Pós-Graduação em Ciência da Computação

AVAILABILITY AND CAPACITY MODELING FOR VIRTUAL NETWORK FUNCTIONS BASED ON REDUNDANCY AND REJUVENATION SUPPORTED THROUGH LIVE MIGRATION

By

ERICO AUGUSTO CAVALCANTI GUEDES

Ph.D. Thesis

Federal University of Pernambuco
posgraduacao@cin.ufpe.br
www.cin.ufpe.br/~posgraduacao

RECIFE

2019

ERICO AUGUSTO CAVALCANTI GUEDES

AVAILABILITY AND CAPACITY MODELING FOR VIRTUAL NETWORK FUNCTIONS BASED ON REDUNDANCY AND REJUVENATION SUPPORTED THROUGH LIVE MIGRATION

A Ph.D. Thesis presented to the Center for Informatics of Federal University of Pernambuco in partial fulfillment of the requirements for the degree of Philosophy Doctor in Computer Science.

Concentration Field: *Performance Evaluation and Dependability*

Advisor: Paulo Romero Martins Maciel

RECIFE
2019

Tese de doutorado apresentada por **Erico Augusto Cavalcanti Guedes** ao programa de Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, sob o título **Availability and Capacity Modeling for Virtual Network Functions based on Redundancy and Rejuvenation Supported through Live Migration**, orientada pelo **Prof. Paulo Romero Martins Maciel** e aprovada pela banca examinadora formada pelos professores:

Prof. Dr. Antonio Alfredo Ferreira Loureiro (Examinador Externo)
Departamento de Ciência da Computação/UFMG

Prof. Dr. Edmundo Roberto Mauro Madeira (Examinador Externo)
Instituto de Computação/Unicamp

Prof. Dr. Djamel Fawzi Hadj Sadok (Examinador Interno)
Centro de Informática/UFPE

Prof. Dr. Nelson Souto Rosa (Examinador Interno)
Centro de Informática/UFPE

Prof. Dr. Paulo Roberto Freire Cunha (Examinador Interno)
Centro de Informática/UFPE

RECIFE
2019

Acknowledgements

I would like to thank my family, friends, professors, and my advisor Paulo Maciel, for all the support over these years.

I am very grateful to my mother, Adelair.

A very special thanks to Lorena, for her admirable patience, support, presence, understanding, and care. Without your love, this work would not be possible.

A special thanks to my friends Patricia Endo for her constant support, Marco Domingues, who helped me at the initial stages of my graduate and research studies, and Renata Vilela, for listening to me at difficult times.

Thanks to the professors and coworkers of the Federal Institute of Education, Science, and Technology of Alagoas - Campus Palmeira dos Índios, among them: Carlos Antônio, Rafael Antonello, Pablo Tibúrcio, Rodrigo Lustosa, and Carlos Guedes.

Thanks to my cousins Jefferson, Kleber, and Clériston, and to my aunt Maria do Socorro.

To all my beloved friends of Miraflôres Building, thank you so much.

Thanks for collaboration to several colleagues I made at CIn/UFPE and MODCS Research Group: Júlio Mendonça, Eric Borba, Iure Fé, Jonas Pontes, Jamilson Dantas, Ronierison Maciel, Matheus Torquato, Débora Stefani, Renata Dantas, Jean Teixeira, Carlos Julian, Airton Pereira, Paulo Roberto, Danilo Mendonça, Bruno Silva, and Rubens Matos.

Finally, I would like to thank IFAL/PIn, CIn/UFPE, and FACEPE for the support provided in this thesis.

If you think high availability is expensive, try downtime.

—TERRY CRITCHLEY

Resumo

O sucesso da virtualização de servidores e da computação em nuvem levou a um subsequente requisito de virtualização de rede, porque a flexibilidade alcançada pelos recursos de hardware virtualizados poderia ser prejudicada por interconexões de rede estáticas. A virtualização de rede refere-se à capacidade de executar instâncias virtuais de roteadores, switches e links sobre um substrato de rede físico. Assim, várias redes virtualizadas podem coexistir em uma infraestrutura de rede comum. Tecnologias como Redes Definidas por Software, Virtualização de Funções de Rede e Encadeamento de Funções de Serviços foram lançadas para permitir a substituição de dispositivos de hardware de rede tradicionais por cadeias lógicas de Funções de Redes Virtuais (VNFs - Virtual Network Functions). Como uma conseqüência, as redes virtualizadas representam obstáculos adicionais ao fornecimento de serviços de alta disponibilidade, porque resultam em mais camadas de software: o número crescente de componentes de software necessários para executar sistemas virtualizados também aumenta o número de possíveis falhas. Esta tese projetou e avaliou um conjunto de modelos estocásticos para melhorar o fornecimento de funções de rede virtual, considerando métricas de disponibilidade e capacidade. Os modelos são capazes de representar mecanismos de alta disponibilidade, como redundância e rejuvenescimento de software, permitindo estimar o comportamento das métricas estudadas diante desses mecanismos. A metodologia adotada abrange a montagem e configuração de uma infraestrutura de alta disponibilidade de computação em nuvem. A nuvem implementada suporta o fornecimento de VNFs e cadeias de serviços virtuais redundantes, permitindo a medição de valores dos parâmetros a serem injetados nos modelos. Para mostrar a aplicabilidade das soluções propostas, também é apresentado um conjunto de estudos de caso. Os resultados demonstram a viabilidade em fornecer cadeias de VNFs em uma infraestrutura de nuvem para os cenários estudados, e podem ser úteis para provedores e operadoras de telecomunicações nas suas infraestruturas heterogêneas.

Palavras-chave: Funções de Rede Virtuais, Virtualização de Funções de Rede, Cadeias de Funções de Serviços, Alta Disponibilidade, Modelagem Estocástica, Agrupamento, Envelhecimento e Rejuvenescimento de Software

Abstract

The success of server virtualization and cloud computing led to a subsequent network virtualization requirement, because the flexibility achieved by virtualized hardware resources could be impaired by static network interconnections. Network virtualization refers to the ability to execute virtual instances of routers, switches, and links on top of a physical network substrate. So, multiple virtualized networks can co-exist in a shared network infrastructure. Technologies such as Software-Defined Networks, Network Function Virtualization and Service Function Chaining have been launched to enable the replacement of traditional network hardware appliances by softwarized Virtualized Network Function (VNF)s chains. As a consequence, virtualized networks represent additional obstacles to the provision of high availability services, because it results in more layers of software: the increasing number of software components required to run virtualized systems also increases the number of possible failures. This thesis designed and evaluated a set of stochastic models to improve virtual network functions provision considering metrics of availability and capacity. The models can represent high availability mechanisms, such as redundancy and software rejuvenation, allowing to estimate the behavior of the studied metrics facing these mechanisms. The adopted methodology encompasses the assembling and configuration of high available cloud computing infrastructure. The implemented cloud supports the provision of redundant virtual network functions and service function chains, enabling the measurement of parameter values that were injected in the designed models. In order to show the applicability of proposed solutions, a set of case studies are also presented. The results demonstrate the feasibility in providing high available Virtual Network Functions and Service Function Chains in a cloud infrastructure for the studied scenarios. Such results can be useful for telecommunication providers and operators and their heterogeneous infrastructures.

Keywords: Virtual Network Functions, Network Function Virtualization, Service Function Chaining, High Availability, Stochastic Modeling, Clustering, Software Aging and Rejuvenation

List of Figures

List of Tables

List of Acronyms

AMQP Advanced Message Queuing Protocol 33
API Application Programming Interface 18
AWS Amazon Web Services 17
CAPEX Capital Expenditure 18
COA Capacity Oriented Availability 48
COUA Capacity Oriented Unavailability 48
CT Container 67
CTMC Continuous Time Markov Chain 44
DC Data Center 53
DPI Deep Packet Inspection 18
ETSI European Telecommunications Standards Institute 18
GRE Generic Routing Encapsulation 25
HA High Availability 20
IaaS Infrastructure as a Service 30
IoT Internet of Things 25
IETF Internet Engineering Task Force 27
KVM Kernel-based Virtual Machine 60
NAT Network Address Translation 32
NaaS Network as a Service 32
NIST National Institute of Standards and Technology 17
NFV Network Function Virtualization 18
NIC Network Interface Card 24
NTP Network Time Protocol 33
OS Operating System 17
OPEX Operational Expenditure 18
OVS Open vSwitch 25
PaaS Platform as a Service 30
RA Resource Agent 38
RBD Reliability Block Diagram 43

S3 Simple Storage Service 19

SaaS Software as a Service 30

SDN Software-Defined Networking 17

SFC Service Function Chaining 18

SLA Service Level Agreement 19

SPOF Single Point Of Failure 20

SPN Stochastic Petri Net 43

UGC User Generated Content 66

vNIC Virtual Network Interface Card 25

VCS Virtual Computer System 17

VIM Virtualized Infrastructure Manager 52

VIP Virtual IP 69

VM Virtual Machine 17

VMM Virtual Machine Monitor 17

VNF Virtualized Network Function 18

VLAN Virtual Local Area Network 24

VXLAN Virtual eXtensible Local Area Network 25

WSGI Web Server Gateway Interface 32

Contents

1

Introduction

Virtualization is a technique to abstract the resources of computer hardware, decoupling the application and operating system from the hardware, and dividing resources into multiple execution environments. Previously to virtualization, it was not uncommon to accommodate only one application per Operating System (OS), i.e., per server. This approach was known as server proliferation. It increases the availability of services, mainly in scenarios where reboot the OS was overused as main troubleshooting action. However, server proliferation promotes machine's resources wastage.

Goldberg [50], in 1976, stated the concept of Virtual Computer System (VCS), also called as Virtual Machine (VM). VMs were conceived to be very efficient simulated copies of the bare metal host machine. One new layer of software, called Virtual Machine Monitor (VMM), was adopted to mediate the communication between the VMs and the hardware resources.

The main motivation for the adoption of virtualization was to increase the efficiency of hardware resources, with a direct effect on the contraction of infrastructure costs, dropping power requirements at a green data center effort.

With the adoption of VMs, server virtualization enables the consolidation of services. Previously isolated into individual machines, the services started to be provided in VMs that share virtualized server resources.

In 2006, Amazon Web Services (AWS) began offering IT infrastructure services to businesses in the form of web services, which is now known as cloud computing [8]. In 2011, the five essential characteristics that define cloud computing were standardized by National Institute of Standards and Technology (NIST) [82], namely: on-demand self-service, broad network access, resource pooling, rapid elasticity, and measured service. Such a set of features results in an increased rate of changes in networks.

The success of virtualization and cloud computing led to subsequent network virtualization requirement, because static network interconnections could impair the flexibility achieved by virtualized hardware resources. Cloud providers need a way to allow multiple tenants to share the same network infrastructure. It was needed to virtualize the network.

Similarly to decoupling between OS and bare-metal hardware occurred in server virtualization, Software-Defined Networking (SDN) [65] implements the decoupling between the

control plane (which decides how to handle the traffic) from the data plane (which forwards traffic according to decisions that the control plane makes) in network interconnection devices, such as switches. Such a decoupling also enables the consolidation of control planes to a single control program managing all data plane devices [41]. So, SDN relates to network virtualization as an enabling technology.

The SDN control plane performs direct control over the state in the network's data-plane elements via a well-defined Application Programming Interface (API). From 2007 to around 2010, the OpenFlow [45] API development represented the first instance of widespread adoption of an open interface, providing ways to make a practical control-data plane separation. The most adopted implementation of OpenFlow is Open vSwitch [43], functioning as a virtual switch in VM environments.

In 2012, an European Telecommunications Standards Institute (ETSI) white paper proposed Network Function Virtualization (NFV) [38] an alternative to reduce Capital Expenditure (CAPEX) and Operational Expenditure (OPEX) through the virtualization of network specialized (and expensive) hardware, known as appliances. These expensive appliances run a set of services (such as firewalling, load balancing, and Deep Packet Inspection (DPI)) throughout traffic aggregation points in the network, intending to apply traffic policies. They are added in an over-provisioned fashion, wasting resources. Several other issues contribute to the deprecation of adding appliances model:

- additional cost of acquisition, installing, managing, and operation. Each appliance requires power, space, cabling, and an all lifecycle that must be managed;
- network virtualization. As a virtual network topology can be moved to diverse servers in the network, it is problematic to move the appliances to accomplish the dynamism of virtualized networks.

As can be noted, virtualized cloud infrastructures claim for connectivity for migration of VMs, in several use cases, a live migration. Virtualized network appliances should become mobile. Its service continuity could be improved by mobile network infrastructure. The ossification of traditional data center networks denied the agile and required mobility of VMs because it requires manual tunning in the physical infrastructure.

Virtualized Network Function (VNF) replaces vendor's appliances by systems performing the same functions, yet running on generic hardware through the adoption of server virtualization. Chains of VNFs quickly emerged and can be mobile. The term Service Function Chaining (SFC) [104, 54] was used to describe an ordered list of VNFs, and the subsequent steering of traffic flows through those VNFs. SDN can handle the classification and forwarding tasks required by SFCs.

As a consequence of the massive adoption of server virtualization, cloud computing, and network virtualization was the emergence of new network architectures designed to be fault tolerant [22]. Regarding resilience, NFV moves the focus from physical network nodes, that are

highly available, to highly available end-to-end services comprised of VNFs chains [37].

1.1 Motivation and Justification

As we have seen, several technologies were launched to enable the replacement of traditional network hardware appliances by softwarized VNFs chains. As a consequence, virtualized networks demands greater efforts over availability, because it represents more layers of software: the increasing number of software components required to run a cloud also increases the number of possible failures [125].

According to Han et al. [56], the virtualization of network services "*may reduce capital investment and energy consumption by consolidating networking appliances, decrease the time to market of a new service [..], and rapidly introduce targeted and tailored services based on customer needs*". However, along with the benefits, there are many technical challenges to be covered by the network operators. For instance, ensuring that the network resilience will remain at least as good as that of commodity hardware implementations, even if relying on virtualization, is a great challenge. The VNF chains need to ensure the availability of its part in the end-to-end service, just as in the case of non-virtualized appliances [58].

Furthermore, the network operators should also be able to dynamically create and migrate their SFCs in order to consolidate VNFs or to provide service elasticity based on user demand or traffic load. When migrating SFCs, the network operator should keep in mind that service availability and service level agreements cannot be affected. Replication mechanisms have already been proposed to target the required service reliability based on VNF redundancy [21].

Service downtime not only negatively effects in user experience but directly translates into revenue loss [35]. Along last decade, several cloud service outages were reported [48]. In February 28, 2017, Amazon reported a outage from 9:37AM to 1:54PM in Simple Storage Service (S3) [9], with clients' estimated losses around US$150 millions. Due to a typo, a large scale services restart was required. A number that can explain such impressive losses: big companies, such as Google, Microsoft, and Amazon, have millions of servers on their data centers [15]. Regarding virtualized values: eBay has 167,000 of VMs [90]. Such huge numbers are persuasive regarding availability. Amazon S3 has three 9's (99.9%) of availability in its Service Level Agreement (SLA), meaning a maximum downtime of 9 hours per year. They spent 4 hours and 17 minutes already with the 2017 February typo issue.

Another downtime source may be generated by continuously execution software, such as those offered by cloud and network providers. The continuous software execution is susceptible to slowly degradation regarding the effective usage of their system resources [59]. Such a phenomenon is called software aging, and it impacts the availability and performance of computer systems. The occurrence of software aging in systems where multiple software components are joined, such as those at cloud computing environments, can be catastrophic. The more software components there are, the greater the risk of failure caused by aging. However, some proactive

action can be triggered to minimize the effects of aging software, known as software rejuvenation. This action is commonly triggered by a time-based, threshold, or prediction-based strategies [13].

Moreover, than software rejuvenation, successful approaches adopted to setup High Availability (HA) include elimination of Single Point Of Failure (SPOF) through redundancy, as well as the interconnection of redundant components in clusters. However, those methods are only suitable for existing infrastructures.

Models can be designed to aid network specialists in the assembling of high available software-centric virtualized networks. Dependability modeling [14, 57, 75, 77] is a largely accepted technique that is concerned about measuring the ability of a system to deliver its intended level of service to users, especially related to failures or others incidents which affect performance. The quantitative fault forecasting [120] adopts models such as Markov Chains and Stochastic Petri Nets to evaluate, in terms of probability, the extent to which some attributes of dependability are satisfied.

This work deals with modeling of VNF chains aiming at aid network specialists to fit availability and capacity requirements, not only in their existent virtualized network but also to estimate these metrics in future virtual infrastructures.

1.2 Objectives

Increase the availability of virtualized infrastructures, such as virtualized data centers and cloud computing environments, and subsequent savings in COPEX and OPEX costs while maintaining user's agreements, are goals of network operators. The main objective of this research is to propose and analyze a set of stochastic models to evaluate and improve virtual network functions considering metrics of availability and capacity. The following list presents the specific objectives that should be accomplished to realize the main objective:

- Create stochastic models to estimate the behavior of availability and capacity metrics of virtual network functions provided in cloud computing infrastructure;
- Assembly and configure a highly available cloud computing infrastructure with support for redundant virtual network functions in order to generate parameter values that will be used as input into stochastic models;
- Adopt the models in case studies aiming at identifying the behavior of the metrics of interest.

