward compatibility supported by the guest OS. VM live migration could be applied. Scenario 3 is required for incompatible guest OS upgrade in order to keep service continuity. If the VNF software is required for the guest OS upgrade, the methods presented in clause 8.2.1 could be used for updating VNF software and the guest OS software simultaneously.

Both the bare-metal virtualisation layer and the hosted virtualisation layer have very high dependency on the host OS. It might be possible to patch the host OS without impact on the service of the virtualisation layer; however, the upgrade of host OS will have an impact on the virtualisation layer. For simplification, the host OS should be maintained together with the virtualisation layer as discussed in scenario 3 and its upgrade will not be considered as a separate scenario.

Scenario 3: Upgrade of virtualisation layer

In this scenario, the host OS is considered to be delivered and maintained together with the virtualisation layer software. The upgrade of the virtualisation software needs to consider cases of the software with and without backward compatibility. The backward compatible virtualisation layer software assumes that both the virtualisation layer and host OS software support the backward compatibility. If the virtualisation layer software is backward compatible, the "VM live migration" described in scenario 1 could be applied to the software update procedure. The incompatibility of the virtualisation layer software mainly arises due to the difference in the type of virtualisation layer and/or host OS. There are many kinds of software available for VM image conversion in order to instantiate the same type of VM in the new virtualisation environment; however, the VM image conversion can be used only for "cold VM migration" which is not suitable for the software upgrade of carrier grade service and could cause service discontinuity and service unavailability. Currently, there is no "VM live migration" solution in the cases of virtualisation layer software incompatibility. In order to maintain the service availability and continuity, "service migration" might require service data synchronization between VNFs on old and new releases. The corresponding signalling for service data synchronization at the NFVI management plane is needed for orchestration. Another alternative discussed in clause 8.2.1 could be applied for "service migration" to support the upgrade of incompatible virtualisation layer software. New instances are instantiated for the new release to support new traffic while the old release instances are still carrying old traffic, and a slow migration of traffic to the new release instances is launched. More detail is provided in clause 8.2.1.

8.3 Summary for software upgrade in NFV

The software upgrade recommendations for service availability and continuity in the NFV environment have been highlighted in this clause. According to available network outage statistical reports, the software or network upgrade is one of the main causes of outage. The traditional software upgrade method of rolling upgrade with data conversion seems not to be very efficient to prevent the outage caused by software upgrade. In this clause, the software upgrade methods based on simultaneously instantiating instances of old and new releases for the same network service and smoothly transitioning or migrating of services from the old release into new release have been proposed to VNF software upgrade. A few scenarios of NFVI software upgrade have been analysed and the corresponding high level principles for software upgrade have been presented. With those proposed methods or principles, the outage resulting from software upgrade could be minimized and the service availability and continuity could be maintained. However, some issues of NFV software upgrade have not been studied such as:

1) The upgrade of NFV-MANO software has not been investigated. The availability and reliability of NFV-MANO are quite important to maintain the service availability and service continuity.

2) Even though some methods or principles have been proposed for VNF and NFVI software upgrade, however, the software upgrade orchestration procedures and messaging flows have not been specified yet.

Further studies for those issues are needed.

9 E2E operation and management of service availability and reliability

9.1 Service flows in a network service

According to ETSI GS NFV 002 [i.1], "an end-to-end network service (e.g. mobile voice/data, Internet access, virtual private network) can be described by a NF Forwarding Graph of interconnected Network Functions (NFs) and end points". Such network functions might be virtualised network functions (VNFs) or physical network functions (PNFs). Since resource management of PNFs is not in the scope of NFV, this clause only considers NFV environments in which all NFs of an end-to-end network service are virtualised (Figure 74). The network service chaining in NFV is the instantiation of a Network Service (NS) through a VNF Forwarding Graph (VNFFG) which includes the instantiation of VNFs and the creation of virtual links (VLs) for network connections between VNFs. The description of network forwarding path (NFP) and VLs is used to specify the service requirements for steering traffic through VNFFGs. One network service chaining could support one or multiple service flows specified with service parameters such as QoS, co-location, physical separation, regulation constraints, service reliability and availability, security, etc.

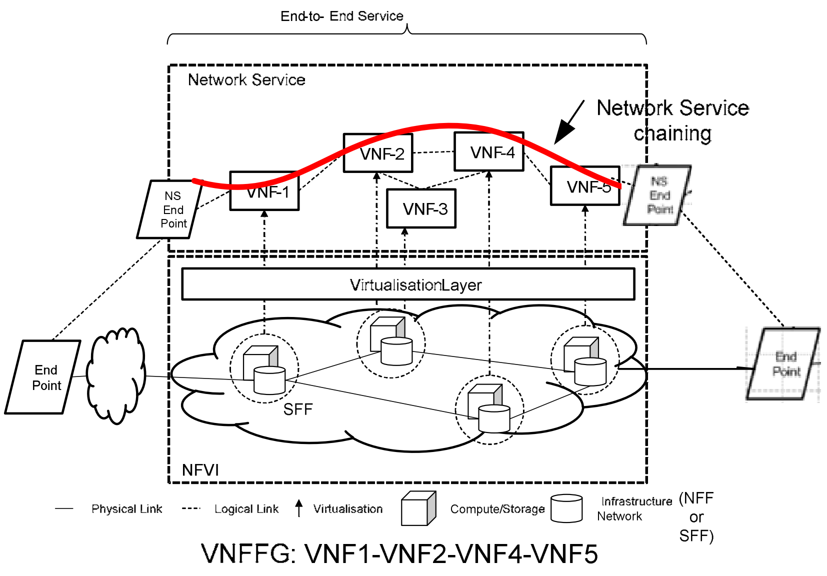


Figure 74: A network service chaining on VNFFG

Insider a VNF**,** the internal service graph consists of a few entities, e.g. resource entities (such as virtual containers) and internal virtual link entities, such as vSwitches, and physical networking components. Each virtual container might support multiple processes or software components. As an extension of network service chaining into VNF, the internal service chaining (one or multiple service flows) will therefore be supported by networking, hardware, hypervisor, virtual machine (VM), and service processes (Figure 75).

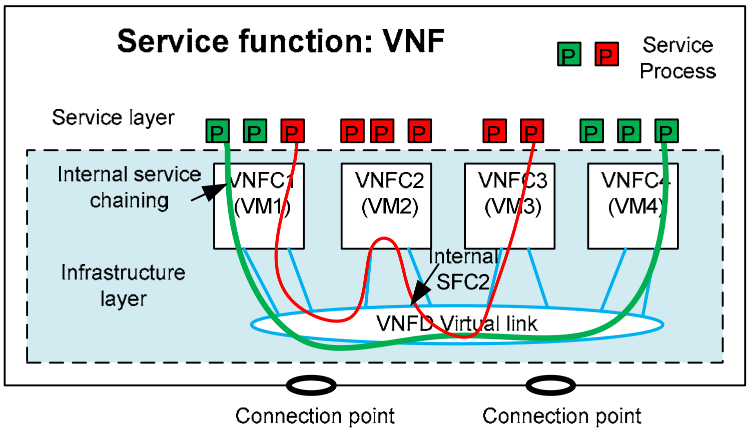


Figure 75: VNF internal service chaining

From the service availability and reliability's point of view, the service resilience requirements of a NS are applicable to both the external and the internal service chaining.

For traditional networks, service function deployment models are relatively static and are tightly coupled to network topology and physical resources. The static nature of deployments greatly reduces or limits the ability of a service provider to introduce new services and/or service functions, since any service deployment change might affect the reliability and availability of other services. The design of NS availability and reliability is traditionally done hop by hop and the service of each hop is configured and operated independently. The hardware and software are shipped to a service provider after reliability and availability deployment has been configured in the equipment. Once their shipment, the service provider is nearly unable to change the reliability and availability configuration. There is no mechanism available for managing the end-to-end service reliability and availability.

Another limitation of traditional Telco systems is that all applications or services are normally designed with the same level of service availability and reliability without any differentiation. The need for availability and reliability of each service has not been considered in the implementation. The Web browsing service, for example, could tolerate 30 seconds or more of service interruption without impacting the user experience. However, current systems only provide one level of resilience, typically with a failure recovery time of a few seconds. Thus, over-provisioned system resources have been assigned to a service requiring lower levels of service availability and reliability.

One of the NFV key design objectives is the end-to-end availability of telecommunication services ETSI GS NFV-REL 001 [i.2]. According to deliverables ETSI GS NFV 002 [i.1] and ETSI GS NFV-REL 001 [i.2], NFV frameworks is such that not all services need to be "built to the peak", but Service Level Agreements (SLAs) can be defined and applied according to given resiliency classes.

9.2 Metadata of service availability and reliability for NFV descriptors

9.2.1 Introduction

NFV has potential benefits such as reduced Capex, risk reduction, faster time to market, flexibility and optimal resource utilization, etc. Since Network Service (NS) deployment in NFV environments is more dynamic than in traditional networks, the flexibility, visibility and automation for deploying NS should be one of the essential properties of an NFV framework. When new services need to be on-boarded in the existing NFVI network, the network administrator might need to consider various sets of network configuration policies to be deployed in order to have the corresponding NSD based on the SLAs required by the customer. If the policy encompasses various parameters such as quality and performance, utilization, capacity, security, availability, provisioning and customer-specific policies, it would be a very complex process. The NFV framework should enable SLA parameters to be commissionable as much as possible for NS deployment from the automation, service assurance and service operation transparency's point of view. For example, since the service providers understand what kind of service availability and reliability is required for the deployed services (service types or group), the NFV framework should be able to provide mechanism for deploying end-to-end service availability and reliability based on the SLA resilience parameters.

A network service is defined in ETSI GS NFV-MAN 001 [i.5] by descriptors, such as NS Descriptor (NSD), VNFFG descriptor (VNFFGD), VNF descriptor (VNFD), PNF descriptor (PNFD), VL Descriptor (VLD), etc., while the NFP of a VNFFGD is used to describe the connection endpoints for VNF Forwarding Graph (VNFFG). The virtual deployment unit (VDU) of VNFD is used to describe the VNF components (VNFCs). The current NFV framework neither supports end-to-end management and operation of service availability and reliability, nor provides any mechanism for service availability and reliability differentiation. NFV descriptors have to be extended to enable end-to-end management of service availability and reliability with differentiation to services (types or groups) to fulfil the NFV resilience requirements (ETSI GS NFV 002 [i.1] and ETSI GS NFV-REL 001 [i.2]).

As shown in Figure 74, the main components of a NS include VNFs, virtual links and network forwarding paths. The service availability and reliability requirements should be included in the NFV descriptors to allow for automatic deployment, while the relevant defined parameters for those requirements should enable end-to-end service availability and reliability with service differentiation. To achieve this goal, the following issues have to be considered in the design:

- VNFFG comprises various equipment and entities produced by multiple vendors, who have applied different proprietary solutions.
- A VNF should be supported by multiple vendors' layered software and hardware, i.e. VNF software, virtualisation layer, hardware, virtual network, etc.
- NFV entities, e.g. NFVO, VNFM, VIM and NFVI, are not able to obtain the service related information. It is required to have a mechanism for delivering service availability and reliability requirements to those entities that manage the virtual resources which are provided to a NS.

It would be an advantage to have a mechanism for assigning service availability and reliability metadata which may be used for instantiating the end-to-end service availability and reliability with service flow differentiation for a network service. A mechanism for delivering end-to-end service availability and reliability metadata in the traffic data is also needed. Thus each network function or service function such as VNF and Service Function Forwarding (SFF) entities of the network may be able to handle traffic with the same predefined policy during anomaly situations, e.g. failure, or overload.

9.2.2 Metadata of service availability and reliability

Service availability level is the abstraction of service availability and reliability requirements. Three levels of service availability (1 to 3) are defined in ETSI GS NFV-REL 001 [i.2]. Each Service Availability Level (SAL) defines the (same) level for the following elements:

- Priority for service access.
- Priority for congestion control.
- Priority for failure recovery.

- Level of (Service) failure recovery time.
- Level of failure probability.
- Level of failure impact.

For example, service availability level 1 indicates:

- Service access: priority 1 (highest) for service (or virtual resource) access.
- Congestion control: priority 1 (highest) for congestion control (i.e. low probability of being subjected to congestion control).
- Failure recovery: priority 1 (highest) for failure recovery.
- Failure recovery time: level 1 of service (failure) recovery time.
- Failure probability: level 1 (lowest).
- Failure impact: level 1 (lowest).

ETSI GS NFV-REL 001 [i.2] recommends that the level 1, level 2 and level 3 of failure recovery time are in the range of 5 - 6 seconds, 10 - 15 seconds and 20 - 25 seconds.

It needs to be aware that one of the SAL elements, the failure impact, is used to describe system availability. For the deployment of VNFC (within VNF) and NS (e.g. VNFFG and NFP), the failure impact element needs to be considered in capacity planning (e.g. load sharing), and anti-affinity rules. However, if SAL is used to define individual virtual resources, the failure impact is not a valid requirement.

From the service availability and reliability's point of view, a single service flow of a NS has its own availability and reliability requirements. Since SAL has the measurable elements of availability and reliability, the metadata of service availability and reliability of a service flow could map into SAL during service commissioning. SAL could be used to abstract the service availability and reliability requirements of service flows. More specifically, the service availability level could be used to summarise the resilience requirements of a certain service type (e.g. voice call, gaming, etc.), or certain user group (e.g. VIP, closed user group, tenant group, etc.) during service commissioning for simplifying the service deployment. By a mapping of the service availability and reliability requirements of a service flow into the corresponding SAL (by the service provider), while the entities of resources management (i.e. NFVO, VNFM and VIM) only handle or interpret SAL into requirements for virtual resources, the deployment model of service availability and reliability should be much simplified.

Whenever a virtual resource (computing, storage and network) needs to be provided for a certain service flow, the resource orchestrator, e.g. NFVO, could deliver the parameter of SAL to the entities of the virtualised infrastructure manager (VIM). VIM would assign the resources, based on the capability of hardware and software in the NFVI, which are potentially able to fulfil the requirements specified in SAL. During runtime, the VIM might need to properly observe the system for service assurance.

9.2.3 Service availability level in NFV descriptors

Due to the virtualisation, neither NFV-MANO nor NFVI have the knowledge what type of service is running on top of the infrastructure. The service availability and reliability could not be represented with one or two parameters. In order to provide service availability and reliability differentiation in NFV, it is required that the service availability and reliability metadata, e.g. service availability level, are represented in some descriptors for providing the requirement to NFV-MANO and NFVI. With SAL, NFV-MANO has the possibility to acquire the properly resources for a service.

To reach end-to-end service availability and reliability, there is a need to manage and operate service availability and reliability from the context of "VNF", "virtual link" and "network forwarding path". Service availability level represented in VNF or VDU, VLD and NFP could be a solution for end-to-end service availability and reliability with differentiation to service flows.

SAL in VDU

A VNF might consist of one of more VNFC whose virtual resources are described in the VDU. Service availability level including the VDU information element could be used to specify service availability and reliability requirements for the virtual resources requested by a VNFC. Thus, NFV-MANO is able to derive the service availability and reliability requirements for the virtualisation containers during network service instantiation. When the NFVO, or the VNFM, requests the creation of virtual resources for a VNF, it will deliver the SAL parameters in the creation message to the VIM. The VIM is thus able to assign the virtual resources (computing, storage and network) which possess certain service availability and reliability attributions (e.g. hardware and software attributions) to fulfil the service needs. The VIM might acquire the NFVI to activate some resilience mechanism for fulfilling the service requirements. For example, a VM failure obviously needs to be recovered. The SAL then provides the abstraction of service availability and reliability requirements for the failure recovery, e.g. failure recovery time. How the failure is recovered and how the virtual resources are provided to respect the service availability and reliability attributions are an implementation issue.

SAL in NFP and VLD

The Network Forwarding Path (NFP) may comprise an ordered list of connection points and rule related information which may enable the NFVI or network controllers to configure accordingly the forwarding tables. Multiple service flows of a network service may be handled by the same NFP or VL with VLAN, VXLAN, MPLS, IP, etc. and the service flows could be identified by the traditional transport layer identifier, e.g. VLAN tag ID, or VXLAN tag ID, or MPLS tag ID, or IP address, while the priority of a service flow may use the Class of Service (CoS), or Qquality of Service (QoS). However, the CoS or QoS of a service flow is unable to provide information for the failure recovery because the tolerable time of a service interruption cannot be derived from CoS or QoS.