One of the main challenges of this research is to propose scalable models considering the high complexity of cloud computing infrastructures and virtual network functions configurations. To cover this problem, we developed hierarchical stochastic models that are not so detailed at a point that forbids its analysis as well as are not so simple that fail in representing the real systems.

We restrict our proposal to private clouds due to configuration complexity of high available cloud computing infrastructures. The assembling of a private cloud offers complete flexibility of configuration and has a low cost.

1.3 Thesis Organization

The remainder of this thesis is organized as follow: Chapter 2 presents the background that is required for understanding the remaining chapters. Chapter 3 shows a series of related works with points in common with this thesis. The support methodology is described in Chapter 4. It explains the methodology for evaluation of VNF chains adopting stochastic modeling. Chapter 5 presents the measurement experiments that were executed in the assembled testbeds during this research. Chapter 6 presents the models that represent the components aimed at providing VNF chains. Chapter 7 presents case studies that were adopted to evaluate the scenarios of interest, as well as to present the models' analysis results. Chapter 8 summarizes the results achieved during this research and also presents suggestions for future works.

2

Background

This chapter discusses the basic concepts of primary areas that set up the focus for this work: network virtualization - including Network Function Virtualization, Software-Defined Networks, and Service Function Chaining - cloud computing, dependability modeling, and software aging and rejuvenation. The background presented here shall provide the necessary knowledge for a clear comprehension of the subsequent chapters.

2.1 Server Virtualization

Server virtualization is the abstraction of applications and operating systems from physical servers. It is required to select a virtualization approach to apply server virtualization. Actually, there are four main accepted approaches that can be applied to implement server virtualization:

i. Full Virtualization: it is the particular kind of virtualization that allows an unmodified guest operating system, with all of its installed softwares, to run in a special environment, on top of existing host operating system. Virtual machines are created by the virtualization software by intercepting access to certain hardware components and certain features. Most of the guest code runs unmodified, directly on the host computer, and in a transparent way: the guest is unaware that it is being virtualized. Virtual Box [100], VMWare Virtualization softwares [122] and [103] are examples of full virtualization products. KVM [67] kernel-level virtualization is a specialized version of full virtualization. The Linux kernel serves as the hypervisor. It is implemented as a loadable kernel module that converts the Linux kernel into a bare-metal hypervisor. As it was designed after the advent of hardware-assisted virtualization, it did not have to implement features that were provided by hardware. So, it requires Intel VT-X or AMD-V (see Hardware Virtualization below) enabled CPUs.

ii. Paravirtualization: this approach requires to modify the guest operating system running in the virtual machine and replace all the privileged instructions with direct calls into the hypervisor. So, the modified guest operating system is aware that is running on a hypervisor and can cooperate with it for improved scheduling and I/O: it includes code to make guest-to-hypervisor transitions more efficient. Paravirtualization does not require virtualization

extensions from the host CPU. Xen hypervisor [129] was the precursor of paravirtualization products.

iii. Operating System virtualization, also known as container-based virtualization, is a lightweight alternative. It presents an operating system environment that is fully or partially isolated from the host operating system, allowing for safe application execution at native speeds. While hypervisor-based virtualization provides an abstraction for full guest OS's (one per virtual machine), container-based virtualization works at the operating system level, providing abstractions directly for the guest processes. OpenVZ [99], LXC[73], and Docker [60] are examples of container-based virtualization solutions.

iv. Hardware Virtualization: it is the hardware support for virtualization. VMs in a hardware virtualization environment can run unmodified operating systems because the hypervisor can use the native hardware support for virtualization to handle privileged and protected operations and hardware access requests; to communicate with and manage the virtual machines [123]. Both Intel and AMD implement hardware virtualization, calling their products as Intel VT-X and AMD-V, respectively.

There are some server virtualization platforms that aid to achieve application availability and fault tolerance. Proxmox VE [110], Citrix XenServer [26], VMWare vSphere [61], and Windows Hyper-V [86] are well-known server virtualization platforms. Some of their features are compared in Table 2.1.

Table 2.1: Server Virtualization Platforms

	Proxmox VE	**VMware vSphere**	**Windows Hyper-V**	**Citrix XenServer**
Guest OS support	Linux and Windows	Linux, Windows,UNIX	Windows and Linux(limited)	Windows and Linux(limited)
Open Source	Yes	No	No	Yes
License	GNU AGPL v3	Proprietary	Free	Proprietary
High Availability	Yes	Yes	Requires MS Failover clustering	Yes
Centralized control	Yes	Yes, but requires dedicated management server or VM	Yes, but requires dedicated management server or VM	Yes
Virtualization	Full and OS	Full	Full and Paravirtualization	Paravirtualization

In this research, we adopted Proxmox VE due to its full operation in Linux systems, its native support for high availability and its compatibility with full and OS virtualization.

2.1.1 Proxmox VE

Proxmox VE is a virtualization environment for servers. It is an open source tool, based on the Debian GNU/Linux distribution, that can manage containers, virtual machines, storage, virtualized networks, and high-availability clustering through both web-based interface or command-line interfaces [110].

Proxmox VE supports OpenVZ container-based virtualization kernel. OpenVZ adds virtualization and isolation, enabling: various containers within a single kernel; resource management, that limits sub-system resources, such as CPU, RAM, and disk access, on a per-container basis; checkpointing, that saves container's state, making container migration possible. OpenVZ guest OSs are instantiated based on templates. These templates are pre-existing images that can create a chrooted environment - the container - on a few seconds, enabling small overhead during creation, execution, and finalization of containers, providing fast deployment scenarios. Programs in a guest container run as rehular applications that directly use the host OS's system call interface and do not need to run on top of an intermediate hypervisor [99].

OpenVZ offers three major networking modes of operation:

- Route-based (`venet`);
- Bridge-based (`veth`);
- Real network device (`eth`) in a container.

The main differences between them are the layer of operation. While route-based mode works in Layer 3, bridge-based works in Layer 2 and real network in Layer 1. In the real network mode, the server system administrator will assign a real network device (such as eth0) into the container. This latter approach will provide the best network performance, but the Network Interface Card (NIC) will not be virtualized.

2.2 Network Virtualization

Network virtualization is a technique that enables multiple isolated logical networks, each with potentially different addressing and forwarding mechanisms, to share the same physical infrastructure [108]. Historically, the term virtual network refers to legacy overlay technologies, such as Virtual Local Area Network (VLAN), a physical method for network virtualization provided in traditional switches. Through a VLAN ID, hosts connected to a switch could be separated in distinct broadcast domains. This approach has several well-known limitations, such as the available number of VLAN IDs (4094). It is not enough to divide multi-tenants VMs.

Jain and Paul [62] performed a detailed explanation about network virtualization required by server virtualization and clouds, exposing the components that must be abstracted to virtualize a network. These components are:

- a NIC, where a computer network starts;

- a Layer 2 (L2) network segments, like Ethernet or WiFi, in which hosts' NICs are connected;
- a set of switches (also called bridges) interconnecting L2 network segments to form an L2 network;
- a Layer 3 (L3) network (IPv4 or IPv6), in which L2 is accommodated as sub-nets;
- routers, in which multiple L3 networks are connected to form the Internet.

Each physical system has at least one L2 physical NIC. If multiple VMs are running on a system, each VM needs its own Virtual Network Interface Card (vNIC). One solution to implement vNICs is through hypervisor software. The hypervisor will not only abstract the computing, memory, and storage, but also implement as many vNICs as there are VMs.

L2 segments virtualization is deployed through overlay. Server virtualization and cloud solutions have been using Virtual eXtensible Local Area Network (VXLAN) [76] to address the need for overlay networks within virtualized data centers. VXLAN overcome several restrictions of VLANs: it enables to accommodate multiple tenants; it runs over the existing networking infrastructure; it provides a means to expand an L2 network, and; it enables location-independent addressing. An alternative protocol is Generic Routing Encapsulation (GRE) [40].

A virtual switch (vSwitch) is the software component that connects virtual machines to virtual networks of L2. L2 switching is typically implemented by means of kernel-level virtual bridges/switches interconnecting a VM's vNIC to a host's physical interface [19]. Many hypervisors running on Linux systems implement the virtual LANs inside the servers using Linux Bridge, the native kernel bridging module. The Linux Bridge basically works as a transparent bridge with MAC learning, providing the same functionality as a standard Ethernet switch in terms of packet forwarding. But such standard behavior is not compatible with SDN and is not flexible enough when aspects such as multitenant traffic isolation, transparent VM mobility, and fine-grained forwarding programmability are critical. The Linux-based bridging alternative is Open vSwitch (OVS), a software switching facility specifically designed for virtualized environments and capable of reaching kernel-level performance.

L3 network virtualization provides addressing (IPv4 or IPv6) for the VMs. Blocks of addresses can be configured and provided for each virtual L3 network. These virtual networks are connected through virtual routers. They replicate in software the functionality of a hardware-based Layer 3 routers.

VM networking had initially been implemented using Linux bridging. Besides its suitable operation and simplicity of configuration and management, it was not originally designed for virtual networking and therefore posed integration and management challenges [23].

2.2.1 Network Function Virtualization

With the exponential increase in bandwidth demand, heavily driven by video, mobile, and Internet of Things (IoT) applications, service providers are constantly looking for ways to expand and scale their network services, preferably without a significant increase in costs [22].

The features of traditional devices are bottlenecks to the expansion of services, because they present several limitations such as: coupling between software system (such as Internetworking Operating Systems (IOS) and management systems) and hardware, loosing flexibility (the ability to adapt to changes); and scalability constraints, because the design of each hardware device is limited to a certain maximum performance requirement.

NFV is a network architecture concept that uses virtualization to implement classes of network functions into building blocks that may connect or create communication services [78].

NFV involves the implementation of network functions (NF) in software that can be moved to, or instantiated in, various locations in the network as required, without the need for installation of new equipment [38].

As specified by European Telecommunications Standards Institute (ETSI) [36], the NFV architecture is composed by three working domains, as depicted in Figure 2.1:

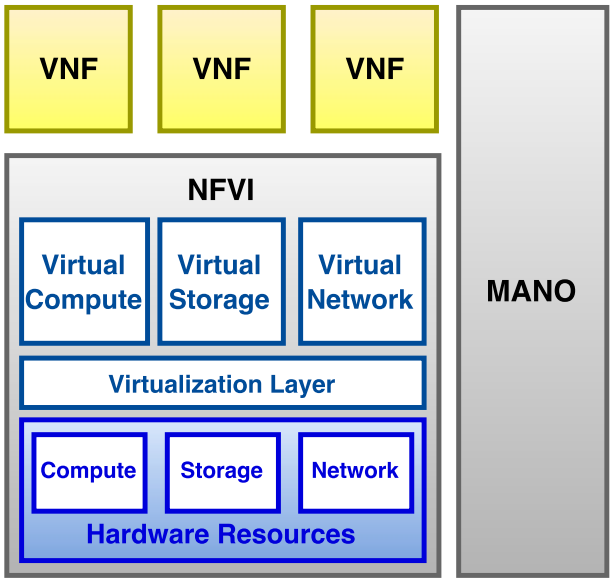


Figure 2.1: High-level NFV Framework

- Virtualized Network Function (VNF): it is the software implementation of the network function which is able to run over NFVI;
- NFV Infrastructure (NFVI): it includes physical resources and how they can be virtualized. It supports the execution of the VNF;
- NFV Management and Orchestration (MANO): it covers the orchestration and lifecycle management of physical or software or both resources that support virtualized infrastructure, as well as the lifecycle management of VNFs.

The NFV framework enables a dynamic construction and management of VNF instances, as well as a relationship between these VNFs, considering several attributes, such as data, control, management, and dependencies. We highlight two relationships among VNFs: (i) VNF chains, in which the connectivity between VNFs is ordered, following routing decision based on policies;

(ii) a collection of VNFs, in which the forward decisions follows traditional routing (based on destination IP).

VNFs chains are the analog of connecting existing physical appliances via cables. Cables are bidirectional and so are most data networking technologies that will be used in virtualized deployments. So, NFV describes a software architecture with VNFs as building blocks to construct VNF Forwarding Graphs (FG) [36] to represent Chains of VNFs. A VNF FG provides the logical connectivity between virtual appliances (i.e., VNFs). An example is depicted in Figure 2.2.

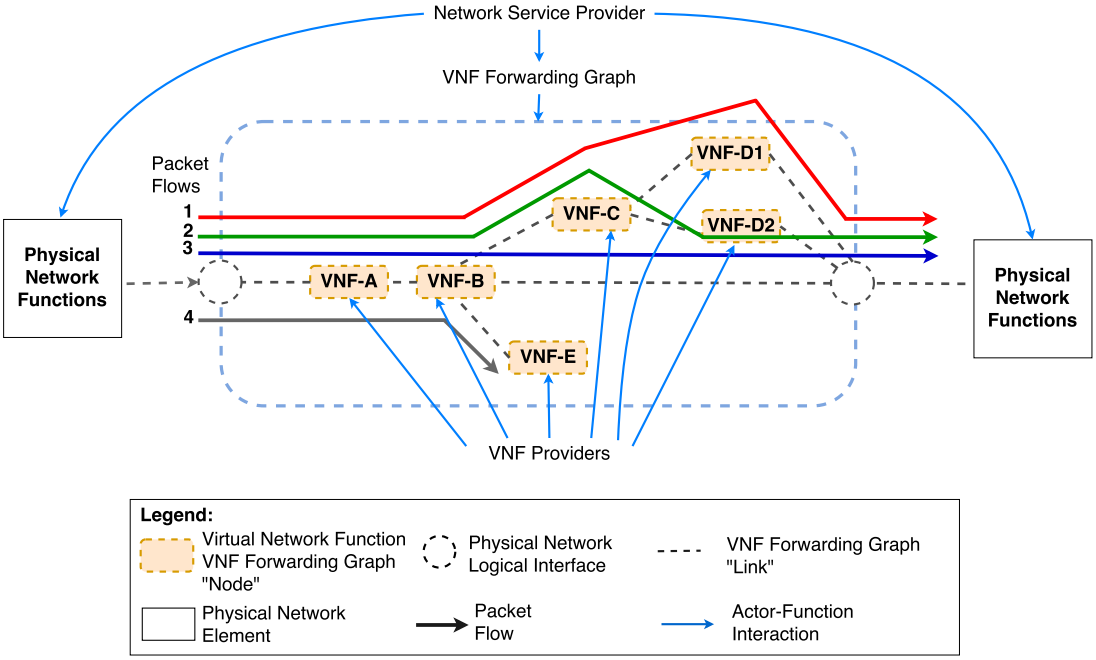


Figure 2.2: VNF Forward Graph representing Chains of VNFs. **Source:** ETSI [36]

A network service provider designed an end-to-end network service between two physical network functions that involve several VNFs (VNF-A, VNF-B, VNF-C, VNF-D1, VNF-D2, VNF-E). This set of VNFs will be combined, forming an ordered chain according to the tenant's requirements. The physical network logical interface at left, represented by a dashed circle, is responsible for performing the classification of distinct tenant's packet flows. Four distinct packet flows, representing different tenant's requirements, are exhibited. According to traffic classification, the tenant's packet flow will be forward through the designed VNF chain. For example, packet flow 1 will be forwarded through VNF-A, VNF-B, VNF-C, and VNF-D1. Observe that service VNF-D is replicated. Some motivations are load balancing and failover. All these functionalities involving VNF chains, flow classification, and traffic steering motivated the creation of Internet Engineering Task Force (IETF) Service Function Chaining Working Group.

2.2.2 Service Function Chaining

The term Service Function Chaining (SFC) is used to describe the definition and instantiation of an ordered list of instances of virtual network service functions, and the subsequent steering of traffic flows through those service functions [104]. Fundamentally, SFC routes packets through one or more service functions instead of conventional routing that routes packets using the destination IP address.

The emergence of SFC is aimed to address three main functionalities:

- service overlay: SFCs adopts the decoupling of services to the physical topology. It allows operators to use whatever overlay or underlay they prefer to create a path between service functions and to locate service functions in the network as needed;
- service classification: it is used to identify which traffic will be steered through an SFC overlay;
- SFC encapsulation: it enables the creation of a service chain in the data plane and also carries data-plane metadata to enable the exchange of information between logical classification points and service functions.

The combination of VNF chains and a flow classifier creates a service function chain. A flow classifier is a component that matches traffic flows against policies for subsequent application of the required set of network service functions, whereas a service function chain defines an ordered set of abstract service functions and ordering constraints that must be applied to packets and/or frames and/or flows selected as a result of classification.

As presented by Luis et al. [72], flow classification, as well as its monitoring, can make networking more dependable, motivating the investigation of dependability metrics.

2.2.3 Software-Defined Networking

Software-Defined Networking (SDN) is a network architecture in which network control (called control-plane) is decoupled from forwarding control (called data-plane) and is directly programmable [96]. It emerged as an approach to network management conducted in Stanford University[81].