Currently there is no end-to-end resilience requirements defined in NSDs. Without any service availability and reliability requirement for the virtual links and network forwarding paths of a VNFFG, the corresponding network service might not be able to achieve the object of end-to-end service availability and reliability. In current resilience technologies, the recovery of a link or path failure varies from 40 - 120 seconds (e.g. OSPF) to 50 ms (e.g. MPLS fast rerouting). The failure recovery for a path or a link needs to be planned during service deployment so that the infrastructure or network controller can act accordingly in case of failure. NFP and VLD should include the service availability level in the information element for end-to-end deployment of service availability and reliability. In addition, a mechanism to deliver the parameter of service availability level in the network traffic might be needed and how the SAL represented in virtual links depends on the link format. The reuse of CoS or QoS to deliver SAL is a possible option.

Since there might be multiple service flows transported in the same data path and virtual link, service availability level in NFP and VLD should include:

- service availability levels of service flows in NFP or VL;
- policy for identifying service flows (e.g. VLAN tag ID, VXLAN tag ID, MPLS tag ID, IP address) and the corresponding mapping between CoS or QoS and service availability level.

The service availability level including NFP and/or VLD information element is used to provide the service resilience requirements for each service flow in NFP and/or VL to NFV-MANO, so that the NFVI or the network controller can handle flow traffic with service differentiation during anomaly situations (e.g. failure, or overload).

9.3 End-to-end operation and management of service availability and reliability

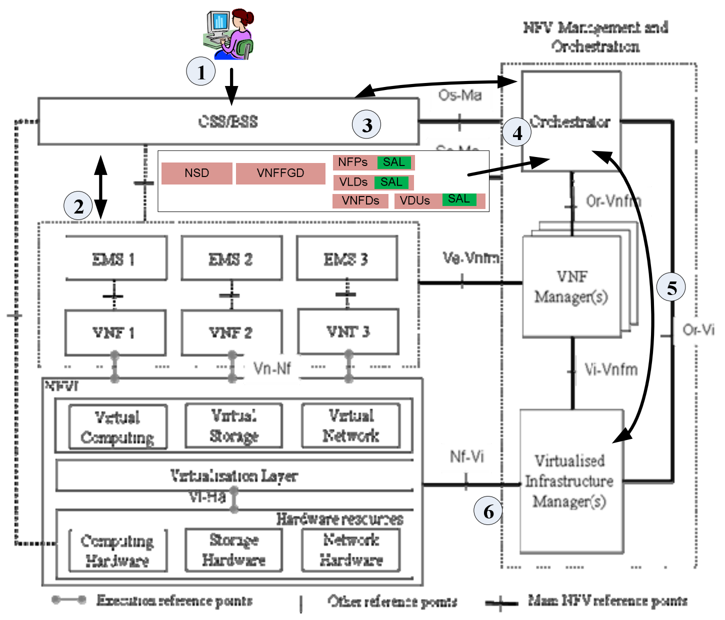


Figure 76: Operation and management procedures of service availability and reliability

The service availability level represents an abstraction of service availability and reliability requirements for the associated virtual resources. The introduction of service availability level in the descriptor for the VDU, NFP and VLD provides the possibility for end-to-end service availability operation and management with service differentiation by taking the information into account during deployment. A possible set of procedures for service availability deployment for a NS are highlighted in the following (Figure 76):

1) The service provider's administrator commissions each service flow (service type or service group) to a certain service availability level per requirement.

2) The administrator selects VNFD for each VNF if the service availability level has been defined in VDUs for each VNFD according to the service commission in step 1. Otherwise, the OSS needs request the corresponding VNFD from the operation and management system (e.g. EMS) responsible for each VNF.

3) When the OSS has received all VNFDs for the VNFs included in the VNFFG and the corresponding software image(s), the administrator commissions the NS by configuring NSD and VNFFGD. The service availability level(s) of the service flow(s) of a Network Service is (are) configured in NFP and/or VLD.

4) After NSD and other descriptors are configured and other NS operation and management procedures are done, the OSS delivers the software images and the NFV descriptors (NSD, VNFFGD, NFP, VLD, VNFD, VDU), which include the SAL parameters, to NFVO. When NFVO obtains those descriptors, it initiates the following steps for NS instantiation.

5) The NFVO establishes the virtual resources based on descriptors by requesting the virtual resource creation for the NS from the VIM. The virtual resource (computing, storage and network) creation requests include the corresponding service availability level which delivers the abstraction of the service availability and reliability requirements for the requested virtual resources.

6) The VIM will assign the virtual resources which have certain service availability and reliability attributes (e.g. hardware and software resilience related features, set of policies for anomaly handling) according to the service availability level in the resource assignment requests. The VIM might coordinate with the NFVI and the network controller for assigning the virtual resource to fulfil the service availability and reliability requirements quantified by SAL.

7) When virtual resources have been assigned to a NS, NFVO requests VNFM to instantiate each VNF instance. Thus, the service flows in the instantiated NS have been deployed with the commissioned service availability and reliability support.

In summary, the service availability level configured in NFV descriptors is based on the service availability commissioning in a NS. In response to a request for set up a VNFFG for a network service, the NFVO collects the service availability level from the descriptors for the VNFFG components - VNFs, NFPs, and VLs - that will comprise the VNFFG. The NFV descriptor data may be provided directly to the NFVO or collected by request from the NFVO to the OSS. The service availability level is used to deliver the service availability and reliability requirements for associated virtual resources and can be included in the NFVO request to VIM to assign and manage the appropriate resources.

During system runtime, the metrics for the service availability ETSI GS NFV-REL 001 [i.2] and service quality ETSI GS NFV-INF 010 [i.10] need to be collected and analysed. Fault management, enhanced by fault correlation and root cause analysis should be included in the NFV system operation and maintenance. System statistics and analytics should be able to collect availability-related data from each layer of the system architecture and analyse it automatically against the system availability requirements. A failure prediction and prevention system is needed and some proactive operations (e.g. scaling, migration, network service adjustment, etc.) might be executed if the service availability and reliability requirements quantified by SAL are potentially violated. The VIM could control or adjust resource usage (with predefined policies) according to the service availability level during virtual resource assignment and system runtime in case of congestion.

The main benefits of such a framework of service availability and reliability are:

- Enabling service providers to operate and manage end-to-end service availability and reliability.
- Enabling service availability and reliability differentiation; The availability and reliability can be configured to adapt to the need of each service flows in the NS commissioning and instantiation.
- Providing flexibility and visibility for service availability and reliability management.
- Providing efficiency in virtual and physical resource management.

10 Recommendations/Guidelines

The present document describes some models and methods for estimating the reliability/availability of end-to-end services in an NFV environment. In such an environment, there are several points which need to be considered for the estimation of reliability and availability compared to traditional networks composed exclusively of PNFs. By investigating reliability-related topics from the viewpoint of lifecycle management, including fault management, several conclusions are derived, though there still remains a lot of work that needs to be done to estimate reliability/availability more precisely. In this clause, the recommendations derived from the current state of the work are listed below.

- By partitioning the end-to-end network service into several service chains, the reliability and availability of an end-to-end network service can be estimated as a function of the reliability and availability of each service chain and the topological connection pattern of the chains. This implies that the investigation of each service chain produces input for estimating the overall network service reliability and availability.
- As VNFs can be issued from different vendors, their reliability is best provided by the software vendor to estimate the resilience of the NS using these VNFs. Some methods for evaluating software reliability are described in clause 5.3.
- Operational reliability/availability parameters can be measured after the deployment, but input for reliability estimation is needed before the deployment. Software and infrastructure reliability, as well as investigation of configurations can be used as input for such estimation.
- For the evaluation of reliability and availability, reliability block diagrams may be useful, when considering some components and their dependency relationship specific to NFV. Lifecycle operation and fault management cycle also need to be taken into account.
- In the fault management cycle, NFVI and NFV-MANO play important roles. These roles differ according to the chosen protection scheme. Besides fault management, scaling and software upgrade procedures are also closely related to NFV-MANO. To perform these procedures, further development of reliability supporting mechanisms in NFV-MANO is recommended to improve reliability and availability. Though mechanisms need to be defined to increase the reliability of NFV-MANO and NFVI, VNFs can have mechanisms by themselves to increase their reliability/availability.

- The estimated reliability of a network service varies according to the phase of the lifecycle operations. Some equations for evaluation during each of these operations are listed, though more precise estimation models throughout the lifecycle operation should be created in further studies.
- Software upgrade methods for VNFs and NFVI software aiming to increase the overall reliability are proposed. The principle of the upgrade mechanism is based on a smooth traffic transition from the old version instance(s) to the new version one(s). Fractions of the whole load are directed to the new version software in steps. With these methods, the software can be upgraded in a scalable manner and roll-back made easier, while preserving the service availability and service continuity.
- For networks that provide multiple services, the reliability and availability do not need to be "built to the peak" for all service flows. A method to realise the differentiation of service availability is presented.