Its well-known dissociation between control plane (where routing decisions are built) and data plane (responsible for reception and transmission of packets) enables to centralize the management of several network devices. The network intelligence is logically centralized inside controllers. The software-based SDN controllers perform the administration of the network using high-level policies. The controllers build flow tables with the aim of forwarding packages to connected VMs.

The separation of the control plane and the data plane can be realized employing a well-defined programming interface between the switches and the SDN controller. The controller exercises direct control over the state in the data plane elements via this well-defined API. The

most notable example of such an API is OpenFlow [45] whereas the most adopted implementation of OpenFlow is Open vSwitch [43], functioning as a vSwitch in VM environments. The Open vSwitch enables Linux to become part of a SDN architecture. In this research, we adopted Open vSwitch as the underlying connection technology of VNF chains.

Regarding NFV, SDN enables to separate the network functions from expensive appliances. It also enables to implement the separation between VNFs and the underlying physical network. So, it has a central role in the network virtualization and all its previously discussed benefits. SDN, as an enabler of network virtualization, can expand the services provided by cloud infrastructures and offer an even higher level of innovation. It can allocate networking resources dynamically and proficiently as dictated by the demands of VM clients.

Among the benefits of SDN, we highlight increasing in network availability as a result of centralized and automated management of network devices, decreasing manual configuration errors.

2.3 Cloud Computing

Cloud computing is the on-demand delivery of computing power, database storage, applications, and other IT resources through a cloud services platform via the Internet with pay-as-you-go pricing [10].

A set of 5 attributes, depicted in Figure 2.3, was defined by NIST to define cloud computing in a more instructive way.

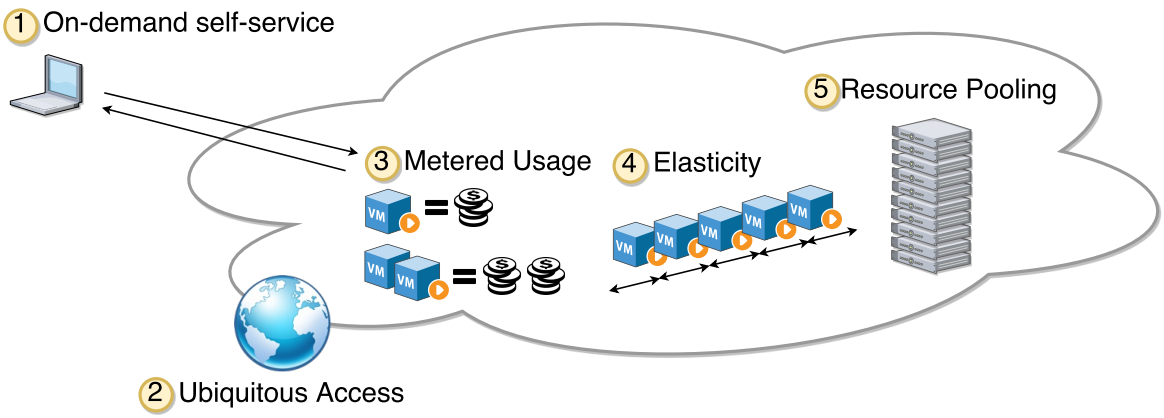


Figure 2.3: Cloud Computing Features

1. On-demand self-service: a client can provision computing resources from the cloud without having to interact with IT or service provider personnel. It is possible to rent virtualized resources online.
2. Ubiquitous access: it is possible to use standard platforms that provide simple connectivity without having to put down dedicated cables or systems and without having to buy custom hardware for access.

3. Measured service: resource usage can be monitored, controlled, and reported, providing transparency for both the provider and consumer of the utilized service. It enables a pay-per-use model.
4. Rapid elasticity: Capabilities can be elastically provisioned and released, in some cases automatically, to scale rapidly outward and inward commensurate with demand.
5. Resource pooling: the cloud provider has different types of grouped resources, including storage, memory, computer processing, and bandwidth (to name a few), which instead of being dedicated to anyone client are allocated to clients as they need them.

There are different ways that providers can offer resources as services within a cloud to their clients.

1. Infrastructure as a Service (IaaS): it refers to on-demand provisioning of infrastructural hardware resources.
2. Platform as a Service (PaaS): it refers to providing platform layer resources, including operating system support and software development frameworks.
3. Software as a Service (SaaS): it refers to providing on-demand applications over the Internet.

With clear attributes that define what cloud services are and the types of services that the cloud could provide, it makes sense to look at how cloud services are deployed. Clouds have four basic deployment models:

1. Public cloud: The cloud infrastructure is provisioned for open use by the general public.
2. Private cloud: it is created when an enterprise data center is reconfigured in such a way as to provide the five cloud attributes. The enterprise owns the cloud. It is both the provider and the client.
3. Community cloud: the cloud infrastructure is provisioned for exclusive use by a specific community of consumers from organizations that may own, manage, and operate the cloud resources.
4. Hybrid cloud: it is a composition of two or more distinct cloud infrastructures (private, community, or public) through a single management interface.

There are several opensource cloud platforms that allow to provide private IaaS approach. Openstack [98], CloudStack [28], and OpenNebula [97] are the most adopted. In this research, we have been using the openstack IaaS cloud platform due to its natural relationship with NFV [44].

2.3.1 Openstack

Openstack is a cloud management system that controls pools of computing, storage, and networking resources. It is exposed to the cloud end users as HTTP(s) APIs that provide its independent parts called openstack services. Openstack cloud logical architecture is depicted in Figure 2.4:

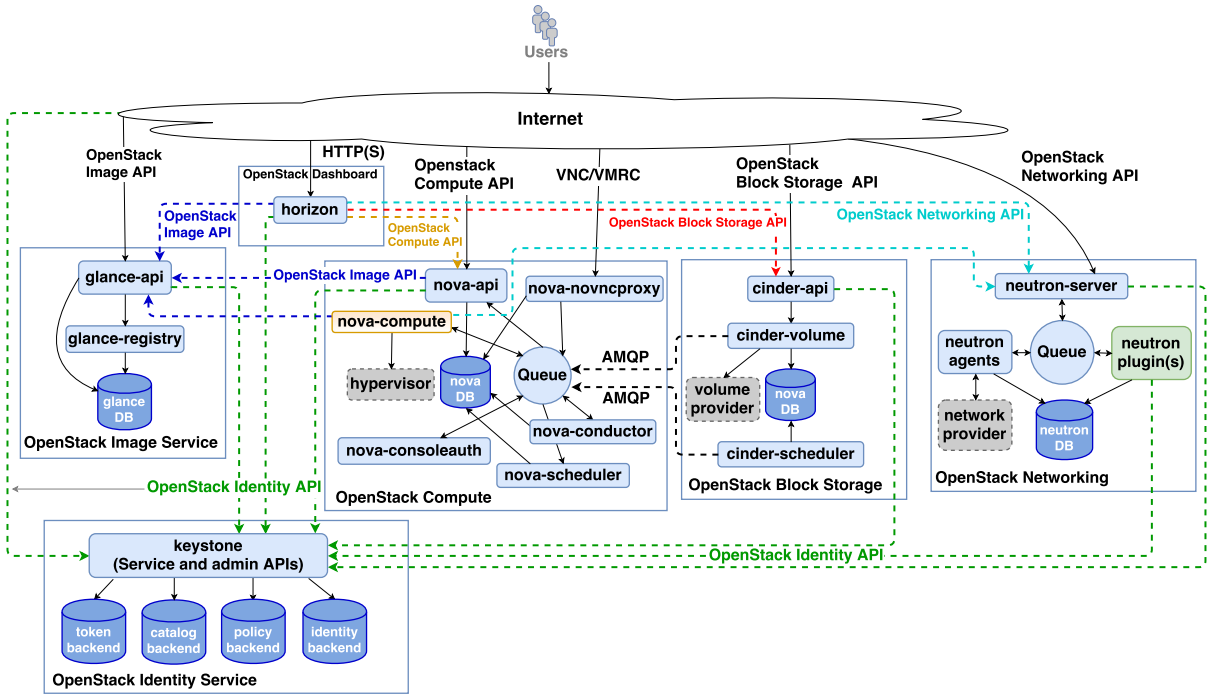


Figure 2.4: Openstack Architecture

Openstack Dashboard, Image Service, Compute, Block Storage, Networking, and Identity Service are core projects of openstack cloud. In Figure 2.4, they are represented by the external rectangles. Each openstack project releases components, represented by internal rounded rectangles. These are the daemons (background processes) that are executed to provide the associate functionality of each component. Internal solid arrows represent the relationship among daemons that form a component, whereas dashed arrows represent the relationship between the daemons of distinct projects. Openstack components are explained below.

- **Compute**: known as nova, it provides a way to provision Compute instances. It supports the creation of the virtual machines through the execution of a set of daemons in Linux or UNIX servers. Compute services manage and automate pools of CPU and RAM resources. It is split into five daemons:
  - **nova-api**: it accepts and responds to end user compute API calls;
  - **nova-scheduler**: it picks a compute node to run a VM instance;
  - **nova-conductor**: it provides coordination and database query support for nova;
  - **nova-consoleauth**: it provides authentication for nova consoles;
  - **nova-novncproxy**: it provides a proxy for accessing running instances through a VNC connection;
  - **nova-compute**: is responsible for building a disk image, launching it via the underlying virtualization driver, responding to calls to check its state, attaching persistent storage, and terminating it.

- **Networking**: known as neutron, the standalone openstack Networking provides cloud operators and users with an API to create and manage networks in the cloud. Indeed, openstack Neutron is a SDN project focused on delivering Network as a Service (NaaS) in virtual computing environments. It does so by deploying several network processes across many nodes. Such processes provide resources used by the VM network interface, including IP addressing and routing. Its main daemon is:
  - **neutron-server**: it relays user requests to the configured layer 2 plug-in (responsible for interacting with the underlying infrastructure so that the traffic can be routed). Linux Bridge and Open vSwitch are two examples of L2 plug-ins.

  Besides neutron-server, openstack Networking, also includes two agents:
  - **neutron-dhcp-agent**: it provides DHCP services to tenant networks;
  - **neutron-l3-agent**: it does L3/Network Address Translation (NAT) forwarding to enable external network access for VMs on tenant networks.
- **Block Storage**: provides the traditional disk block-level storage for VMs. It is composed of the following daemons:
  - **cinder-api**: it authenticates and routes requests throughout the Block Storage service;
  - **cinder-volume**: manages Block Storage devices, specifically the backend devices themselves;
  - **cinder-schedule**: schedules and routes requests to the appropriate volume. By default, it uses round-robin, but can use more sophisticated policies based capacity or volume type, deciding which cinder-volume node will be used.
- **Dashboard**: it provides a web-based user interface to the various openstack components. It includes both end user area for tenants to manage their virtual infrastructure and administration area for cloud operators to manage the openstack environment. Horizon service runs as a Web Server Gateway Interface (WSGI) application, hosted by Apache Web Server.
- **Identity Service**: it provides identity and access management for all the openstack components. It dells with all authentication and authorization transactions aimed at using cloud resources. Users have credentials they can use to authenticate, and they can be a member of one or more groups. Any API call results in an authentication interaction with this service. Similar to the horizon, identity service, called keystone, runs as a WSGI application and is also hosted by apache. Each project contains a component that establishes communication with Identity Service, as can be evinced by the green dashed lines, labeled as Opentack Identity API in Figure 2.4.

- **Image Service**: known as glance, it includes registering, discovering, and retrieving of virtual machine images. It allows querying of VM image metadata as well as retrieval of the actual image. VM images made available through glance can be stored in a variety of locations from simple filesystems to disk-block storage systems like openstack cinder. Glance is split into two daemons:
  - **glance-api**: interacts with users requesting VM images;
  - **glance-registry**: connects to the database backend aiming to store, process, and retrieves images metadata, such as size and type.

Figure 2.4 also depicts openstack shared services, adopted by openstack components to support the provision of their functionalities:

- relational database: each of the previously mentioned components has its own database, represented by traditional cylinder symbol. Each service requires the creation of tables to store its data. Most traditional used databases in openstack are MySQL, MariaDB, and PostgreSQL. For HA, Galera Cluster has been used;
- a message queue is used for communication among all openstack daemons through Advanced Message Queuing Protocol (AMQP). It coordinates operations and status information among services. Most traditional message queue services supported by openstack include RabbitMQ, Qpid, and ZeroMQ. Message queues are represented by circles in Figure 2.4. One example of AMQP is exhibited by black dashed lines, labeled as AMQP, between cinder daemons and openstack Compute;
- a cache system, used to speed up dynamic database-driven openstack services by caching data in RAM. It reduces the number of times that an external data source must be read. Keystone and Identity are examples of openstack components that use a cache system. Memcached is the default cache system adopted by openstack.

The openstack platform defines a nomenclature formality that associate services to servers. They are called deployment modes and are described below:

- **Controller node**: it is the control plane for the openstack environment, running Identity service for access control, Image service for virtual machines image provision, the management portions of nova and neutron services, and the Dashboard. Moreover, it also includes support for the shared services: SQL database, message queue, and cache system. Finally, it also executes Network Time Protocol (NTP) daemon for clock synchronization. All the components and shared services with a blue background in Figure 2.4 are executed in Controller nodes;
- **Compute node**: it runs the hypervisor portion of nova that operates tenant virtual machine instances. Besides hypervisor, the compute node also runs the networking plug-in and an

agent that connect tenant networks to instances and provide firewalling (security groups) services;

- **Neutron node** it runs networking services for L3, metadata, DHCP, and Open vSwitch. The network node handles all networking between other nodes as well as tenant networking and routing. It also provides floating IPs that allow instances to connect to public networks.

Figure 2.5 depicts one instance for each deployment mode. At the bottom, the names of each role are exhibited in bold. We also insert the hardware (HW), the storage(S), and operating system (OS) at Figure 2.5 to enable a complete view of each openstack deployment mode.

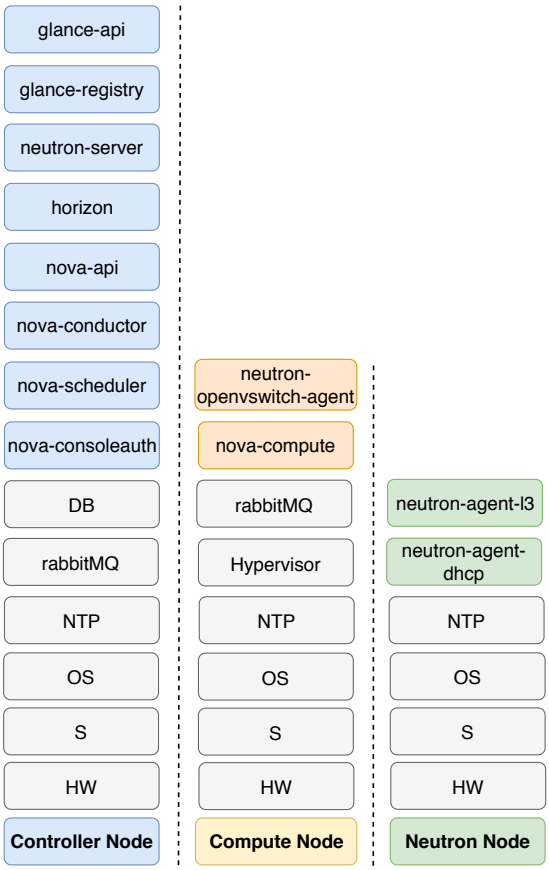


Figure 2.5: Openstack Deployment Modes

Besides the provision of deployment modes in isolated physical servers, it is also possible to group deployment modes in one physical server:

- **Controller/Neutron**: the controller and neutron services are grouped in one single server;
- **All-In-One**: as suggested by its name, in this configuration, all the openstack services are provided in the same physical server.

Figure 2.6 depicts one instance of All-In-One openstack node.

The deployment modes are highlighted in this work because the adopted assembling is relevant for the studied metrics of interest.

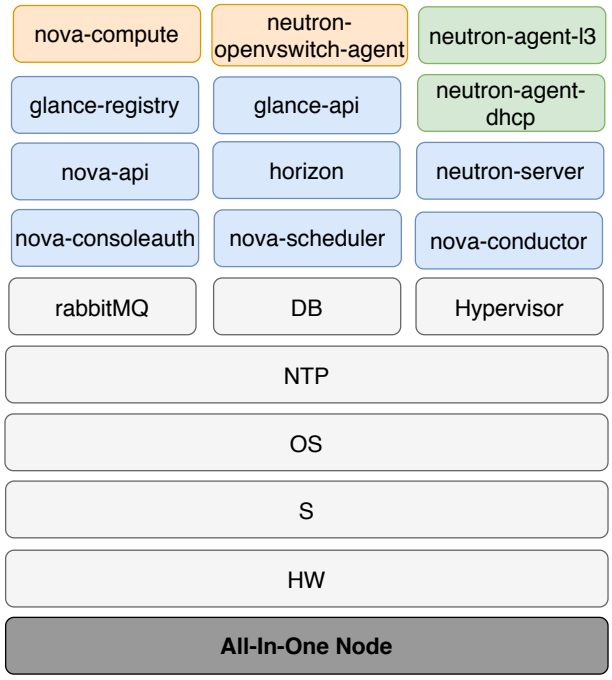


Figure 2.6: Openstack All-In-One Deployment Mode

Designing an openstack cloud requires an understanding of its user's requirements to state the best fit configuration. A specific objective of this research is to assembly and configure a High Available cloud infrastructure. Knowing the openstack architecture is a requirement to provide all its services over a hardware infrastructure without SPOFs. The openstack modular architecture enables to replicate its components and execute them on several physical servers. For HA openstack cloud, each service must be clustered to ensure its high availability.