11 Security Considerations

The phases of the lifecycle operations considered in the present document, which principally deals with the estimation of E2E reliability and availability of network services and the operation of resilient network services, include operation and management during steady state and software update/upgrade.

During these phases, sources of failures are multiple. They span from intrinsic VNF unreliability to physical infrastructure breakout, including the virtualisation layer instability. As the faults leading to failures are not explicitly defined in the present document, e.g. the reliability block diagram approach used to estimate the E2E reliability and availability of network services is agnostic to the source of unreliability of the building blocks it considers, these faults can originate from accidental causes, as well as from intentional ones.

Nevertheless, as the software (seen from a broad perspective, i.e. it includes the code composing VNFs, as well as the one inside NFVO, VNFM, or VIM) reliability is a main component of the overall E2E reliability and availability of network services, its modelling, as presented in clause 5.3.3, is generally based on unintentional faults. The treatment of security-related faults, e.g. trojans leading to software failures, is to be apprehended in further studies.

Furthermore, update/upgrade processes may be jeopardized by attacks, e.g. man-in-the-middle, compromising thus the E2E robustness targeted in the original design: supplementary work is needed to cover these aspects.

Finally, security threats such as DDoS may impact directly the NFV system availability if countermeasures mechanisms are non-existent or inefficient. Such considerations also need to be considered in further studies.

Annex A (informative): Reliability/availability methods in some ETSI NFV PoCs

A.1 Introduction

This annex provides a summary of ETSI NFV PoC related to reliability/availability methods and feedback to the present document.

A.2 PoC#12 Demonstration of multi-location, scalable, stateful Virtual Network Function

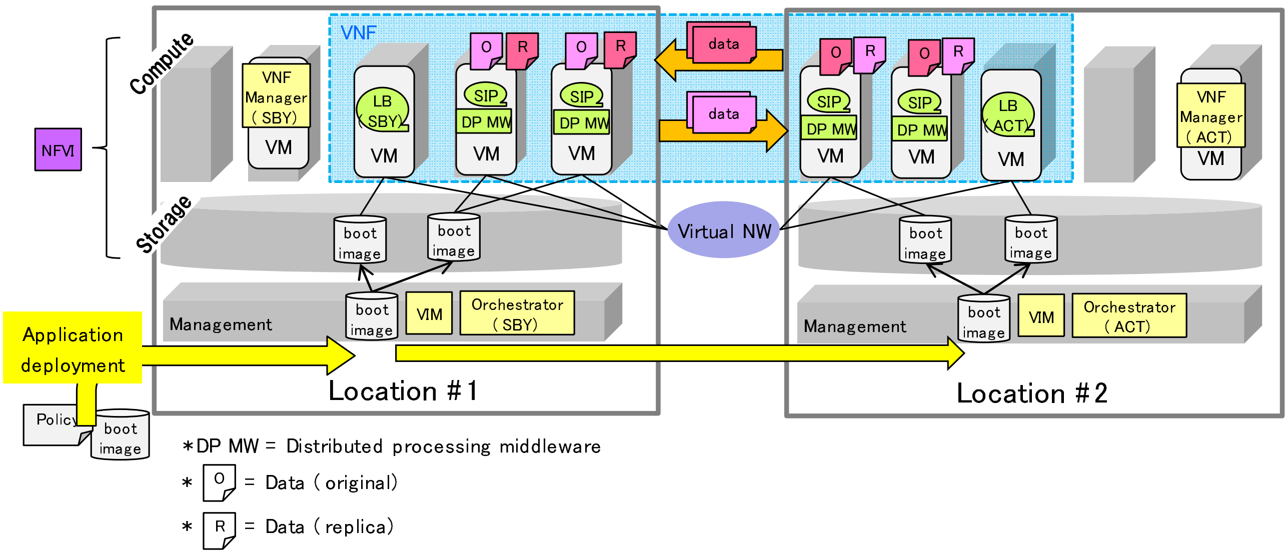


Figure A.1: PoC#12 - PoC configuration

PoC Team

NTT
Fujitsu
Alcatel-Lucent

PoC demo

Bankoku Shinryokan Resort MICE Facility VIP room (Okinawa, Japan) - 14 May 2014
ETSI NFV PoC ZONE @ SDN & OpenFlow World Congress, 15-17 October 2014, Dusseldorf, Germany

Overview

This PoC demonstrates the reliability and scaling out/in mechanisms of a stateful VNF that consists of several VNFCs running on top of multi-location NFVI, utilising purpose-built middleware to synchronize states across all VNFCs. As an example, a SIP proxy server is chosen as a VNF, which consists of a load balancer and distributed SIP servers with high-availability middleware as VNFCs.

Note that the high-availability middleware is an implementation example to synchronize states. Moreover, the same model can be applied to redundancy configurations between VNFs.

Redundancy model

The load balancing method in clause 6.2.3 is applied to a stateful VNF with high-availability middleware in this PoC. Each VNFC is configured to have correspondent VNFCs that store its backup data, including state information of the dialogues in normal conditions on a remote location.

When a VNFC fails, the correspondent VNFCs utilize their backup information and continue the service. At the same time, the VNFCs taking over the failed VNFC copy their backup data to other VNFCs, which are assigned as new backup VNFCs.

Moreover, when the VNFM or EM triggers scale up/down, load balancing will be adjusted among the VNFCs to include new VNFCs or without terminated VNFCs.

It is important to dispatch the requests to VNFCs which have related dialogue in a stateful service. However, the mechanism for load balancing becomes complex, because the number of VNFCs changes in LCM. In order to realize scalability and availability in LCM, there are two load balancing functions in this PoC: one simple load balancer and multiple advanced load balancers.

Lessons learned

The continuity and/or reliability required for services may vary depending on the service characteristics (i.e. application), and some applications provide their own redundancy functions to realize high-availability. In the lifecycle operation, VNFM and/or EM should work in close cooperation with VNFs such as giving notification of LCM, because VNFM/EM does not always know how VNFCs are being managed in VNFs.

A.3 PoC#35 Availability management with stateful fault tolerance

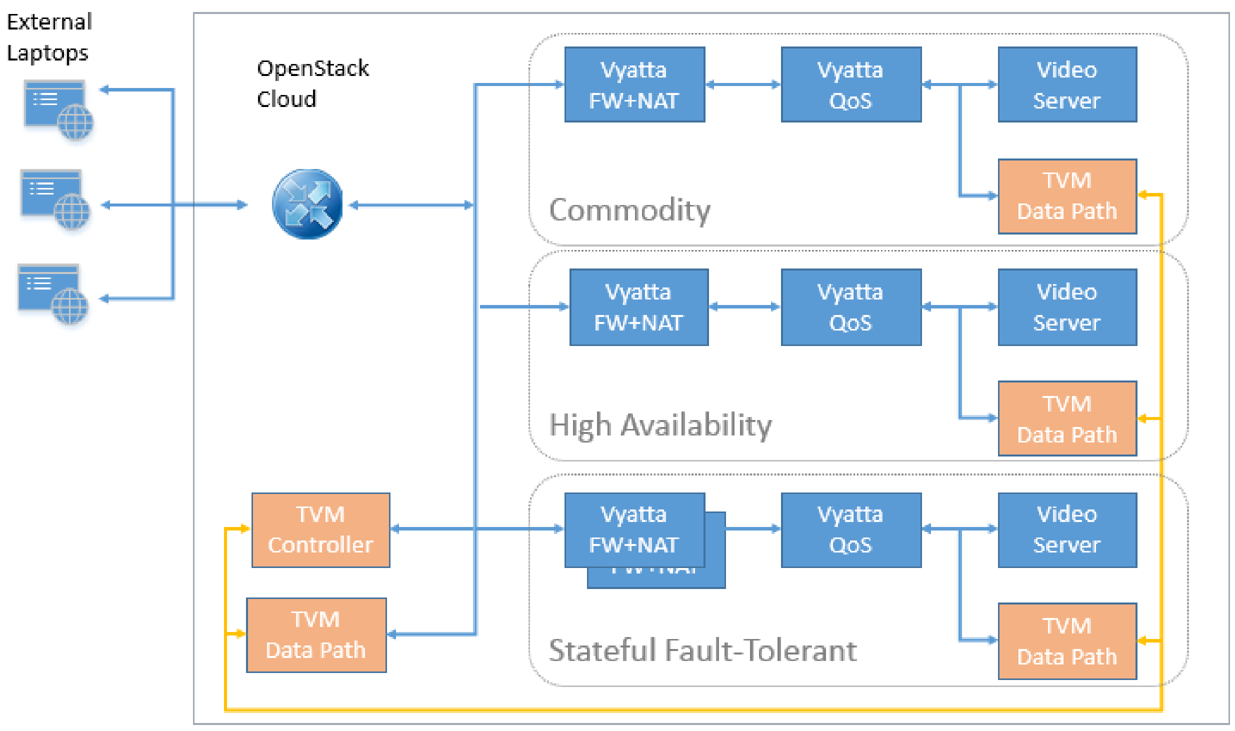


Figure A.2: PoC#35 - PoC configuration

PoC team

ATT
iBasis
NTT
Stratus Technologies
Aeroflex
Brocade
Allot

PoC demo

NFV World Congress, 6-8 May 2015, San Jose, CA, USA
OpenStack summit, 18-22 May 2015, Vancouver, BC, Canada
SDN & OpenFlow World Congress, 12-16 October 2015, Düsseldorf, Germany

Overview

This PoC demonstrated the deployment of a chain of stateful VNFs (and their constituent VNFCs) in three different resiliency levels, with increasing levels of service availability and service continuity. The availability related performance indications were qualitatively demonstrated utilizing streaming media test applications through each different resiliency level in the PoC demonstrations. The PoC also quantitatively characterized the key availability performance metrics for each of the different models. All resiliency support services and mechanisms in this PoC were implemented as infrastructure services (i.e. as a combination of mechanisms implemented in the compute nodes hosting the VNFCs, and the management services implemented as part of the associated management layers within the corresponding NFV-MANO components). The VNF (i.e. application) level availability and resiliency mechanisms were not shown in this PoC, as the key purpose of the PoC is to demonstrate the feasibility of implementing all associated mechanisms required as infrastructure services. The partner's VNFs demonstrated were Brocade Vyatta router instances configured as a stateful firewall and QoS enforcement function, and Aeroflex TeraVM instances for real-time monitoring of the end-to-end IP packet transfer service related performance metrics. In addition, open source video servers and other supporting elements were instantiated as required to support the qualitative demo. NFVI consisted of commodity hardware with Stratus availability services deployed in the compute nodes, while the NFV-MANO environment consisted of OpenStack enhanced with Stratus resiliency services.

Redundancy and resiliency models

The PoC demonstrated three different configurations of the resiliency mechanisms, in the increasing order of resiliency as follows:

- In the general availability mode, no VM instance redundancy, no VM state checkpointing and no automatic recovery were performed. However, other fault management functions such as detection and cleanup were performed by the NFVI and NFV-MANO services.

- In the high availability mode, no VM instance redundancy or VM state checkpointing was performed. However, fault detection and automatic recovery with re-instantiation is performed by the NFVI and NFV-MANO services.

- In the fault tolerance mode, each VNF is deployed with redundant VM instances and full-VM state checking (state pointing), and associated fault management mechanisms from detection to full recovery were automatically performed by the NFVI and NFV-MANO services, ensuring full VM state protection and fast, stateful failover without application awareness.

These three modes were selected to represent the baseline OpenStack performance (essentially no resiliency support), the use of OpenStack exposed services for repair through the re-instantiation under control of the resiliency layer software (essentially fault manager) utilizing those services in combination with fault event indications from the underlying NFVI/NFV-MANO services, and the full VM state protection and stateful VM failover as enabled by combination of node and NFV-MANO level mechanisms. It is important to note that many other combinations of underlying mechanisms are possible, but were not included in the PoC to limit the scope of the effort, as **the fault tolerance** mode implicitly demonstrates that all of the mechanisms to fully protect a VM instance against various failure modes, along with full VM state protection (enabling the associated service availability and service continuity aspects, respectively) can be implemented as infrastructure services (i.e. through combination of NFVI and NFV-MANO level mechanisms).

Lessons learned

The PoC#35 report [i.34] includes comprehensive treatment of the objectives, which were all successfully demonstrated, and associated detailed findings, with many suggestions for the future work required to sufficiently expose and standardize associated mechanisms and interfaces at various NFVI/NFV-MANO sub-layers and components to enable the interoperable fault management infrastructure services.

It is expected that new division of work between NFVI and NFV-MANO and VNFs (and constituent VNFCs) dictated by the NFV cloud architecture, as well as resulting multi-layer fault management will require coordination of responsibilities and/or timelines to fully cover the whole stack without adverse effects of uncoordinated fault detection and/or recovery mechanisms.

Many aspects of the various lifecycle phases associated with state protection and fault management cycle can also only be performed as infrastructure services, which means that even if the application chooses to take responsibility for aspects of state protection and recovery mechanisms internally, it will still need to interface to NFVI/NFV-MANO to utilize the associated infrastructure services for, e.g. instance placement, fast fault detection, fault correlation/localization, fault recovery and repair and other processes that are required for complete fault management solution implementation, particularly for the sufficient coverage of complex NFVI and NFV-MANO and non fail-stop failure modes.

The readers are encouraged to consult the full PoC#35 [i.34] report for further details.

The key findings of the PoC are as follows (from PoC#35 final report [i.34], section B2.5):

This PoC demonstrates that OpenStack based VIM mechanisms alone are insufficient for supporting the "carrier-grade" availability objectives. The baseline functionality by itself is only adequate for supporting development scenarios and non-resilient workloads.

The instance fault detection and associated event delivery mechanisms that are currently native to OpenStack are themselves too slow to support the fast failure recovery (implicitly, one cannot recover in time less than the detection time), even if the recovery mechanisms would otherwise be deployed on top of the baseline OpenStack based VIM installation.

It is not recommended that native OpenStack VIM only mode without additional fault management mechanisms is utilized for any mission critical applications, where failure induced outage performance is considered to be important.

As per the objectives of this PoC, it was successfully demonstrated that the VM based state replication can be used to ensure both high *service continuity* and *service accessibility* as NFVI and NFV-MANO services, and without the application awareness.

The implementation of the fault tolerance mode also implicitly demonstrates that all phases of the fault management cycle (fault detection, fault localization, fault isolation, fault recovery and fault repair) can be provided as infrastructure services (using a combination of NFVI and NFV-MANO level mechanisms). In addition to the fault management processes, it was demonstrated that the full VM stateful replication and recovery, which are required to ensure service continuity, can also be provided by the infrastructure without any application (i.e. VNF) level support mechanisms.

The service availability for an associated fault tolerance instance pair can be calculated to be over 7-nines, assuming the generally very conservative (i.e. high) failure frequency of once per year for all causes, and 500 ms maximum outage time. Due to very rapid repair processes enabled by homogenous resource pools in cloud (in seconds instead of hours), the second failure exposure risk can also be effectively minimized, while at the same time relaxing the constraints of physical repair process times, allowing deferred/batched maintenance processes instead of critical 24x7 maintenance with tight (e.g. 4 hours) total repair time targets. The recovery and repair times should be both improved over the values measured in this PoC.

Annex B (informative): Service quality metrics

User service qualities across end-to-end service chains fall into four broad categories:

- **Service outage downtime** - The telecom industry's standard objective and quantitative measurement for ***service outage downtime*** (SO) is given in [i.15] and includes prorating of partial capacity and partial functionality loss of primary functionality. Availability is mathematically defined as uptime divided by the sum of uptime and downtime.

- **Service reliability** - This characterizes the probability that a particular service transaction or operation will be completed with acceptable service quality. Call completion rates, call drop rates, handover success rates and provisioning fallout rates are examples of service reliability metrics that are often considered within some end-to-end context.

- **Service latency** - Users experience service transaction latencies such as the interval from pressing "Send" on their phone until they hear ringback, or the interval from when they press "Play" on their device and when streaming media begins rendering on their device. Inevitably, each user's service request (e.g. each call setup) will have a slightly different service latency, but across hundreds, thousands, millions or more operations, a statistical distribution of service latencies will emerge. Thus, service latencies are normally measured via statistical characteristics, like 90th percentile latency, 99th percentile latency and so on.

- **Application-specific qualities** - such as mean opinion score (MOS) for bearer voice and audio/visual synchronization (a.k.a., lip sync) for video.