2.3.2 The Openstack SFC API

The Openstack SFC API [132] is formed by two parts: the flow classifier and the port chain, as depicted in Figure 2.7. The flow classifier specifies the classification rules that are used to state which flows will go through the specific chains. The port chain consists of an ordered sequence of service functions that a particular flow will go through. A neutron port (a logical connection of a virtual network interface to a virtual network) receives the ingress flow and another neutron port forwards the egress flow. The SFC API calls these two neutron ports as port pair. Each port pair is attached to a particular VM providing the associated virtualized service. Port chains may be provided in a redundant configuration, in which combined port pairs compose a port pair groups.

Our implemented studied scenario, that is used to exemplify the adoption of SFC API, depicted in Figure 2.7, is composed by a port chain with three port pair groups: Firewall, Load Balancer, and Cache. For 3N redundancy, each port pair group contains 3 VNF instances. We opt for these services due to our previous experience with firewalls, load balancer, and cache implementations.

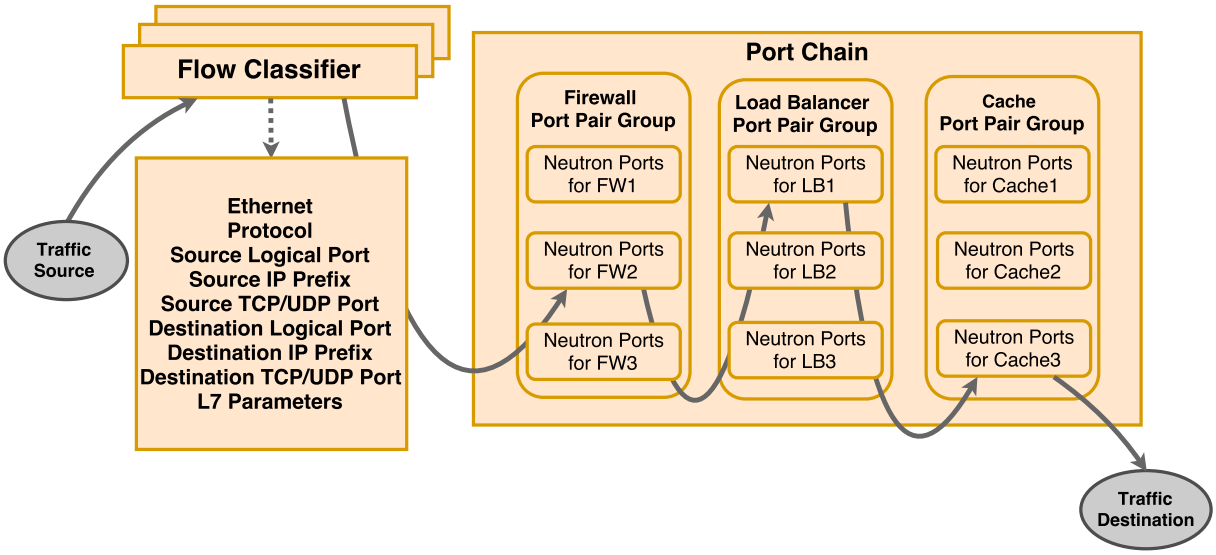


Figure 2.7: SFC Graph

Figure 2.7 represents a classification process that directs the traffic through FW2, LB1, and Cache3 neutron ports. To provide redundancy and high availability, VNFs are executed as VMs in Pacemaker clusters, and the traffic flows can be classified and steering in a load-balanced way by port pair groups containing these NFV VMs.

2.4 Clusters and High Availability

A cluster is any ensemble of independently operational elements integrated by some medium for coordinated and cooperative behavior [126]. Consistent with this broad interpretation, computer clusters are ensembles of independently operational computers integrated through an interconnection network. They support user-accessible software for organizing and controlling concurrent computing tasks that may cooperate on processing a typical application program or workload.

Motivations for the adoption of server clusters include: high availability, load balancing, parallel processing, systems management and scalability. Following the motivations for clusters adoption, we can classify clusters in three types [101, 124]: high performance, load balancing, and high availability.

High-performance clusters are used in environments with intensive and heavy computing requirements. Large rendering tasks, as well as scientific computing, are examples of high-performance clusters adoption. In these kinds of processing, all nodes of the cluster should be active, providing as much processing capacity as possible.

Heavy demand environments typically adopt load balancing clustering. As depicted in Figure 2.8, users' requests are distributed to multiple machines in a server farm to optimize the response time of performed requests.

High availability (HA) clusters are adopted to make sure that critical resources reach the maximum possible availability [124]. In HA clusters, the elimination of SPOFs is a strong

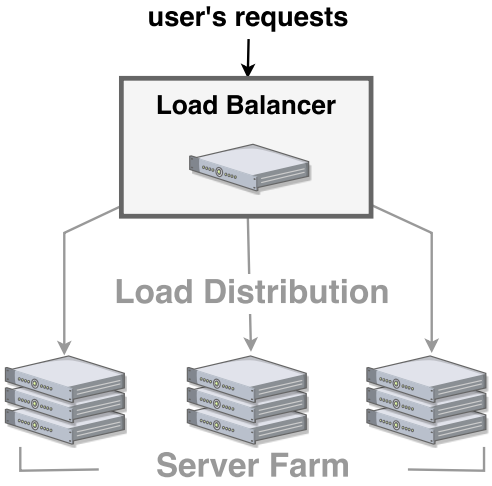


Figure 2.8: Load Balancing Cluster

(or even a mandatory) requirement. Figure 2.9 exhibits a HA cluster. An HA cluster software must be installed to monitors both cluster nodes availability as well as the availability of services provided by the cluster's nodes.

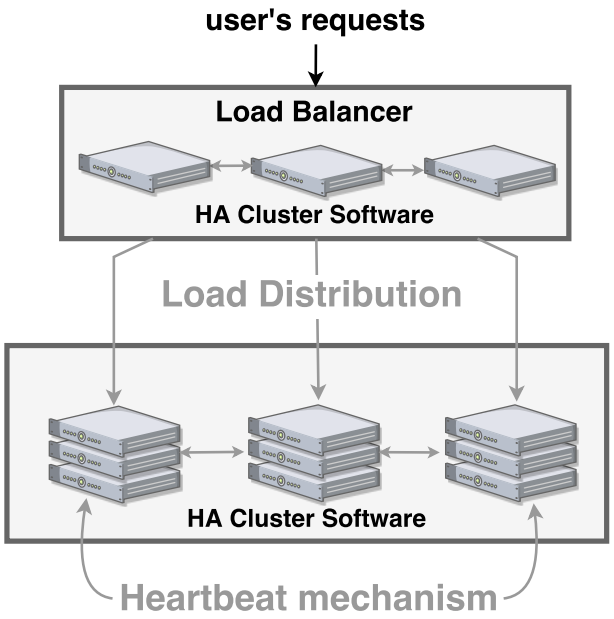


Figure 2.9: High Available Load Balanced Cluster

As can be observed in Figure 2.9, the load balancer was also replicated with the purpose of eliminate SPOFs. Two-way arrows lines represent the monitoring operation of HA cluster software, usually called as heartbeat mechanism.

High availability is established by the usage of failover softwares combination [16], such as Pacemaker [68], Corosync [94], and HAProxy [114], to cite a few. By their joint action, they deploy load balanced HA clusters enabling automatic recovery of services.

Pacemaker is a cluster resource manager. A resource is a service made highly available

by a cluster. So, Pacemaker is responsible for the lifecycle of deployed resources. It achieves HA for cluster services by detecting and recovering failures from resource-level. Pacemaker adopts Corosync for detection and recovering of failures from node-level. Corosync provides messaging and membership services for cluster's nodes.

Figure 2.10 exhibits Pacemaker and Corosync roles in a layer-based view: Pacemaker probes the resources whilst Corosync monitors physical nodes.

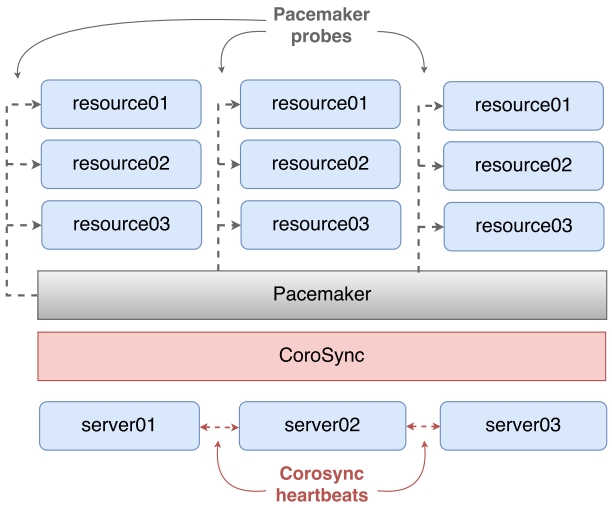


Figure 2.10: Pacemaker and Corosync

Every resource has a Resource Agent (RA). They are external programs that abstract the service in the provision, presenting a consistent view of the cluster. The cluster does not need to understand how the resource works because it relies on the RA to perform a start, stop, or monitor commands over the resource.

2.4.1 Redundancy Strategies

Failover softwares protect computer systems from failure adopting standby equipment(s) that automatically takes over when the main system fails. During a failover, the cluster software will move the required set of resources, such as filesystems, databases, and IP addresses to the standby node(s).

Without a redundancy strategy, the system works in a simplex scheme, i.e., there is at least one SPOF component. Any failure in such components will inevitably result in longer system downtimes. With redundancy strategies, system components are arranged in a pool, resulting in higher availability. The following redundancy strategies are usually adopted:

- Active-Active: two units are deployed in a fully operational capacity, sharing workload during normal operation.
- Active-Standby: two units are deployed, however one active unit will provide service for all the traffic intended for the system. A standby unit, also known as secondary unit, can be maintained at varying degrees of readiness to restore service, which may be classified as:

- Hot Standby: the secondary unit is on, and its data are maintained in sync with an active unit;
- Warm Standby: the secondary unit is powered on, but is not receiving the workload;
- Cold Standby: the secondary unit is powered off, and the activation time is larger than the warm approach.

Pacemaker can create any of the previous redundancy strategies.

2.5 Dependability

Due to ubiquitous provision and access to services on the Internet, dependability has become an attribute of prime concern in computer systems development, deployment, and operation [75]. The main concern of dependability is to deliver services that can be justifiably trusted. Laprie [69] provides a conceptual framework for expressing the attributes of what constitutes dependable and reliable computing. Such a framework has been widely adopted by academy and industry to guide research and practical works regarding systems' trustability. Because cloud computing is a large-scale distributed computing paradigm, and its applications are accessible anywhere, anytime, cloud dependability is becoming more important and more difficult to achieve [112].

In Laprie's work, a set of basic definitions and associated terminology regarding dependability concepts were presented. They are presented in Figure 2.11 and mentioned below.

- The impairments encompass faults, errors, and failures. A system fault represents the event that occurs when service delivery does not happen as planned. A failure is a deviation of system behavior from its specification. The cause of a failure is an error. An error is the portion of the state of the system responsible for driving it to failure. Each error thus is caused by the activation of a failure.
- The means are the methods, tools, and solutions enabling to provide the ability to deliver a service with the desired specification. A system is fault tolerant when it does not fail even when there are faulty components;
- The attributes make it possible to obtain quantitative measures, which are often crucial for the analysis of the provided services.

Computer systems analysts have been adopting dependability with the aim of compute and evaluate systems' dependability metrics, such as reliability and availability. Reliability can be defined as the continuity of correct service, whereas availability is the readiness for correct service. Each of the methods to compute these two metrics are discussed below.

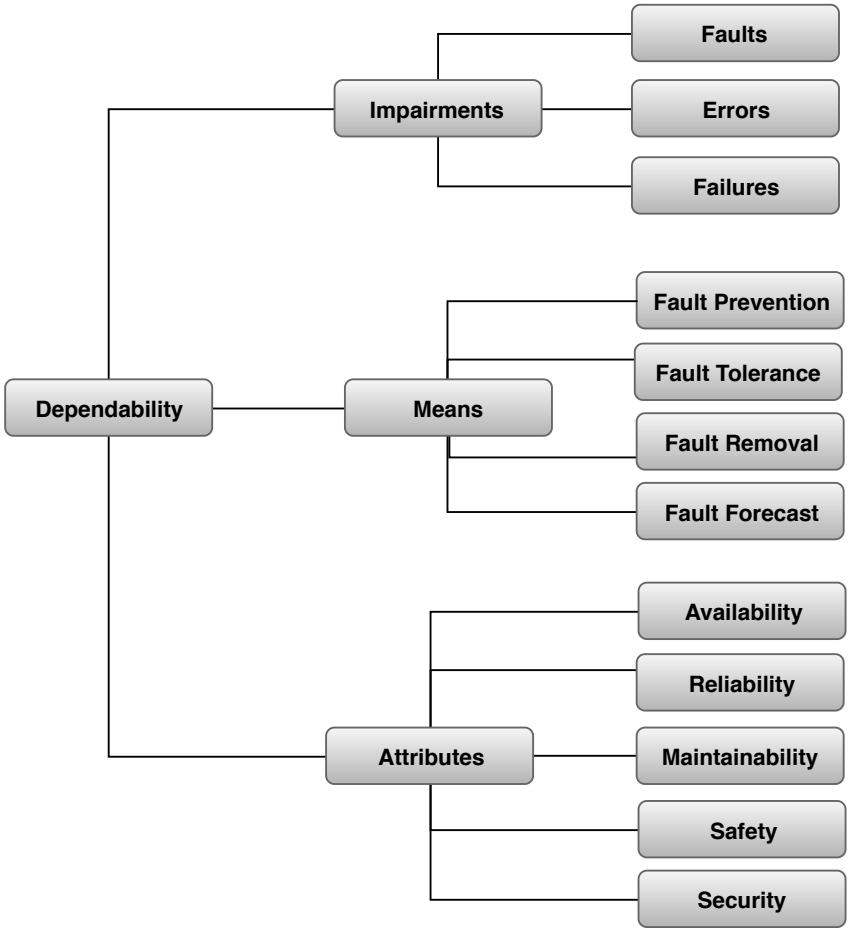


Figure 2.11: Dependability Taxonomy

2.5.1 Reliability

The reliability metric measures the probability that a system will deliver its function correctly during a specified time interval, without the occurrence of failures in this interval. So, the reliability does not consider the system repair and consider the working of analyzed system uninterruptedly.

The Equation

$$R(t) = P\{T > t\} = e^{-\int_0^t \lambda(t)dt}, t \geq 0 \tag{2.1}$$

provides the reliability of a system, where T is a random variable that represents the time to failure of the system and $\lambda(t)$ is known as the Hazard function. So, T represents the required time interval to reach the state $X(t) = 0$, starting from state $X(t) = 1$, i.e., the time to failure (TTF) of a system.

2.5.2 Availability

The availability is the probability of a system being in operation at a desired time, or during an expected period of time, considering failures and/or repairs within that interval.

A system failure occurs when the system does not provide its specified functionality. A system failure can be defined as the failure of a system component, a system sub-system, or another system that interacts with the considered one. With the aim to model the time to failure of a system, consider an indicator random variable $X(t)$ that represents the system state at time t. $X(t) = 1$ represents the operational state and $X(t) = 0$ represents the faulty state. A random variable T represents the time to reach the state $X(t) = 0$, given that the system started in state $X(t) = 1$ represents the time to failure of the system, and its cumulative distribution function $F_T(t)$ and the respective probability density function $f_T(t)$ are defined as:

$$F_T(0) = 0 \quad and \quad \lim_{t\to\infty} F_T(t) = 1, \tag{2.2}$$

$$f_T(t) \geq 0 \quad and \quad \int_0^\infty f_T(t)dt = 1. \tag{2.3}$$

So, the time to failure is a non-negative continuous random variable.

The simplest definition of Availability is expressed as the ratio of the expected system uptime to the expected system up and downtimes:

$$A = \frac{E[Uptime]}{E[Uptime] + E[Downtime]}. \tag{2.4}$$

The availability metric can be defined as

$$A = \frac{MTTF}{MTTF + MTTR}, \tag{2.5}$$

where,

$$MTTF = \int_0^\infty R(t)dt, \tag{2.6}$$

and

$$MTTR = MTTF \times \frac{UA}{A}. \tag{2.7}$$

The UA represents the system unavailability (Equation (2.8))

$$UA = 1 - A. \tag{2.8}$$

The instantaneous availability is the probability that the system is operational at a specific time instant t, that is,

$$A(t) = P\{X(t) = 1\} = E\{X(t)\}, t \geq 0. \tag{2.9}$$

If the system approaches stationary states as the time increases, it is possible to quantify the steady-state availability and estimate the long-term fraction of time the system is available.

$$A = \lim_{t\to\infty} A(t), t \geq 0. \tag{2.10}$$

Availability can be expressed using the number of nines that represents the probability of service readiness, as shown in Table 2.2 [115]. For instance, a system with five 9's of availability is classified as high availability, meaning an annual downtime of nearly 5 minutes.