End-to-end user service qualities like service outage downtime, call completion rates and 99th percentile call setup latency are key quality indicators so they are routinely measured and managed by service providers. Measuring any of these service qualities requires defining:

- **one or more measurement point**(s), such as the demarks for an "end-to-end" service or for a specific VNF or PNF instance;

- **method of measurement** detailing exactly how data will be gathered, such as how time stamped event records or performance measurements will be processed;

- **counting and exclusion rules**, such as only counting data from production (rather than lab) systems.

For example, [i.15] gives counting and exclusion rules, as well as method of measurement, for service outage downtime. Predictions of end-to-end service outage downtime, reliability, latency or other key quality indicators should strive to estimate the feasible and likely performance of the target indicator, so that the prediction is very close to the long term average of measured performance of the deployed service. For instance, the method of section *5.1 Steady State Operation Outage Downtime* [i.15] strives to predict the feasible and likely long term service outage downtime of a particular end-to-end service delivery chain.

Annex C (informative): Accountability-oriented end-to-end modelling

C.1 Steady state operation outage downtime

Service outage downtime of a demarcated service chain in steady state operation primarily comes from the following causes:

1) Service outage downtime (e.g. SO [i.15]) for detection and recovery of VNFs and PNFs in the service delivery chain. The root causes of these steady state failures are likely to be:

   a) activation of residual VNF and PNF software defects;

   b) random hardware failures or activation of residual infrastructure software defects that produce ***VM premature release events*** ETSI GS NFV-INF 010 **[**i.10**]**.

2) Network outage downtime between elements (i.e. VNFs and/or PNFs) within the demarcated service chain, as well as intra-VNF network outage downtime (e.g. ***network outage*** [i.10]).

3) Outage downtime for functional components offered as-a-Service (e.g. *Technology Component as-a-Service Outage* [i.10]).

The quality accountability framework ETSI GS NFV-REL 005 [i.14], coupled with service quality metrics [i.10], enables most availability modelling details to be encapsulated by cloud service providers so that the cloud service customers can focus on the outage downtime performance of the resources they require, rather than being troubled by the cloud service provider's implementation details. For example, cloud service customers can focus on service key quality indicators of their key suppliers, like the rate of ***VM premature release events*** [i.10] or ***network outage*** [i.10] downtime actually delivered by their cloud infrastructure service provider.

C.2 Lifecycle operation

Each configuration change action to a network service, network function or network connection introduces the risk of producing a fault or activating a residual defect which cascades into a user service impacting incident. While automatically executed lifecycle management actions are not subject to human procedural errors, a portion of those automatically executed actions will occasionally fail; *B.2 Automated Lifecycle Management Action Failures* [i.14], reviews several likely failure modes. [i.16] gives a quality measurement model for automated lifecycle management actions based on supplier promise times and measurement of on-time delivery (relative to promise times) as well as service quality of each action. These measurements, together with the NFV Quality Accountability Framework [i.14] enable cloud service customers to hold appropriate parties accountable for service impact of failed lifecycle management actions.

C.3 Disaster operation objectives

Physical data centres and networking facilities are vulnerable to a range of force majeure and catastrophic events - like earthquakes, terrorist acts, fires and emergency power-off actions - that overwhelm high availability mechanisms with large correlated failures; these incidents can render some, or all, of an NFV data centre temporarily, or permanently, unavailable. A best practice is to assure business continuity via disaster recovery plans. Such best practice for disaster recovery planning of IT systems is to set both recovery time objective (RTO) and recovery point objective (RPO):

1) XX % of service users impacted by a disaster event (e.g. data centre outage) should have their service recovered within YY minutes (RTO requirement).

2) No persistent application data changes committed more than ZZ minutes before a disaster event (e.g. data centre outage) should be lost (RPO requirement).

Cloud service customers are responsible for their RTO and RPO requirements, and they will usually cascade budgets and appropriate requirements to their developers, integrators, VNF suppliers and cloud service providers as part of their disaster recovery planning.

Annex D (informative): Automated diversity

D.1 Randomization approaches

The techniques described below automatically randomize some aspects of a program, thus producing a diversity of program versions [i.23]. Diversity occurs in memory, in the operating system, in the bytecode or in the source code but, in all cases, it happens with no human intervention, through random processes.

Static randomization

One can create different versions of the same program, e.g. one can randomize the data structures at the source or at the binary level. An initial work in this domain highlights two families of randomization: randomly adding/deleting non-functional code and reordering code. Those transformations are described in the context of operating system protection, and data structure of C code, i.e. re-ordering fields of data structures (struct and class in C/C++ code) and inserting garbage ones.

Instruction-set randomization consists of creating a unique mapping between artificial CPU instructions and real ones. This mapping is encoded in a key which has to be known at runtime to execute the program. Eventually, the instruction set of a machine can be considered as unique, and it is hard for an attacker ignoring the key to inject executable code. Instruction-set randomization can be done statically (a variant of the program using a generated instruction set is written somewhere) or dynamically (the artificial instruction set is synthesized at load time). In both cases, instruction-set randomization creates indeed a diversity of execution which is the essence of the counter-measure against code injection.

In some execution environments (e.g. x86 CPUs), a "NOP" (no operation) instruction is used for the sake of optimization, in order to align instructions with respect to some criteria (e.g. memory or cache). By construction, NOP does nothing and the insertion of any amount of it results in a semantically equivalent program. However, it breaks the predictability of program execution and mitigates to this extent certain exploits.

Dynamic randomization

Randomization points are automatically integrated in the executable program. For instance, a memory allocation (malloc) primitive with random padding is a randomization point: each execution of malloc yields a different result. Contrary to static randomization, there is still one single version of the executable program, but their executions are diverse. Operating system randomization randomize the interface between the operating system and the user space applications: system call numbers, library entry points (memory addresses) and stack placement. All these techniques are dynamic, done at runtime using load-time pre-processing and rewriting.

Dynamic randomization can address different types of problems. In particular, it mitigates a large range of memory error exploits. This approach is based on three kinds of randomization transformations: randomizing the base addresses of applications and libraries memory regions, random permutation of the order of variables and routines, and random introduction of random gaps between objects.

Static randomization creates diverse versions of the same program at compilation time, while dynamic randomization creates diverse executions of the same program under the same input at runtime, i.e. introduction of randomization points. In the just-in-time compilation approach, randomization happens directly in the compiler. It is based on two diversification techniques: insertion of NOP instructions and constant blinding.

In the previous techniques, the support for dynamic randomization is implemented within the execution environment. On the contrary, self-modifying programs embed their own randomization techniques. This is done for sake of security and is considered one of the strongest obfuscation mechanisms.

Data diversity, used for fault tolerance, represents another family of randomization, enabling the computation of a program in the presence of failures [i.24]. When a failure occurs, the input data is changed so that the new input does not result in a failure. The output based on this artificial input, through an inverse transformation, remains acceptable in the domain under consideration. This technique dynamically diversifies the input data.

Environment diversity refers to techniques that change the environment to overcome failures. For instance, changing the scheduler, or its parameters, is indeed a change in the environment. This is larger in scope than just changing some process data, such as standard randomization.

It is noteworthy that diversification techniques can be stacked: as such, one can stack static and dynamic randomization. In this case, there are diverse versions of the same program which embed randomization points that produce themselves different executions.

Unsound randomization

This technique is different from traditional randomization ones which are meant to produce programs or executions that are semantically equivalent to the original program or execution, e.g. recombination of binary object files of commodity applications. If an application is made of two binary files A and B, it is possible to run the application by artificially linking a version of A with a different, yet close, version of B. The technique enables to tolerate bugs and even let new functions emerging but has no guarantee on the behaviour of the recombination.

Software can be mutated and, at the same time, it can preserve a certain level of correctness. Using an analogy from genomics, the "software mutational robustness" property has a direct relation to diversification: one can mutate the code in order to get functionally equivalent variants of a program. Doing this in advance is called "proactive diversity".

Different transformation strategies were deployed on Java statements to synthesize "sosie" programs. The sosies of a program P are variants of P, i.e. different source code, which pass the same test suite and that exhibit a form of computation diversity. The technique synthesizes large quantities of variants, which provide the same functionality as the original through a different control or data flow, reducing the predictability of the program's computation.

D.2 Integrated approach for diversity

Stacked diversity

The intuition is that each kind of artificial diversity has value in one perspective (a specific kind of attack or bug), and thus, integrating several forms of diversity should increase the global ability of the software system with respect to security or fault tolerance [i.17].