Table 2.2: Service availability and downtime ratings

Number of 9's	Service Availability (%)	System Type	Practical Meaning
1	90	Unmanaged	Down 5 weeks per year
2	99	Managed	Down 4 days per year
3	99.9	Well managed	Down 9 hours per year
4	99.99	Fault tolerant	Down 1 hour per year
5	99.999	High availability	Down 5 minutes per year
6	99.9999	Very high availability	Down 30 seconds per year
7	99.99999	Ultra availability	Down 3 seconds per year

2.5.3 Maintainability

The maintainability is the probability that a failed system will be restored to operational effectiveness within a given period of time when the repair action is performed in accordance with prescribed procedures.

Consider a a continuous time random variable $Y_S(t)$ that represents the system state. $Y_S(t) = 0$ when S is failed, and $Y_S(t) = 1$ when S has been repaired. Consider yet the random variable D that represents the time to reach the state $Y_S(t) = 1$, given that the system started in state $Y_S(t) = 0$ at time $t = 0$. The random variable D represents the system time to repair, $F_D(t)$ its cumulative distribution function, and $f_D(t)$ the respective density function, where:

$$F_D(0) = 0 \quad and \quad \lim_{t\to\infty} F_D(t) = 1, \tag{2.11}$$

and

$$f_D(t) \geq 0 \quad and \quad \int_0^\infty f_D(t)dt = 1, \tag{2.12}$$

Quantitatively, the maintainability M(t) can be computed as the probability that the system S will be repaired by t considering a specified resource.

$$\begin{gathered} P\{D \leqslant t\} = F_D(t) = \int_0^t f_D(t)dt \\ M(t) = 1 - F_D(t) \end{gathered} \tag{2.13}$$

Then, the MTTR can be defined as:

$$MTTR = \int_0^\infty t \times M(t)dt = \int_0^\infty (1 - F_D(t))dt \tag{2.14}$$

2.6 Dependability Modeling

Dependability models can be broadly classified into combinatorial models and state-space models. In combinatorial models, it is assumed that the failure or recovery (or any other behaviors) of a system component is not affected by the behavior of any other component. i.e., system components are independents. Reliability Block Diagram (RBD) and Fault Trees are examples of combinatorial models.

Analysts may adopt state-space models to represent more complex interactions between system components, such as active-standby redundancy. Markov chains and Stochastic Petri Net (SPN) are examples of state-space models.

2.6.1 Reliability Block Diagrams

RBDs are models represented by a source and a target vertex, a set of blocks, each representing a system component, and arcs connecting the components and the vertices, as depicted in Figure 2.12. Besides its graphical representation, RBDs are used to analyze systems and assess their availability and/or reliability through equations. The blocks represent the groups of components or the smallest entities of the system, which are not further divided. The blocks are usually arranged using composition mechanisms: series, parallel, bridge, k-out-of-n blocks, or a combination of previous compositions [20]. The RBDs adopted in this research are series-parallel structures only. If the individual components of a system are connected in series, the failure of any component causes the system to fail. If the individual components of a system are connected in parallel, the failures of all components cause the system to fail. For series and parallel structures, the steady-state availability is given respectively by:

$$A_S = \prod_{i=1}^{n} A_i \tag{2.15}$$

and

$$A_P = 1 - \prod_{i=1}^{n} (1 - A_i), \tag{2.16}$$

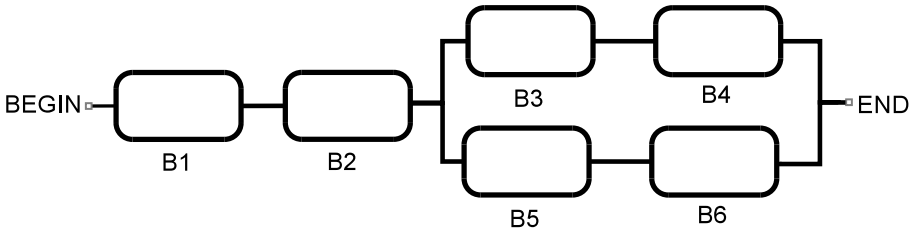


Figure 2.12: Reliability Block Diagram

2.6.2 Continuous Time Markov Chains

First introduced by Andrei Andreevich Markov in 1907, Markov chains have been in use intensively in dependability modeling and analysis since around the fifties [75]. Aiming to understand Markov chains, we should start by the comprehension of a stochastic process.

A stochastic process is a family of random variables X(t) defined on a sample space. The values assumed by X(t) are called states, and the set of all the possible states is the state space, I. The state space of a stochastic process is either discrete or continuous. If it is discrete, the stochastic process is called a chain.

A stochastic process can be classified by the dependence of its state at a particular time on the states at previous times. If the state of a stochastic process depends only on the immediately preceding state, we have a Markov process. So, a Markov process is a stochastic process whose dynamic behavior is such that probability distributions for its future development depend only on the present state and not on how the process arrived in that state. It means that at the time of a transition, the entire past history is summarized by the current state.

If we assume that the state space, I, is discrete (finite or countably infinite), then the Markov process is known as a Markov chain (or discrete-state Markov process). If we further assume that the parameter space t, is also discrete, then we have a discrete-time Markov chain (DTMC). If the parameter space is continuous, then we have a continuous-time Markov chain (CTMC).

Markov chains can be represented as a directed graph, with the nodes representing the system's states and the edges representing changes in the system's state. There are labels in transitions, indicating the probability or rate at which such transitions occur.

When dealing with Continuous Time Markov Chain (CTMC), such as the availability model of Figure 2.13, transitions occur with a rate, instead of a probability, due to the continuous nature of this kind of model. The CTMC is represented through its transition matrix, often referenced as the infinitesimal generator matrix.

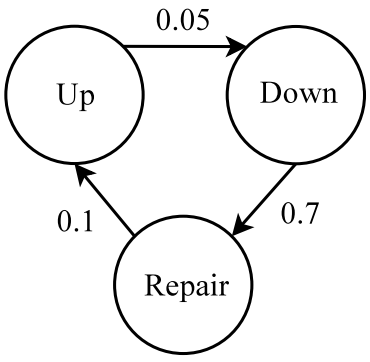


Figure 2.13: CTMC: availability model

Considering the CTMC availability model of Figure 2.13, the rates are measured in failures per second, repairs per second, and detections per second. The generator matrix Q is

composed by components q_{ii} and q_{ij}, where $i \neq j$ and $\sum q_{ij} = -q_{ii}$. Considering a state-space S = Up, Down, Repair = 0, 1, 2 the Q matrix is:

$$Q = \begin{pmatrix} q_{00} & q_{01} & q_{02} \\ q_{10} & q_{11} & q_{12} \\ q_{20} & q_{21} & q_{22} \end{pmatrix} = \begin{pmatrix} -0.05 & 0.05 & 0 \\ 0 & -0.7 & 0.7 \\ 0.1 & 0 & -0.1 \end{pmatrix} \tag{2.17}$$

Equation (2.18) and the system of Equations (2.19 describe the computation of the state probability vector, respectively for transient (i.e., time-dependent) analysis, and steady-state (i.e., stationary) analysis. From the state probability vector, nearly all other metrics can be derived, depending on the system that is represented.

$$\pi'(t) = \pi(t)Q, \qquad given\ \pi(0) \tag{2.18}$$

$$\pi Q = 0, \sum_{i \in S} \pi_i = 1 \tag{2.19}$$

Detailed explanations about how to obtain these Equations may be found in [17].

For all kinds of analysis using Markov chains, an important aspect must be kept in mind: the exponential distribution of transition rates. The behavior of events in many computer systems may be fit better by other probability distributions, but in some of these situations, the exponential distribution is considered an acceptable approximation, enabling the use of Markov models. It is also possible to adapt transition in Markov chains to represent other distributions by means of phase approximation, as shown in [117]. The use of such technique allows the modeling of events described by distributions such as Weibull, Erlang, Cox, hypoexponential, and hyperexponential.

2.6.3 Stochastic Petri Nets

Petri Nets [88] are a family of formalisms very well suited for modeling discrete event systems due to their capability for representing concurrency, synchronization, and communication mechanisms, as well as deterministic and probabilistic delays. Time (stochastic delays) and probabilistic choices are often used in dependability evaluation models [74, 75, 117, 119]. The original Petri Net does not have the notion of time for analysis of performance and dependability. The introduction of a duration of events results in a timed Petri Net. A special case of timed Petri Nets is the SPN, where the delays of activities (represented as transitions) are considered random variables with exponential distribution. An SPN can be translated to a CTMC, which may then be solved to get the desired dependability or performance results. This is especially useful because building a Markov model manually may be tedious and error-prone, especially when the number of states becomes very large. Marsal et al. [5] proposed extensions to SPN that considers two types of transitions: timed and immediate. The transition firing times in

SPNs correspond to the exponential distribution. The exponentially distributed firing times are associated only with timed transitions, since immediate transitions, by definition, fire in zero time.

We adopted the formal SPN definition (according to German [49]), presented below, composed by the 9-tuple SPN = (P,T,I,O,H, Π,G, M_0, Atts), in which:

- $P = \{p_1, p_2, ..., p_n\}$ is the place set, where n is the places number;
- $T = \{t_1, t_2, ..., t_m\}$ is the immediate and timed transitions set, $P \cap T = \emptyset$, where m is the transitions number;
- $I \in (\mathbb{N}^n \to \mathbb{N})^{n \times m}$ is the matrix representing the input arcs (may be marking dependent);
- $O \in (\mathbb{N}^n \to \mathbb{N})^{n \times m}$ is the matrix representing the output arcs (may be marking dependent);
- $H \in (\mathbb{N}^n \to \mathbb{N})^{n \times m}$ is the matrix representing the inhibitor arcs (may be marking dependent)
- $\Pi \in \mathbb{N}^n$ is the vector that associates the priority level to each transition;
- $G \in (\mathbb{N}^n \to \{true, false\})^n$ is the vector that associates a guard condition related to the place marking with each transition;
- $M_0 \in \mathbb{N}^n$ is the vector that associates an initial marking of each place (initial state);
- $Atts = (Dist, Policy, Concurrency, W)^m$ defines the transitions attributes set:
  - $Dist \in \mathbb{N}^m \to F$ is a probability distribution function associated to each transition, with $F \leq \infty$. This distribution may be depedendent of the marking;
  - $Policy \in \{prd, prs\}$) defines the memory policy adopted by a transition (prd-preemptive repeat different, the default value, identical to enabling memory policy;prs-preemptive resume, equivalent to age memory policy);
  - $Concurrrency \in \{ss, is\}$ defines the transition concurrency policy, in which ss represents single server semantics and is represents infinity server semantics;
  - $W \in \mathbb{N}^+$ is the weight function, associating a weight(w_t) to immediate transitions and a rate(λ_t) to timed transitions.

The addition of timed transitions introduces the concept of multiple enabling, which should be considered in timed transitions with enabling degree greater than one. In this case, the firing semantics should consider the number of tokens that can be fired in parallel. The possibilities of semantics are:

- Single Server (SS): The firing time is computed when the transition is enabled. After the transition firing, a new time will be computed if the transition remains enabled. Therefore, the firings will occur in series, regardless of the enabling degree of the transition;
- Infinite Server (IS): the entire set tokens from the enabled transition is processed simultaneously. Then, all the tokens will be processed in parallel.

Figure 2.14 depicts an example of a SPN model. Places are represented by circles, SPNs transitions are depicted as hollow rectangles (timed transitions) or as filled rectangles (immediate transitions). Arcs (directed edges) connect places to transitions and vice versa. Tokens (small

filled circles) may reside in places. A vector containing the current number of tokens in each place denotes the global state (i.e., marking) of a Petri Net. An inhibitor arc is a special arc type that depicts a small white circle at one edge, instead of an arrow, and they are used to disable transitions if there are tokens present in a given place. The behavior of Petri Nets, in general, is defined in terms of a token flow, in the sense that tokens are created and destroyed according to the transition firings. Immediate transitions represent instantaneous activities, and they have higher firing priority than timed transitions. Besides, such transitions may contain a guard condition, and a user may specify a different firing priority among other immediate transitions.

Figure 2.14 represents the availability of a system comprising three servers. Each token in the place **Servers Up** denote one server that is properly running. All three servers might fail independently, by the firing of (exponential) timed transition **Failure**. A token in **Servers Down** might be consumed either by immediate transition **Repairable** or by immediate transition **Non-repairable**. Weights are assigned to each of those transitions to represent the probability of firing one or another. When the failed server can be repaired, the transition **Repairable** puts a token in **Servers to repair**. When the repair is not possible, the transition **Non-repairable** puts the token in **Servers to replace**. The transitions **Repair** and **Replace** fire after exponential delays corresponding to those activities. The probability of having at least one server available, the average number of servers waiting for repair or replacement and other similar metrics can be computed from the underlying CTMC generated from that SPN.

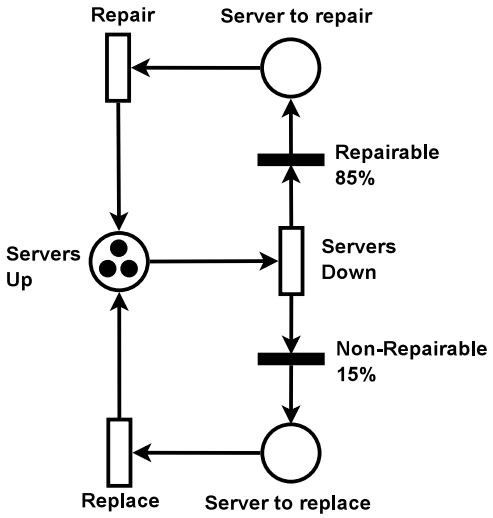


Figure 2.14: Example SPN

SPNs also allow the adoption of simulation techniques for obtaining dependability and performance metrics as an alternative to the generation of a CTMC, which is sometimes prohibitive due to the state-space explosion.

2.6.4 Capacity Oriented Availability

Users and analysts may also be interested in how much service the system is delivering, including situations of partial failure. An important measure associated with available capacity is Capacity Oriented Availability (COA) [11]. This metric allows estimating how much service the system is capable of delivering considering failure states. According to Liu and Trivedi [70], COA is computed as:

$$COA = \sum_{i=1}^{n} \frac{i}{n} \pi_i, \tag{2.20}$$

where i is the number of available service units and π_i is the probability that these units are working. The contrary measure is called Capacity Oriented Unavailability (COUA), and can be calculated as: $COUA = (1 - COA)$.

2.6.5 Hierarchical Modeling

Complex systems modeling may take advantage of multiple levels of models, forming hierarchies, where the lower-level models capture the detailed behavior of sub-systems and the topmost is the system-level model [117]. Hierarchical modeling makes sense when the whole system can be analyzed by the sub-systems that compose it.

Hierarchical models scale better when the number of system sub-systems and sub-system components than non-hierarchical regular models. It is possible to isolate the number of system components in each level of the hierarchy, decreasing the analysis time of the whole model.

Hierarchical models also support heterogeneous modeling, in which the models presented in distinct hierarchical levels are also distinct. For example, an analyst may use combinatorial RBD model at the bottom level and a SPN at the top level of a hierarchical model.

As the number of sub-systems and sub-system components in cloud computing and virtual networks are not a small one, the adoption of hierarchical modeling may improve the dependability analysis of these systems.

2.7 Software Aging and Rejuvenation

The ordinary usage of computational systems results in the fatigue of its components. Such a phenomenon is called aging and can result in premature failure of a computer system. The procedure that aims to reverse the aging of computer systems is called rejuvenation. It is the technique which refreshes system components in order to avoid failures caused by its aging [89]. It can be categorized into two primary levels: hardware and software rejuvenation.

From the hardware aging perspective, besides the replacement of failed components to rejuvenate the system as a whole, a re-initialization, and a possible power-off period, can reduce the electric effects over microelectronic components [113] and is commonly adopted

as the hardware rejuvenation technique. We perform preventive maintenance through nodes re-initialization with the aim of mitigating physical aging.

The phenomenon of software aging refers to the accumulation of errors during the execution of the software, which eventually results in its crash/hang failure. Software often exhibits an increasing failure rate over time, typically because of increasing and unbounded resource consumption, data corruption, and numerical error accumulation.

Some techniques to tackle software aging considers to classify the system into layers and propose rejuvenation approaches following layers' classes [7]. The considered layers into virtualized software systems is presented in Figure 2.15.