Multi-level program transformation aims at introducing diversity at multiple levels in the control flow so as to provide in-depth obfuscation. This work on program transformation takes place in the context of software architecture for survivable systems. As an example, the architecture relies on probing mechanisms that integrate two forms of diversity: in time (the probe algorithms are replaced regularly) and in space (there are different probing algorithms running on the different nodes of the distributed system).

A technique was developed for address obfuscation in order to thwart code injection attacks. It relies on the combination of several randomization transformations: randomize base addresses of memory regions to make the address of objects unpredictable; permute the order of variables in the stack; and introduce random gaps in the memory layout. Since all these transformations have a random component, they synthesize different outputs on different machines, thus increasing the diversity of attack surfaces that are visible to attackers.

A report of the DARPA project Self-Regenerative System summarizes the main features of the Genesis Diversity Toolkit. This tool is one of the recent approaches that integrate multiple forms of artificial diversity. Its goal is to generate 100 diverse versions of a program that are functionally equivalent, but for which a maximum of 33 versions have the same deficiency. The tool supports the injection of 5 forms of diversity:

i) address space randomization;

ii) stack space randomization;

iii) simple execution randomization;

iv) strong instruction set randomization;

v) calling sequence diversity.

Super diversification is proposed as a technique that integrates several forms of diversification to synthesize individualized versions of the programs. The approach, inspired by compilation super optimization, consists in selecting sequences of bytecode and in synthesizing new sequences that are functionally equivalent. Given the very large number of potential candidate sequences, several strategies are discussed to reduce the search space, including learning occurrence frequencies of certain sequences.

Another approach advocates for massive-scale diversity as a new paradigm for software security. The idea is that today some programs are distributed several million times, and all these software clones run on millions of machines worldwide. The essential issue is that, even if it takes a long time for an attacker to discover a way to exploit a vulnerability, this time is worth spending since the exploit can be reused to attack millions of machines. In a new envisioned context, each time a binary program is shipped, it is automatically diversified and individualized, to prevent large-scale reuse of exploits.

Moving Target Defense (MTD) was announced in 2010 as one of the three "game changing" themes to cyber security. The software component of MTD integrates spatial and temporal software diversity, in order to "limit the exposure of vulnerabilities and opportunities for attack". With such a statement, future solutions for MTD will heavily rely on the integration of various software diversity mechanisms to achieve their objectives.

Multiple forms of diversity and code replacement in a distributed system are proposed in order to protect from remote man-at-the-end attacks. The diversification transformations used for spatial diversity are adapted from obfuscation techniques. They combined with temporal diversity (when and how frequently diversity is injected), which relies on a diversity scheduler that regularly produces new variants.

A recent proposal uses software diversification in multiple components of Web applications. It combines different software diversification strategies, from the deployment of different vendor solutions, to fine-grained code transformations, in order to provide different forms of protection. This form of multi-tier software diversity is a kind of integrated diversity in application-level code.

Controllers of automated diversity

If mixed together and put at a certain scale of automation and size, all kinds of automated diversity need to be controlled. Indeed, diversity controlled with specific management decisions is better than naive diversity maximization. The idea of N-variant systems consists in automatically generating variants of a given program and then running them in parallel in order to detect security issues. This is different from N-version programming, because the variants are generated automatically and not written manually. The approach is integrated because it synthesizes variants using two different techniques: address space partitioning and instruction set tagging. Both techniques are complementary, since address space partitioning protects against attacks that rely on absolute memory addresses, while instruction set tagging is effective against the injection of malicious instructions.

A name proposed for this concept is "multi-variant execution environment". It provides support for running multiple diverse versions of the same program in parallel. These versions are automatically synthesized at compilation time, with reverse stack execution. The execution differences allow some kind of analysis and reasoning on the program behaviour. This concept enables to detect malicious code trying to manipulate the stack.

"Collaborative application communities" proposes that the same application (e.g. a Web server) is run on different nodes: in presence of bugs (invalid memory accesses), each node tries a different runtime fix alternative. If the fix proves to be successful, a controller shares it among other nodes. This healing process contains both a diversification phase (at the level of nodes) and a convergence phase (at the level of the community).

Annex E (informative): Software reliability models

E.1 Exponential times between failures modes

Jelinski-Moranda

This very first reliability model [i.22] is based on the following hypotheses:

- the software contains at t_0 an unknown number of faults N;
- when a failure occurs, its related fault is corrected without introduction of new faults;
- the failure intensity λ_t is proportional to the number of residual faults (Figure E.1).

If Φ represents the proportionality coefficient, $\lambda_1 = N\,\Phi$, $\lambda_2 = (N-1)\,\Phi$, …, $\lambda_{n+1} = (N-n)\,\Phi$, …, $\lambda_N = \Phi$, and for $t \geq t_N : \lambda = 0$. The replacement of λ_I by (N-i+1) in the formulas given in the ETBF section of clause 5.3.3 thus provides estimation of R(t), MTTF and $E[T_n]$.

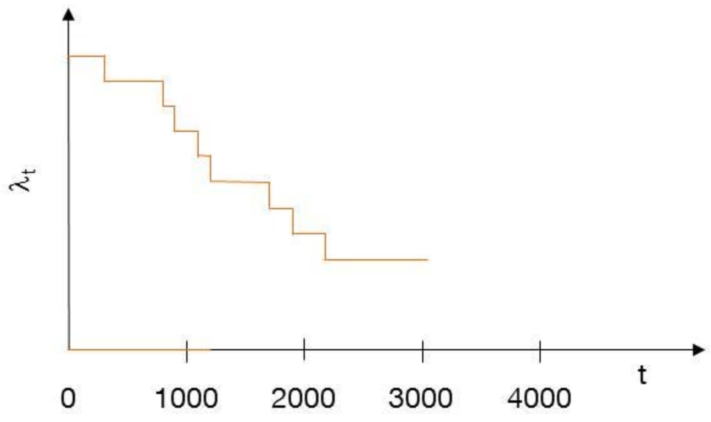


Figure E.1: Jelinski-Moranda failure intensity

Although this pioneering model has been extensively used in the past, it presents some drawbacks:

- the correction of the N initial faults is supposed to leave a perfect software, which is not plausible for a complex program;
- faults do not have the same occurrence rate - indeed, a fault may manifest at each execution, while another only after a million executions;
- as any activated fault is not identified and not perfectly corrected, this model is too optimistic, i.e. R(t) is over estimated.

Geometric model

The previous model requires that the faults have the same occurrence rate, i.e. the failure intensity's decrease is the same after each correction. As faults may encounter different severity levels, e.g. serious faults happening earlier leading to a drop of λ_t after their correction, software enhancement is much higher at the start of the observation period (i.e. the corrections' impact decreases with time), a solution is to have a geometric, and not arithmetic, decrease of the failure intensity, i.e. $\lambda_i = c\,\lambda_{i-1}$ (and not $\lambda_i = \lambda_{i-1} - \Phi$). The reliability growth thus results in $c < 1$, while $c = 1$ reflects a Homogeneous Poisson Process ($\lambda_i = \lambda$).

The failure intensity of this geometric model [i.27] can be written: $\lambda_t = \lambda\, c^n$ with the initial failure intensity $\lambda = \lambda_1$ and c representing the correction quality (i.e. a small value of c corresponds to a good correction, while the correction is totally inefficient if c = 1). A reduction of the correction efficiency with time is shown in Figure E.2 with the different variation steps of λ_t decreasing geometrically.

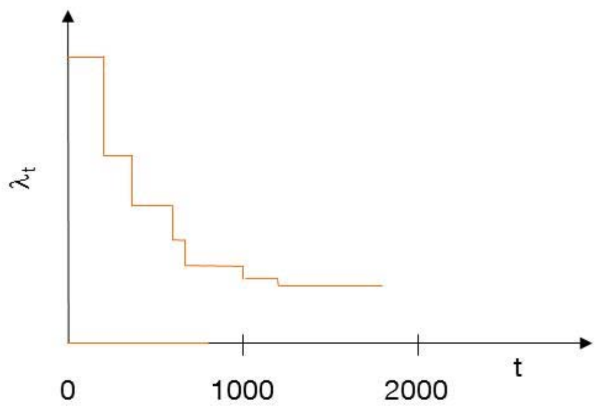


Figure E.2: Failure intensity of the geometric model

The useful formulas thus become: $R(t) = \exp(-\lambda c^n t)$

$MTTF = 1/(\lambda c^n)$

E.2 Non-homogeneous Poisson processes

Power-law process

Issued from observations of electrical equipment failures [i.18] and [i.20], this NHPP model introduces the following failure intensity: $\lambda(t) = \alpha\beta t^{\beta-1}$, with α=scale parameter, and β=degree of the system's degradation or improvement. Three cases are thus possible (Figure E.3):

- $\beta>1$: $\lambda(t)$ increases with time (decrease of reliability);
- $\beta<1$: $\lambda(t)$ decreases with time (reliability growth);
- $\beta=1$: $\lambda(t)$ constant (stabilized reliability, i.e. HPP).