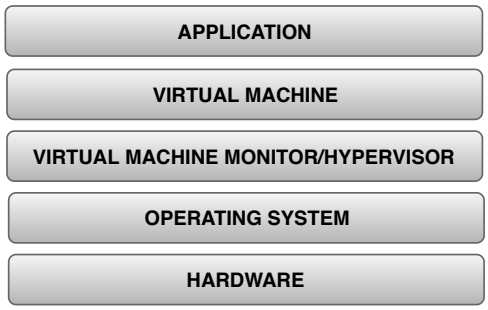


Figure 2.15: System Layers considered in the software rejuvenation classification

Some of the techniques regarding layered classification are:

- Application re-initialization: it is the restart of the application. In production environments, it is performed automatically. In cloud computing environments, the cloud framework has a set of services that also degrades due to aging phenomena. Aiming at eliminating the aging of the framework components, their re-initialization is also required;

- Virtual machine re-initialization: due to its hardware isolation feature, virtual machines can be replicated, including the usage of alternative physical machines, to rejuvenate the primary VMs through its re-initialization. This approach mitigates and even eliminates in several cases the downtime overhead perceived by the end users;

- VMM/hypervisor re-initialization: with the broad adoption of cloud computing in data centers, the Virtual Machine Monitor gained growing attention. VMMs are complex enough to suffer software aging phenomena [7]. VMMs are usually implemented as operating modules, such as kvm.ko, for KVM hypervisor, in Linux systems. Their re-initialization comprises unload and reload of such modules;

- Operating system re-initialization: this technique re-initializes the entire operating system, which usually involves rebooting the firmware, re-executing Power On Self Test (POST) routines, rebooting the kernel, the essential operating system services, and finally rebooting the services being provided;

- Fast Operating system re-initialization: the default operating system reboot leads to a large delay due to the sequence of stages that need to be executed before the application can run. The operating system fast reboot maintains the current state of the machine concerning the firmware, so it is not necessary to perform the POST routines. It is only required to reboot the operating system kernel, the necessary routines and finally the application;
- Physical machine re-initialization: the hardware restart results in cleaning up the internal state of all provided software. Indeed, the hardware re-initialization also implies software rejuvenation whose origin if due to degradation phenomena over time.

2.8 Sensitivity Analysis

Sensitivity analysis is a technique used to determine the factors that are most relevant to the parameters or outputs of a model. In computer system dependability analysis, one can apply sensitivity analysis to help identify the components that most influence system availability, reliability, or performance [55].

A sensitivity analysis can be performed through distinct methods, including: differential analysis, correlation analysis, regression analysis, or perturbation analysis. The simplest method is to repeatedly vary one parameter at a time while keeping the others constant. By applying this method, a sensitivity ranking is obtained by observing changes in the model output.

Parametric sensitivity analysis aims at identifying the factors for which the smallest variation implies the highest impact in the model's output measure. System parameters, such as failure and repair rates, are used to identify the most relevant factors that impact on system's operation.

Differential sensitivity analysis is performed through the partial derivatives of the measure of interest with respect to each input parameter. Through partial derivatives of closed-form equations, one is able to find the most important system factors and improve its behavior in an optimized fashion.

The sensitivity of a given measure Y, which depends on a specific parameter λ, is computed as shown in Equation (2.21), or Equation (2.22) for a scaled sensitivity:

$$S_\lambda(Y) = \frac{\partial Y}{\partial \lambda} \tag{2.21}$$

$$SS_\lambda(Y) = \frac{\theta}{Y}\frac{\partial Y}{\partial \lambda} \tag{2.22}$$

$S_\lambda(Y)$ is the sensitivity index (or coefficient) of Y with respect to λ , and $SS_\lambda(Y)$ is the scaled sensitivity index, commonly used to counterbalance the effects of largely different units between distinct parameters values.

3

Related Works

This chapter presents a list of related works in the main topics covered in this thesis. The researches are divided into two categories: **Hierarchical Modeling of Virtual Environments** and **Software Aging and Rejuvenation of Server Virtualized Systems**. The following sections are not intended to provide an exhaustive view of the works published on those topics, but rather to point out significant advances which go towards a similar direction as this research do, or give a basis for future extensions.

3.1 Hierarchical Modeling of Virtual Environments

Kim et al. [64] construct a two-level hierarchical model of non-virtualized and virtualized systems representing two hosts systems in an active/active redundancy configuration. Fault trees are used to represent the overall system in the top-level, and CTMCs are used to capture the behavior of sub-components in the low-level. They incorporate hardware failures (e.g., CPU, memory, power), and software failures, including VMM, VMs, and application failures. The aim is to estimate steady-state availability, downtime, and COA. A strong point of this work is the scope analysis, considering failure and repair rates from the hardware to the application. Besides scope analysis, this work also resembles this thesis by the comparison between a baseline environment and a proof-of-concept environment. This paper is one that most closely resembles this thesis. However, in this thesis, a complete study of the application was carried, with a focus on VNF chains. Furthermore, we were also able to measure VM live migration time intervals for the considered service, whereas Kim et al. adopted previous measurements for VM live migration that do not consider the application.

Melo et al. [84] propose models to evaluate the capacity of nodes in cloud computing environments. They combine RBD and SPN models to state the real amount of available hardware resources at predetermined time intervals. A strong point of this paper is to consider a generic cloud computing platform, composed by Platform Manager, Cluster Manager, Block-based Storage, File-based Storage, and Instance Manager. As expected, all of these generic components are presented in the openstack platform. Through a sensitivity analysis, the authors were able to state that the Node component is higher impactful for COA in comparison with the

VM. This paper resembles this thesis, as availability and COA analyses were performed in a cloud computing environment. The main differences are our concern regarding high availability, as well as our analysis of SFC applications.

Dantas et al. [32] present availability models for private cloud architectures based on the Eucalyptus platform, as well as a comparison of costs between evaluated architectures and similar infrastructure rented from a public cloud provider. The COA and steady-state availability were used to compare architectures with distinct numbers of clusters. A heterogeneous hierarchical modeling approach, using RBD and CTMC, was employed to represent the systems considering both hardware and software failures. They concluded that it takes 18 months, on average, for the private cloud architectures to be paid off the cost equivalent to the computational capacity rented from a public cloud. This work resembles the presented thesis mainly for the adoption of hierarchical modeling of availability and COA, whereas the main distinction is our analysis of network function virtualization applications.

Mauro et al. [33] address an availability evaluation of a chain of network nodes implementing an SFC managed by a Virtualized Infrastructure Manager (VIM). A double-layer model is adopted, where Reliability Block Diagram describes the high-level dependencies among the architecture components, and Stochastic Reward Networks (SRN) model the probabilistic behavior of each component. In particular, a steady-state availability analysis is carried out to characterize the minimal configuration of the overall system guaranteeing the so-called five 9's requirement, along with a sensitivity analysis to evaluate the system robustness concerning variations of some key parameters. This paper closest resembles this thesis regarding the analysis of service function chains availability. A strong point of this paper is the adoption of elasticity in the SRN to model the variation of the workload during 24 hours. However, they do not present a testbed to capture time parameters with the aim to be used as input in their model.

Fernandes et al. [42] proposed and evaluated a method to estimate dependability attributes in virtualized network environments. The proposed approach provides a basis for estimating metrics, such as reliability and availability, through the adoption of hierarchical and heterogeneous modeling based on RBDs and SPNs. The hierarchical modeling was adopted with the aim to mitigate the complexity of representing large virtual networks. Experimental results demonstrated the feasibility of the proposed modeling approach, in which dependability metrics have been estimated from results generated by the resource allocation algorithm for virtual networks. A strong point of this paper is to reach the goal in demonstrating the feasibility in adoption proposed modeling approach to study dependability metrics. This work resembles the presented thesis by the study of virtualized network infrastructures adopting hierarchical modeling, whereas no concern about high availability was presented, which is a noticeable difference between the works.

Costa et al. [29] proposed a hierarchical model, adopting RBD and CTMC, to assess the availability of a mobile cloud platform. A strong point of this work was the validation of designed models through testbed measurements by automatically fault injecting and repairing of

the infrastructure, taking into account the three evaluated layers: hardware, operating system, and the Mobile cloud Platform. This work resembles this thesis by the experiments of fault injection in the experimental infrastructure.

Cotroneo et al. [30] proposed a methodology for the dependability evaluation and benchmarking of NFV Infrastructures (NFVIs), based on fault injection. Authors applied experimental availability, defined as the percentage of traffic units, such as packets, that are successfully processed during a fault injection experiment. They aimed to analyze the effects over requests, due to VNF unavailability, that will not be processed by an IP Multimedia Sub-system (IMS) deployed as VNF. During fault injection experiments, they found experimental availability varying from 8.01% to 82.40% for all tested scenarios, with an average value of 51.37%. They showed the important factors that can impact the experimental availability, concluding that while it is important to have redundant and reliable devices to prevent I/O faults, it is even more important to introduce additional resources to mitigate CPU and memory faults, including more VM instances and physical CPUs to compensate for faulty ones.

Gonzalez et al. [51] focused their work in the system availability of virtualized Evolved Packet Core (vEPC), specifically in how to assess the availability of a vEPC and which are the main availability concerns to be considered. As the all-IP framework for providing converged voice and data on a 4G Long-Term Evolution (LTE) network, the study of potential failures sources in a vEPC environment, that is provided through VNFs under NFVI, is quite relevant. Authors provided a Stochastic Activity Network (SAN), an extension of SPNs, claiming that it is applicable to study how sensitive the availability is to the main parameters. The presented numerical results show that a cluster of twelve Commercial Off-The-Shelf (COTS) servers are required to obtain five 9's of availability in the studied vEPC system. Furthermore, the steady-state availability of each individual hardware (H), software (S), and hypervisor (Y) must be equal to 99.99%. Authors adopted equal values for failure and repair rates of H, S, and Y, i.e.: $\lambda_H = \lambda_S = \lambda_Y$ and $\mu_H = \mu_S = \mu_Y$, and based their recommendations to improve availability in general design criteria, such as: design VNF functions with adequate operational redundancy to cover individual failures; and provide high robustness software, hardware and hypervisor entities. This paper resembles thesis by the study of VNF and NFVI availability.

Endo et al. [35] made a systematic review to present and discuss High Available (HA) solutions for Cloud Computing, where they argued that delivering a higher level of availability has been one of the biggest challenges for Cloud providers (similar to telecommunication service providers serving VNF). As a result of the systematic review, HA cloud solutions were organized into three layers. Redundancy was classified in the services middle layer. Four redundancy models were presented, according to Availability Management Framework (AMF) of the Service Availability Forum (SAF): 2N, N+M, Nway, and Nway active. The systematic review revealed a preference for 2N model, due to its simplicity.

Herker et al. [58] modeled different backup strategies for VNF service chains and provided algorithms for the resilient embedding of such VNF service chains in diverse Data

Center (DC) topologies. Author's purpose was to deploy VNF service chains with predefined levels of availability in DC networks.

Yoon et al.[131] implemented a virtualized network computing testbed on openstack cloud. Such a testbed allows users to configure various types of virtualized networks using their VNFs. They classified virtualized networks in 4 categories, according to how VNF: host VM/nested VM; host VM/nested container; VM based; and container-based. No concerns about high availability were looked on. Just one neutron node and one controller node were deployed, whereas ten compute nodes are available.

Table 3.1 summarizes the most relevant presented related works regarding hierarchical modeling, establishing a comparison between each of them in this thesis. The compared aspects were:

- type of performed dependability evaluation;
- if some HA Testbed was assembled, analyzed, or both;
- concerns regarding high availability;
- adoption of cloud computing in the studies;
- study of virtual network functions or service functions chains;

Our aims to define such aspects were justify the adoption of the used formalisms in the research area and capture the aspects of high availability. One remarkable confirmation obtained from related works is the adoption of small testbeds for measurements experiments, without HA features such as clustering and 3N redundancy.

Table 3.1: Comparison with most relevant related works regarding virtual environments

	Evaluation	**HA Testbed**	**Parameters source**	**five 9's**	**VNF/SFC**
Kim et al. [64]	FT, CTMC	No	previous literature, estimated guesses	Yes	No
Melo et al. [84]	RBD, SPN	No	previous literature	No	No
Dantas et al. [32]	RBD, CTMC	No	previous literature	No	No
Mauro et al. [33]	RBD, SPN	No	previous literature	Yes	Yes
Fernandes et al. [42]	RBD, SPN	No	previous literature	No	Yes
Costa et al. [29]	RBD, CTMC	No	previous literature	No	No
Cotroneo et al. [30]	Measurement	Yes	-	No	Yes
Gonzalez et al. [51]	SPN	No	estimated guesses	Yes	Yes
This thesis	Measurement, RBD, CTMC and SPN	Yes	previous literature, estimated guesses, experiment	Yes	Yes

3.2 Software Aging and Rejuvenation of Server Virtualized Systems

Machida et al. [74] presents analytic models using stochastic reward nets for three time-based rejuvenation techniques for VMs and VMMs. The authors goal is compute the steady-state

availability and the number of transactions lost per year regarding three analyzed rejuvenation techniques, named: (i) Cold-VM rejuvenation, in which all VMs are shut down before the VMM rejuvenation; (ii) Warm-VM rejuvenation, in which all VMs are suspended before the VMM rejuvenation; (iii) Migrate-VM rejuvenation, in which all VMs are moved to the other host server during the VMM rejuvenation. The analyzed server virtualization system comprises two servers. This work resembles this thesis by the adoption of dependability modeling with a focus on availability. A positive aspect of this work is the report of an optimum rejuvenation schedule for VM and VMM rejuvenation. A difference for this thesis is does not consider the remaining layers of virtualized systems, such as services in the top-level and operating system and physical server in bottom-level.

Matos et al. [79] presents an approach that uses time series to schedule rejuvenation to reduce the downtime by predicting the proper moment to perform the rejuvenation. They used a testbed environment to perform experiments looking for aging symptoms, characterized by the consumption of CPU time, memory space, hard disk space, and process IDs. They adopted an Eucalyptus testbed with 4N redundancy for servers hosting VM, however they adopt only one server to the cloud controller. A positive aspect of this work was the detection of aging aspects in virtual and resident memories with experiments with a duration of only 72 hours. This work resembles this thesis by the adoption of a redundant testbed for servers running virtual machines in a cloud environment. A difference for this thesis is not to consider the adoption of an HA environment, such as Pacemaker/Corosync/HAProxy, in the Testbed provision assembling.

Araujo et al. [12] investigate the memory leak and memory fragmentation aging effects on the Eucalyptus cloud-computing framework, as well as proposed a software rejuvenation strategy to mitigate the observed aging effects. A positive aspect of this work was to experimentally detect the existence of the investigated aging effects in the cloud environment under study. The aging phenomenon was detected through workloads composed of intensive requests addressing different virtual machines. The rejuvenation mechanism was implemented sending restarting signals to the aged processes. This paper resembles the presented thesis by the analysis of cloud Virtual Infrastructure Manager, named Eucalyptus. However, the impact of the adopted rejuvenation mechanism over dependability measures was proposed as future work.

Alonso et al. [6] present a comparative experimental study of six different rejuvenation techniques with different levels of granularity: (i) physical node reboot, (ii) virtual machine reboot, (iii) OS reboot, (iv) fast OS reboot, (v) standalone application restart, and (vi) application rejuvenation by a hot standby server. A key aspect of this work was to discover that application-level rejuvenation strategies are better as a first tentative approach to mitigate the aging effects. If the rejuvenation at a higher level was not effective, the rejuvenation of the next level could be adopted. The authors argue that their results can contribute to availability improvement, but did not analyze dependability metrics. That last aspect constitutes the key distinction between paper authors and this thesis.

Nguyen et al. [91] show an availability model for a virtualized servers system using

stochastic reward nets considering software rejuvenation of VMs and VMMs. The models take into account: (i) the detailed failures and recovery behaviors of multiple VMs; (ii) several failure modes and corresponding recovery behaviors; (iii) dependency between different sub-components (e.g., between physical host failure and VMM). The authors' goal is to show numerical analysis on steady-state availability, downtime in hours per year, transaction loss, and sensitivity analysis. This work resembles this thesis regarding the dependencies among virtualized system components. The main contribution of this paper is to show that a frequent rejuvenation policy on VM may lower the SSA of the virtualized systems, whereas that on VMM may enhance the system SSA. We can highlight the main difference between this paper and the presented thesis as the absence of top-level service in authors analyzes.

Melo et al. [83] investigate the software aging effects on the openstack cloud computing platform using a stressful workload. They collect information about the utilization of hardware resources and openstack related processes. An All-in-one openstack deployment mode was adopted. The study was carried out by performing repeated instantiations and terminations of virtual machines. Furthermore, a resource utilization prediction based on time series was performed, with the aim to identify possible failure scenarios. The authors argue that time series models explain the behavior of the adopted cloud computing platform, as well as its associated process. Four different types of time series were analyzed, namely: (i) the linear model; (ii) the exponential growth model; (iii) the quadratic model; and (iv) the S-Curve Model. The goal was to identify which one presents the best fitting with the collected results. The main aspect of this work was the detection of software aging issues regarding database used by openstack, specifically memory leak aging effects. As a result, openstack processes, like nova-api, suffer degradation due to memory shortage. This paper resembles the presented thesis by analyzing the software aging and rejuvenation phenomena in openstack cloud environment. However, a single server All-in-one environment was adopted, without considering High Available implementations.

Torquarto et al. [116] presents an availability model based on SPN for virtualized servers with software rejuvenation of VMM enabled by VM live migration scheduling. During model analyses, this paper adopts two different approaches for VM live migration, named: Cold-Standby Migration and Warm-Standby Migration. A positive aspect of this paper is observed from its results: in environments with a heavy workload, the rejuvenation scheduling brings significant improvement over availability. This work resembles this thesis by adopting live migration during the software rejuvenation process. However, High Available environments were not covered.

Table 3.2 summarizes the most relevant related works regarding SAR of server virtualized systems, also establishing a comparison between each of them and this thesis.

The compared aspects were:

- VMs live migration as a mechanism to rejuvenation;
- Software Aging and Rejuvenation of virtualized networks, with a focus on VNF and SFC adoption;
- concerns regarding high availability;

Table 3.2: Comparison with most relevant SAR works

	LM Rejuvenation	**SAR of VNF/SFC**	**Consider HA**	**Cloud Analysis**
Machida et al. [74]	Yes	No	No	No
Matos et al. [79]	No	No	No	Yes
Araujo et al. [12]	No	No	No	Yes
Alonso et al. [6]	No	No	No	No
Nguyen et al. [91]	No	No	No	No
Melo et al. [83]	No	No	No	Yes
Torquarto et al. [116]	Yes	No	No	Yes
This thesis	Yes	Yes	Yes	Yes

- adoption of cloud computing in the studies.

Our aims to define such aspects were justify the adoption of the proposed research framework and verify the originality in the joint selection of these aspects.

4

A Methodology for Provisioning of High Available VNF Chains

The main objective of this thesis is to propose and analyze stochastic models to evaluate and improve virtual network functions. An aimed goal of this research is to propose high availability solutions in the provisioning of these VNF chains.

We now present the adopted methodology in order to achieve these objectives. We aim to support network specialists on estimate availability and capacity of VNFs, improving the provision of virtualized network services into their infrastructures.

4.1 Methodology Overview

Figure 4.1 illustrates the adopted methodology and contextualizes the environment in which this work is inserted. The main activities of the methodology are:

- **System Understanding**: it encompasses the comprehension of the systems, the identification of its components and functionalities and the following infrastructure definition;
- **Environment Conception**: with the defined infrastructure, the environment conception takes place with the implementation of the system that will further be modeled;
- **Definition of Parameters and Metrics**: with the understanding of the system and the defined environment, the next activity defines the parameters the will be used to estimate the defined metrics;
- **Models Design**: this activity is composed of the selection of suitable models and the tools that are able to design and analyze them;
- **State Input Parameters**: it aims at identifying the sources of input parameters;
- **Evaluation**: it aims at performing the analysis of the studied system, producing the results, i.e., the metrics estimations, of designed models;
- **Yield Recommendations**: based on generated results from models analysis, produce the recommendations for network and cloud specialists.

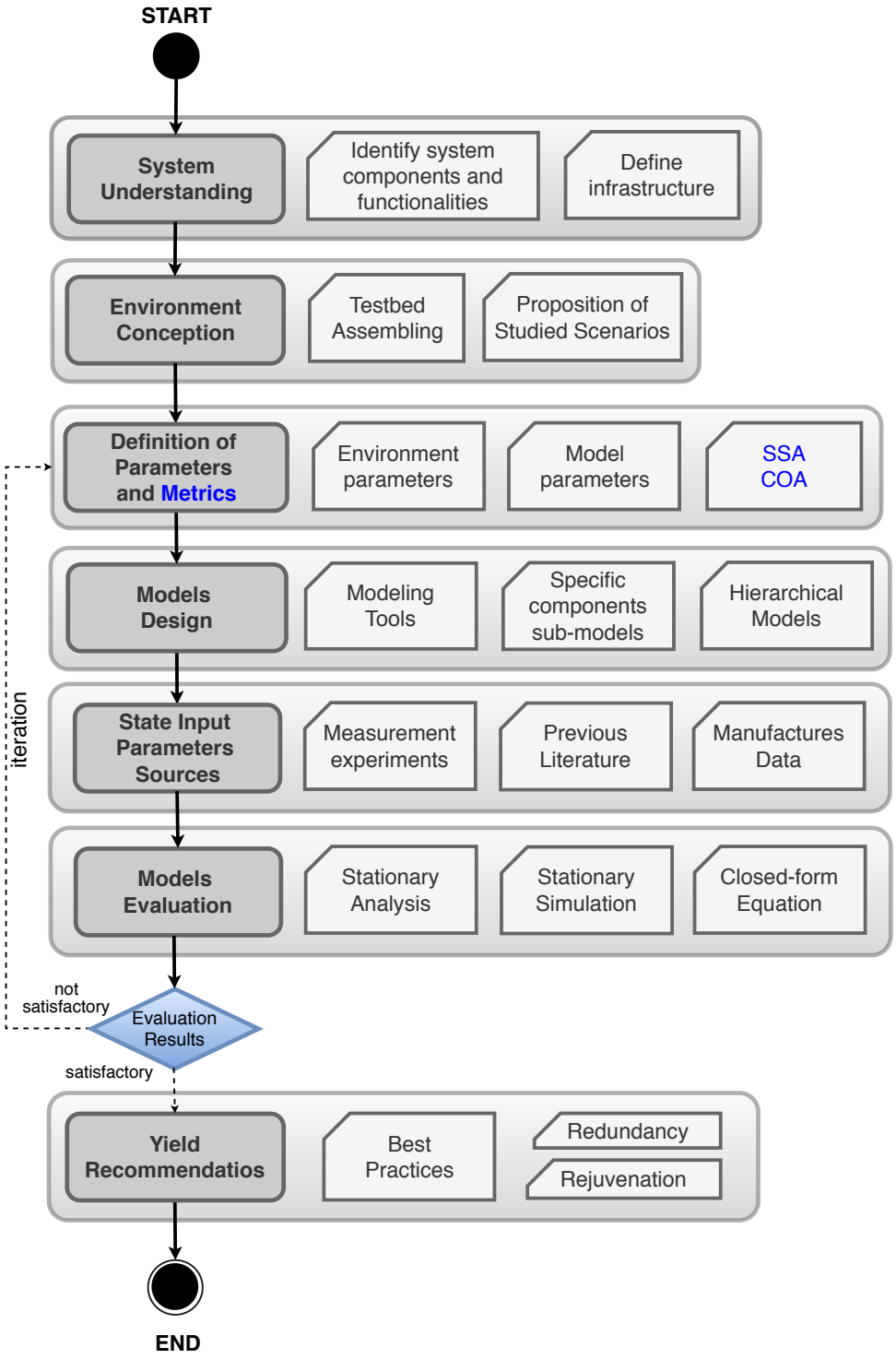


Figure 4.1: Overview of Adopted Support Methodology

We point out that the proposed methodology can be adopted by users with experience in combinatorial and state-space based models for infrastructure planning and decision making in virtualized environments, such as private cloud and its services. It is relevant to highlight that this methodology can also be adopted to assist in the modeling of several systems that present characteristics similar to those addressed in this work. The methodology activities are detailed below.

4.2 Methodology Activities

4.2.1 System Understanding

Aiming at planning virtualized infrastructures, such as server virtualization and cloud computing ones, requires to understand how that systems work, identifying its main components and making a survey of the main existing solutions, applications, and functionalities. For this thesis, this activity represents a relevant goal, because it aids to delimit the work, mitigating the risks to its conclusion. Understanding the system requires great attention and special care by the analyst to avoid misinterpretation that could compromise the later steps of the methodology. This activity is essential, as it enables us to learn about the techniques that can be adopted, adapted, or will have to be created.

- Inputs: reading of technical and scientific material regarding virtual networks (including SDN, NFV, and SFC), dependability modeling, and high available cloud infrastructure;
- Actions: identification of virtual network components and virtual infrastructure manager;
- Outputs: theoretical references, related works, as well as software and hardware requirements for the assembling of cloud infrastructure.

4.2.2 Environment Conception

The assembling of testbeds is a direct activity to consolidate the comprehension of the studied systems. This activity corresponds to the selection, installation, configuration, and management of an environment to execute VMs providing VNFs. The testbed must provide an environment in which an NFV ecosystem can be executed and the VNF chains can be evaluated. We opt by an open-source solution to keep the research with low costs. Some fundamental decisions of this activity are presented below:

- Preliminary testbed: a Proxmox VE Server Virtualization was deployed, with the aim at analyzing VNFs. Such a decision was motivated by its simple installation, configuration, and management, as well as our previous experience [53] with the platform. We use container-based virtualization with OpenVZ;
- Subsequent testbed: Following, an HA Openstack Cloud Infrastructure was implemented. During the infrastructure update, we performed the HA environment installation (both hardware and software). Openstack has an official project regarding SFC [132]. It provides an implementation of the classifier and the VNF chains. We use full-virtualization with Kernel-based Virtual Machine (KVM), the default openstack hypervisor. It has the advantage to provide both server virtualization and network virtualization solutions. As the main OpenFlow Protocol agent, the Open vSwitch was selected as L2 virtual solution (as Open vSwitch was originally designed for virtualized networking, it is fully compatible with L2-L4 networking and can benefit for previously presented advantages provided by SDN). The practical expe-

riences with these structures aids to generate dependability models that can be sufficiently representative.

The summary of inputs, actions, and outputs of this activity is presented below:

- Inputs: software and hardware requirements for the assembling of virtual machines infrastructure provisioning;
- Actions: installation, configuration, and tuning of HA server virtualization and HA cloud computing environments;
- Outputs: functional testbed and proposition of studied scenarios.

4.2.3 Definition of Parameters and Metrics

Any element in which the variation of its values modifies the solution of a problem, without, however, changing its nature, is called a parameter. In this work, we can identify two types of parameters: those used to excite the measurement environments and those used to estimate the metrics of interest on models. So, at this stage, the analyst must define which scenarios and metrics will be covered, focusing mainly on those that have the greatest influence on the quality of provided service.

Regarding environment parameters, the virtualization approach (Full-, Para-, or Container-based Virtualization), the SFC size, and the video cache spool capacity are the most influential.

Regarding models parameters, we widely adopted Mean Time To Failure (MTTF) and Mean Time to Repair (MTTR) of the system components. Furthermore, in rejuvenation scenarios, any preventive maintenance execution, resulting in a live migration of the chain's VMs, is scheduled by Mean Time Between Preventive Maintenance (MTBPM) parameter. As MTBPM expires, a set of conditions is verified to state if the environment is favorable to the execution of preventive maintenance. If the verification is not favorable, all issues postponing preventive maintenance must be resolved before migrate the VNF chain.

The following conditions were adopted:

1. the destination server must be active.
2. there will be no simultaneous VNF chain migration.
3. the VNF chain in destination server must be active.
4. the Open vSwitch in destination server must be active.

Condition 1 is an evident requirement, as the destination server that will receive VMs, must be active. Condition 2 prevents simultaneous preventive maintenance in both servers, that would put the overall system in a down state. Conditions 3 and 4 avoid downtime at the start of any preventive maintenance, verifying if all the VMs, services, and the Open vSwitch are active in the destination server.

Besides MTBPM, we are also considering two other parameters: Mean Time To Perform Preventive Maintenance (MTTPPM) and Mean Time To Chain Live Migration (MTTCLM) (Figure 4.2). The MTTPPM is the estimated mean time interval in which the maintenance procedures related to the aging elimination take place. On the other hand, the MTTCLM is the time interval required to migrate all the VMs belonging to a service chain.

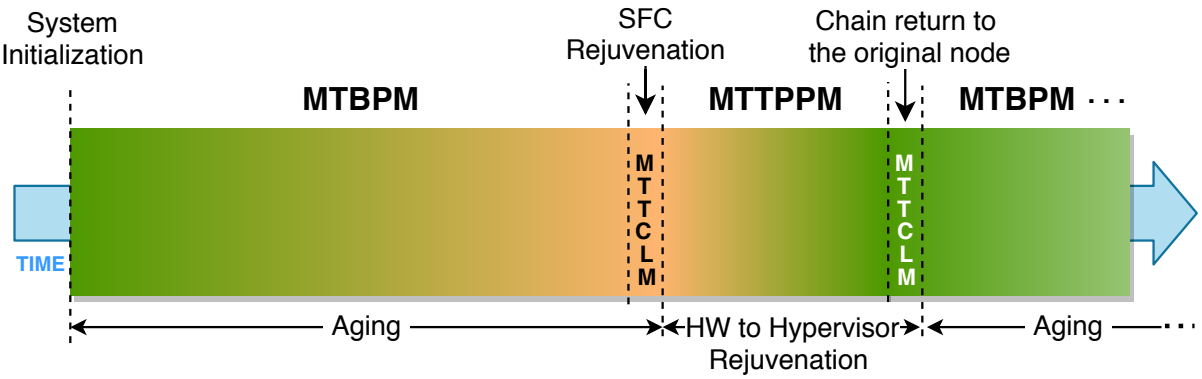


Figure 4.2: Aging and Rejuvenation processes

We consider that the server and their softwares age during MTBPM period, as depicted in Figure 4.2. As soon as MTBPM expires, the chain's VM live migration starts, and during this period, the chain is not working. We take advantage of MTTCLM window to reinitialize the VMs. So, when the service chain is activated in destination openstack node, it is rejuvenated. During the MTTPPM window, preventive maintenance is performed. The last step of the process is to return the chain to its original openstack node, in order to minimize the downtime if the destination node fails. Some of the model parameters only require valid choices. For example, the percentage of a component fast repairs in comparison with the percentage of its slow repairs. It is evident that the sum of these percentages must be 100% for a correct estimation of the metric under analysis.

The main conclusions of this thesis will emerge through the metrics analysis. In this research, we aim at estimating dependability metrics, namely Steady-State Availability and Capacity Oriented Availability.

The summary of inputs, actions, and outputs of this activity is presented below:

- Inputs: list of scenarios that will be studied;
- Actions: identification of parameters types;
- Outputs: list of parameters that will be used during the evaluations; list of requirements to models design; list of metrics of interest.

4.2.4 Models Design

The activity of models designing is associated with the representation of the infrastructure scenarios, such as those observed in environment conception activity. Analytical models

may not be satisfactory in its first proposals. Iterative modeling approach may be applied to adjust proposed models through successive improvements until it is suitable to represent an achievable target behavior. Dependability modeling may adopt an iterative approach to consider progressively: the dependencies between system components [106] or yet improvements in the metrics of interest [80].

In this phase, the scenarios are modeled considering the following variables: number of nodes, number of VNFs, type of services, and redundancy mechanisms. We adopted a hierarchical and heterogeneous approach for modeling, based on a reliability block diagram used at low-level, and CTCM or SPN in the top-level.

A modeling tool that has a visual environment is helpful. It assists in model development while also enables the computation of the metrics of interest. Among some tools, we can mention Mercury [109]. The inputs of this activity are: the type of metrics to be evaluated (for example, dependability metrics, such as availability), established in previous activities; understanding the operation of key components or subsystems, as well as the description of dependencies or interconnections between system components.

Firstly, RBD models were created to evaluated Cache VNFs considering both a single server and two redundant servers. The RBD models enable using closed-form formulas for computing Steady-State Availability. Closed-form formulas are useful to facilitate the generation of sensitivity indices with respect to each model parameter. After, hierarchical and heterogeneous models were conceived to represent systems with a higher number of components, when load balancers were inserted in the analyzed systems. Next, specific sub-models were designed to the study of VNF chains in a cloud computing environment. The combination of these sub-models represents the entire system whose metrics are estimated.

The summary of inputs, actions, and outputs of this activity is presented below:

- Inputs: the metrics to be evaluated; understanding the operation of key components or subsystems; the description of dependencies or interconnections between system components;
- Actions: design and implementation of sub-models; design of hierarchical models composed by previous designed sub-models;
- Outputs: the models of proposed scenarios.

4.2.5 State Input Parameters Sources

With the aim to adequately represent system behavior, some of the input values must be obtained through measurement experiments. Experiments are used to study the behavior of processes and systems. Besides the implementation of a test environment, the experiments may also require the creation of support tools. Such a requirement will depend on the existence of tools that support the combination of the analyzed metrics, the observed scenario, and the investigated environment. However, for scenarios in which experiments execution are not feasible, the input

parameter values should be obtained through previous literature with similar features to conduced study. In addition, manufacturer data can also be used to feed the proposed models. However, caution is required as the values provided by the manufacturers may be overestimated. In this case, the analyst may adopt a reduction factor based on literature or even in empirical knowledge, to reduce values and bring them closer to reality. In some cases, the input parameter values of some models may be obtained through other models, which may be sub-models of this thesis itself or models from the literature.

- Input: parameters of the proposed models;
- Actions: obtain input values for the parameters of each system component, either by performing measurement experiments, literature studies, data provided by component manufacturers or other models;
- Output: input parameter values.

4.2.6 Models Evaluation

During this activity, we analyze the designed models, producing the results for the studied metrics. In the evaluation process, the lower-level models should be resolved first, because their results will be used as input parameters for top-level models. The solution method (i.e., numerical analysis, simulation or closed-form equations) may vary for each model, depending on the constraints of modeling formalism.