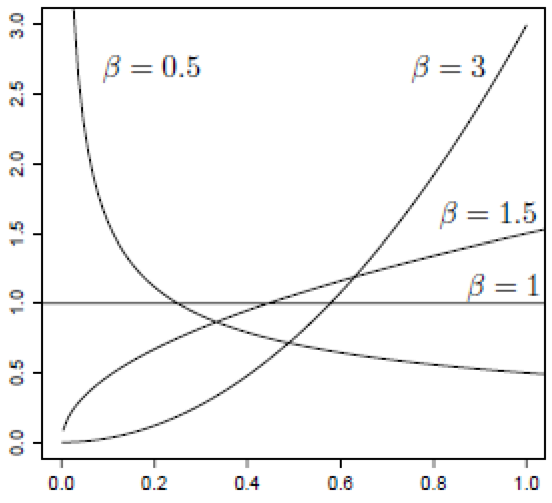


Figure E.3: Failure intensity of the Duane's model

It can be shown that: $R_t(\tau) = \exp(-\alpha[(t+\tau)^{\beta} - t^{\beta}])$

$MTTF_t = \int_0^{\infty} R_t \, \tau \, d\tau$

$E[N_t] = \alpha t^{\beta}$

The estimation of this model's parameters leads to:

$$\hat{\alpha}_n = \frac{n}{T_n^{\hat{\beta}_n}}$$

$$\hat{\beta}_n = \frac{n}{\sum_{i=1}^{n-1} \ln \frac{T_n}{T_i}}$$

Goel-Okumoto

This model [i.21] is based on the following assumptions:

- the software contains at t_0 a random number of faults whose mean value is a;
- when a failure occurs, its related fault is immediately corrected without introduction of new faults;
- the failure intensity is proportional (through a factor b) to the mean value of residual faults.

These hypotheses are quite similar to those of the Jelinska-Moranda model, except that the initial number of faults is a random variable (and not a constant), and λ(t) depends on the mean value of residual faults (and not the exact number of residual faults). The useful formulas become:

$\lambda(t) = a \, b \exp(-bt)$

$R_t(\tau) = \exp(-a \exp(-bt)[1 - \exp(-b\tau)])$

$MTTF_t = \int_0^{\infty} \exp(-a \exp(-bt)[1 - \exp(-b\tau)] \, d\tau$

To estimate this model's parameters, one needs to solve the following equations:

$$\frac{n}{\hat{b}_n} - \sum_{i=1}^{n} T_i - \frac{nT_n \exp(-\hat{b}_n T_n)}{1 - \exp(-\hat{b}_n T_n)} = 0$$

$$\hat{a}_n = \frac{n}{1 - \exp(-\hat{b}_n T_n)}$$

Some other NHPP models

1) Musa-Okumoto [i.28]: derived from the geometric model, its failure intensity can be written:

$$\lambda(t) = \frac{\lambda}{1 + \lambda\gamma t}$$

2) S-shaped models: in the previous software reliability models, the failure intensity increases or decreases in a monotonous way. In practical situations, failures are distant from each other at the beginning, then get closer, and end up to be distant again. This is interpreted as follows (if the data are issued from the test phase): at the start of the tests, bugs are difficult to find. As testers gain knowledge about the program, faults are discovered much easier. Finally, as most of the faults have been identified, it is more and more difficult to find new ones. This phenomenon is represented in Figure E.4 by a failure intensity which starts increasing, and ends up decreasing following an S-shape [i.30]:

$$\lambda(t) = a \, b^2 \, t \exp(-bt).$$

3) The numerous variants of this model include imperfect bug correction, delayed bug correction, dependence between faults, environmental parameters, modular software decomposition, failures type classification, tests and corrections costs.

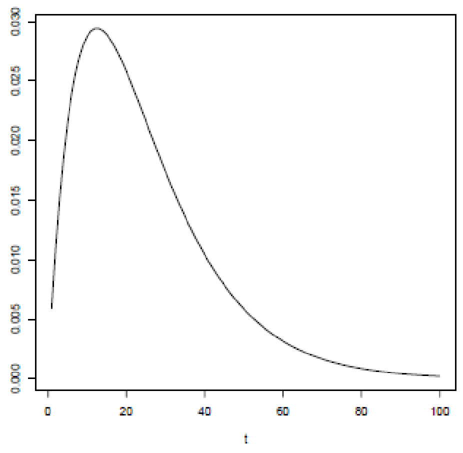


Figure E.4: S-shaped failure intensity

Annex F (informative): Authors & contributors

The following people have contributed to the present document:

Rapporteur:
Hidefumi Nakamura, NTT corporation

Other contributors:
Randee Adams, Alcatel-Lucent
Stefan Arntzen, Huawei Technologies
Zaw Htike, KDDI Corporation
Eriko Iwasa, NTT Corporation
Kenta Shinohara, NTT Corporation
Atsuyoshi Shirato, NTT Corporation
Chidung Lac, Orange
Shaoji Ni, Huawei Technologies
Simon Oechsner, NEC
Marcus Schöller, NEC
Pasi Vaananen, Stratus Technologies Inc.

Annex G (informative): Change History

Date	Version	Information about changes
November 2014	0.0.1	Scope and Table of Contents
8 June 2015	0.1.0	NFVREL(15)000041r2, NFVREL(15)000078, NFVREL(15)000036r1, NFVREL(15)000079 are incorporated.
25 July 2015	0.2.0	Disclaimer text is included. The contributions, NFVREL(15)000046r5, NFVREL(15)000108r1 and NFVREL(15)000122 are incorporated.
6 August 2015	0.3.0	NFVREL(15)000054r5, NFVREL(15)000055r3, NFVREL(15)000151r1, NFVREL(15)000169r1 and NFVREL(15)000172 are incorporated. Some editorial errors are corrected.
23 October 2015	0.4.0	NFVREL(15)000070r4 NFVREL(15)000186 NFVREL(15)000199r1 NFVREL(15)000215r1 NFVREL(15)000224r2 NFVREL(15)000238r1 NFVREL(15)000261, NFVREL(15)000233, NFVREL(15)000242, NFVREL(15)000247r1, NFVREL(15)000264, NFVREL(15)000266 and NFVREL(15)000073r7 are incorporated. Some editorial errors are corrected.
16 October 2015	0.5.0 and 1	NFVREL(15)000219r5, NFVREL(15)000275r1 and NFVREL(15)000273r1 are incorporated. Some editorial errors are corrected. Alignment with ETSI editing rules is partially done.
3 December 2015	0.5.2	NFVREL(15)000219r7 is included. Clause numbering is also corrected.
16 December 2015	0.5.3	NFVREL(15)000296r1 and NFVREL(15)000297r1 are included. Clause numbering is also corrected accordingly.
4 January 2016	0.5.4	NFVREL(15)000285r2 is included.
10 January 2016	0.5.5	NFVREL(15)000288r2 is included.
15 January 2016	0.5.6	NFVREL(15)000307r3, NFVREL(16)000002r3, NFVREL(16)000004r2 and CR1 to 8 of NFVREL(16)000008 are included.
29 January 2016	0.5.7	NFVREL(16)000009r1 is included. Some editorial corrections in annex D are made.
7 February 2016	0.5.8	NFVREL(16)000023, NFVREL(16)000019r3, NFVREL(16)000024r1 and NFVREL(16)000011r6 are integrated. Editorial corrections are made.
18 February 2016	0.6.0	NFVREL(16)000027, 30, 31, 35 are integrated, and comments are solved. Edited within the Dublin meeting as NFVREL(16)000036..
16 March 2016	0.7.0	Changes in this version include: • Removal of all SHALLs and MUSTs by Marcus and Hidefumi (NFVREL(16)000039r1) • Additional feedback from ETSI (Milan & Laurent) to improve the removal of SHALLs and MUSTs. • Editorial CR in NFVREL(16)000042 • CR in NFVREL(16)000043r1

History

Document history		
V1.1.1	April 2016	Publication
V1.1.2	July 2016	Publication