The results are compared with predefined reference values. The aim is to observe the possible improvements that can be obtained by the adoption of the strategies implemented by the models in existing infrastructures.

New iterations may be conducted to capture updates implemented in the modeled system or to adjust proposed models through successive improvements until it is suitable to represent an achievable target behavior.

- Input: models loaded with input parameters values;
- Actions: define whether the estimated metrics are satisfactory or not;
- Output: additional iterations if not satisfactory; improvements in evaluated systems if satisfactory.

4.2.7 Yield Recommendations

After the results generation, the analyst should report recommendations regarding improvements that may be implemented in an existent system, or establish configuration requirements for systems under design and/or deployment. We will report the improvements over Steady-State Availability and Capacity Oriented Availability metrics, that cloud operators may reach through the adoption of the redundancy and rejuvenation strategies presented in this thesis.

- Input: metrics results from models analysis;

- Actions: analyze improvements from the evaluated models;
- Output: yield recommendations regarding metrics improvements.

4.3 Final Remarks

This chapter presented the methodology used to propose and analyze stochastic models to evaluate and improve virtual network functions. Through a set of activities that cover since the understanding of the system's essential operations to propose improvements to system components, this chapter provides details of evaluation procedures that can be replicated by other researchers.

5

Measurement Experiments

This chapter presents the measurement experiments that were executed in the assembled testbeds during this research. They were conducted to obtain a set of parameter values that will be inserted in the proposed models. First, the adopted workload, based on User Generated Content (UGC), is presented. After, the proper testbeds are presented. Finally, the measurement experiments for each testbed, as well as their results, are reported.

5.1 Workload for User Generated Content

Global IP traffic has increased fivefold between 2011 and 2015, and it will increase threefold until 2020. Additionally, Cisco VNI [25] indicates that 82% of all IP traffic will be video and, at every second, nearly a million minutes of video content will cross the network by 2020. Video streaming is the main factor of this growth, and is basically composed of two media platforms: Video on Demand (VoD), such as Netflix, and User Generated Content (UGC), such as Youtube. Motivated by these values, we performed the workload characterization of User Generated Content (UGC) video system, enabling us to reproduce the behavior of an UGC video system in a controlled environment. Abhari and Soraya [2] performed a workload characterization of Youtube, during a five-month period for 250,000 videos. The characterization revealed a file popularity behavior that fits Zipf distribution [3, 18] and a file size behavior that fits gamma distribution [87]. The probability mass function (pmf) and probability density function (pdf) of these distributions are given respectively by:

$$f(x) = \frac{1}{x^s \sum_{i=1}^{n} (1/i)^s} \qquad x = 1, 2, ..., n, \tag{5.1}$$

and

$$f(x) = \frac{x^{\beta-1} e^{-x/\alpha}}{\alpha^{\beta} \Gamma(\beta)} \qquad x > 0. \tag{5.2}$$

In Equation(5.1), s parameter is skew factor of Zipf distribution. In Equation(5.2), α and β are respectively the scale and shape parameters of gamma distribution. The parameter values presented by Abhari and Soraya [2] and exhibited in Table 5.1 were adopted during the workload tool configuration.

Table 5.1: Parameters for UGC video characterization

Parameter	Value
skew(s)	0.9
scale(α)	5441.93
shape(β)	1.80

This workload characteristics have been used during the performed measurements experiments.

5.2 Proxmox Server Virtualization Testbed

The NFV video cache cluster shown in Figure 5.1 was designed to support TTFs estimation. This testbed adopts Proxmox as the MANO component. Proxmox provides a management interface for orchestration, which allows containers' loading and monitoring.

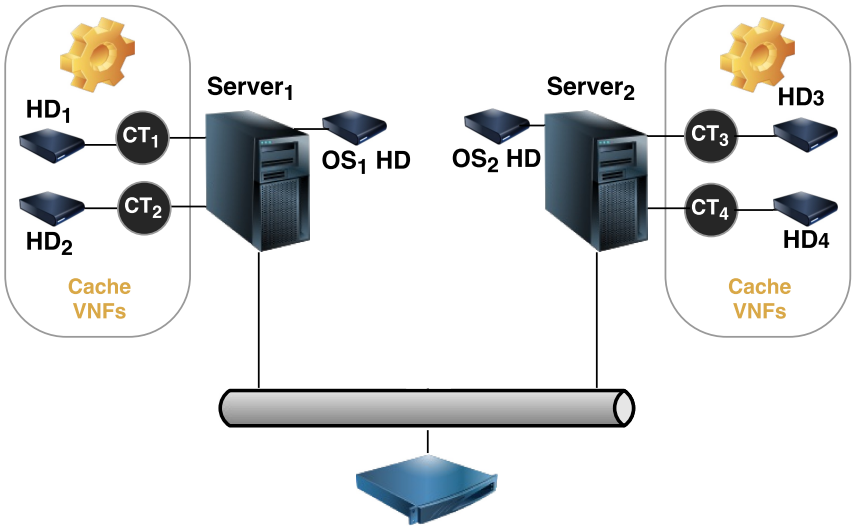


Figure 5.1: Testbed for UGC video cache cluster

Two servers (AMD Phenom Quad-Core Processors, 2.3 GHz, 8 GB RAM, Gigabit Ethernet adapter) compound the cluster. They have 3 directed attached hard disks (HD) (250GB, 6Gb/s, 7200RPM). Two HDs are exclusively used for cache server spools (labeled as HD_1 and HD_2 for $Server_1$ and HD_3 and HD_4 for $Server_2$), whereas the third HD rooms the server operating system.

Each server hosts two Container (CT) with sole hard disks. Such an approach improves availability, as highlighted in our previous work [52]. All VNF cache servers were implemented using Squid Server 3.4.2. A Gigabit switch interconnects the testbed nodes. The nodes with Squid Servers adopted an active-active redundant video cache cluster to measure the failure rates.

5.3 HA Openstack Cloud Testbed

We accomplished HA by incorporating features such as redundancy for failover and replication for load balancing in each component without using costly specialized hardware and software.

- **HA Openstack Cloud: Hardware Infrastructure**

Our HA Cloud is formed by 9 servers divided into 3 Pacemaker clusters, as depicted in Figure 5.2.

- HA Controllers cluster is composed of 3 servers. Two of them with Intel Xeon CPU E3-1220 3.10GHz, 16GB RAM, 4 Gigabit Ethernet Adapters, and 1TB of hard disk capacity, and the third with Intel Core CPU i7-2600 3.40GHz, 8GB RAM, and also 4 Gigabit Ethernet Adapters, and 750GB of hard disk capacity, divided in one disk with 500GB e another with 250GB;
- HA Computes cluster nodes differs from HA Controllers in the amount of RAM for Intel Xeon server (32GB);
- HA Neutron cluster is composed of three servers with AMD Phenom Quad-Core Processors, 2.3GHz, 8 GB RAM, 5 Gigabit Ethernet adapters, and 500GB of hard disk capacity. The fifth network adapter of neutron's nodes is used for an external Internet connection. Two Gigabit switches interconnect testbed nodes.

Figure 5.2 shows the redundant connections between clusters and both testbed switches.

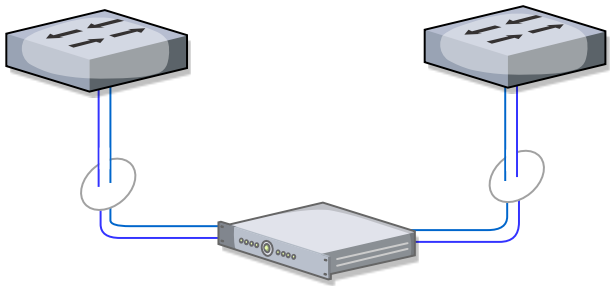


Figure 5.3: Bonding between each testbed server and the switches

Indeed, there are four UTP Cat 5e cables physically connected to each node, as depicted in Figure 5.3. Each pair of cables forms a redundant round-robin Linux bond [127], depicted by the ellipses, and there is no cabling SPOF. Bonding interfaces are a cost-effective way to provide hardware-level network redundancy to the cloud infrastructure.

We are able to ensure that there are no hardware SPOF in the assembled HA Openstack Cloud. The heterogeneity of equipment is due to their availability in the laboratory. During the testbed assembling, no configuration differences were required due to servers heterogeneity. The operating systems, openstack components, and pacemaker/corosync packages were installed without distinctions, as described below.

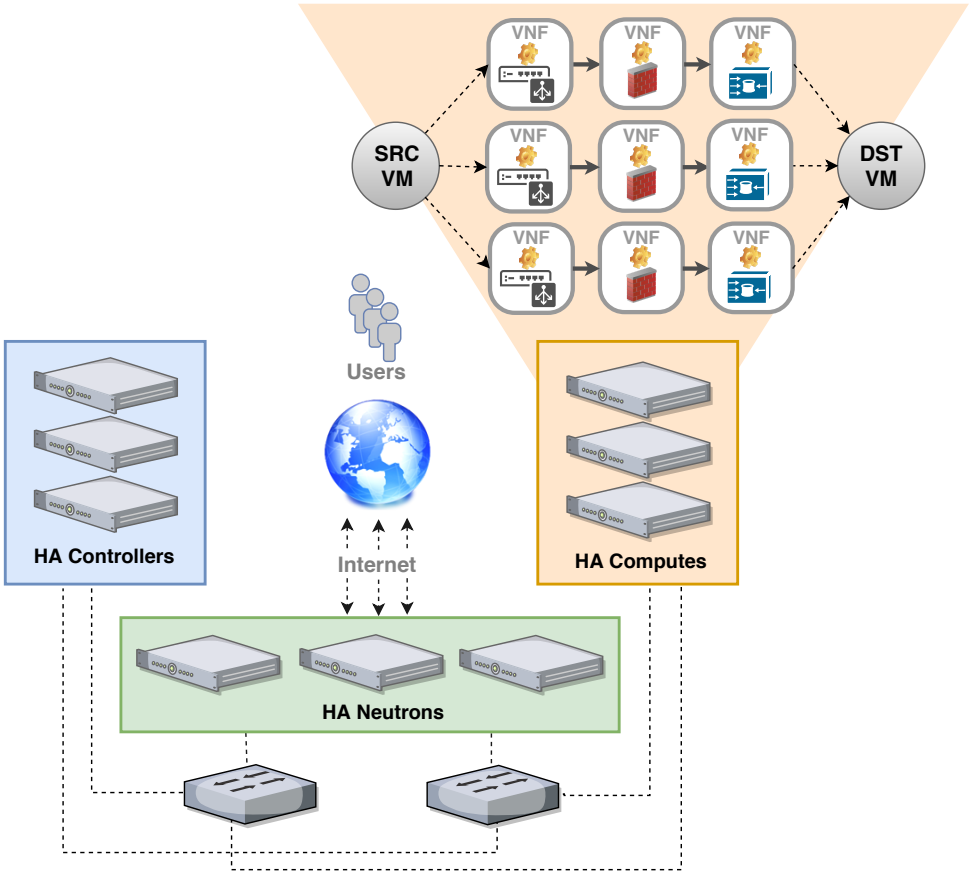


Figure 5.2: Openstack HA Cloud Architecture

▪ HA Openstack Cloud: Software Components

Regarding the installation of openstack software components, Pacemaker and Corosync were installed and configured in all 9 HA cloud nodes, as depicted in Figures 5.4, 5.5, and 5.6. The management (initialization, shutdown, re-initialization, and monitoring) of each openstack daemon is performed by Pacemaker.

MariaDB was adopted as the underlying database. Its HA solution was implemented through Galera Cluster. The AMPQ message queue system was implemented, and its native HA solution configured and enabled.

HAProxy was installed to perform the load balancing to requests performed to the *HA Controllers* cluster. Any requests regarding instantiation of VMs are forwarded to the Virtual IP (VIP) of *HA Controllers* cluster. HAProxy receives these requests and forwards them, using a round-robin policy, to the aimed openstack component. HAProxy was also wrapped by Pacemaker (refer to Table A.2).

All the clusters (*HA Controllers*, *HA Computes*, and and *HA Neutron*) were configured in an active/active/active redundancy. Even without spare nodes, there are advantages in adopt active-active redundancy in the selected clustering softwares system, such as management centralization (all the nodes in the cluster may be managed in any server) and dropping of recovery rates.

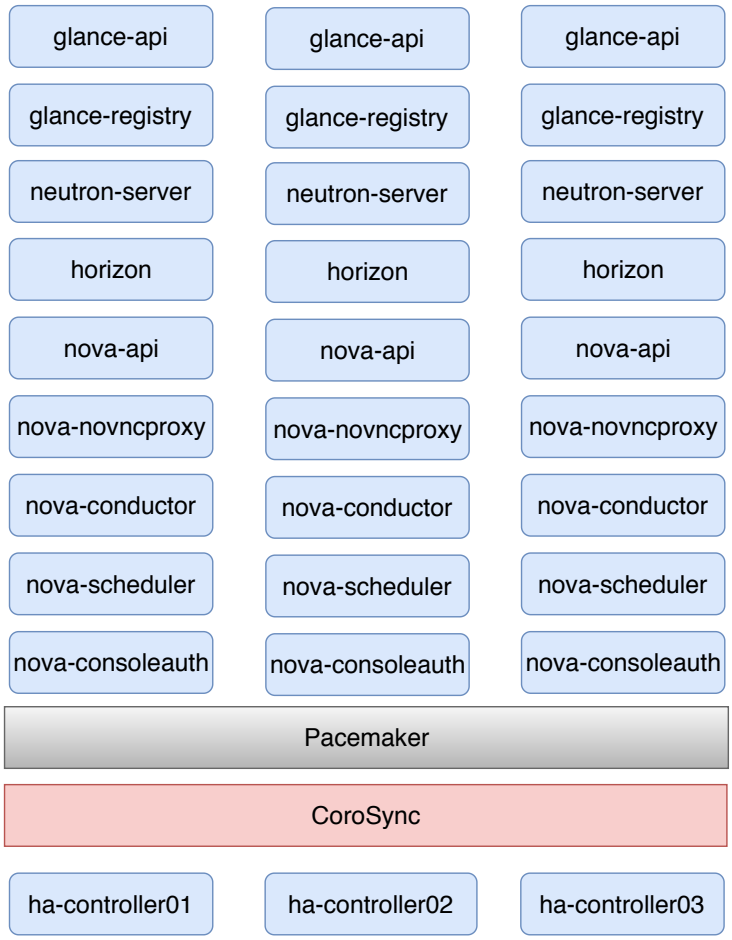


Figure 5.4: Pacemaker: HA Controller Cluster

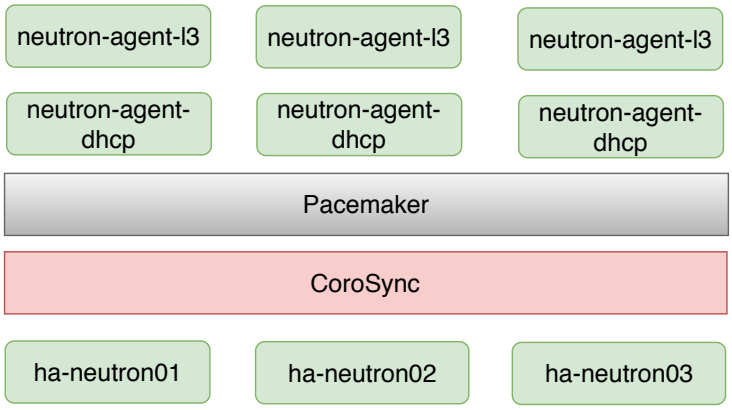


Figure 5.5: Pacemaker: HA Neutron Cluster

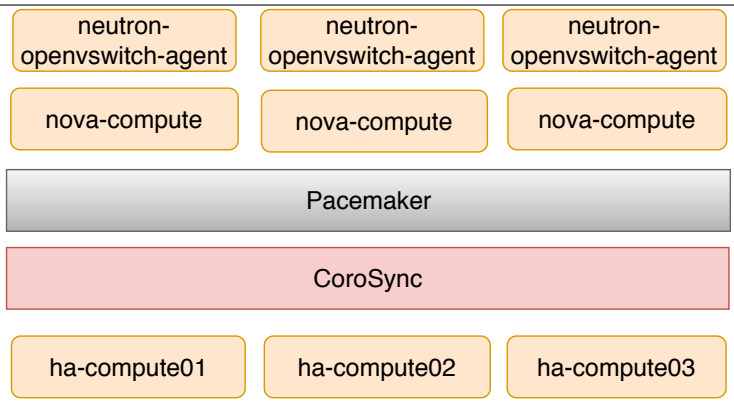


Figure 5.6: Pacemaker: HA Neutron Compute

Openstack HA implementation is not free of conflicting requirements. For example: it is not possible to bound services to IP address 0.0.0.0, which results in offering the service in all configured interfaces, including loopback. So, the service is bounded to virtual IP instantiated by Pacemaker. But some resource agents configured in pacemaker monitor port services bounded to the loopback address, by default. This impacts the deployment time of HA Openstack solution. Some troubles can arise, such as performability issues related to bottlenecks of network I/O. We had experienced and troubleshooted a similar issue in a previous work [52].

5.4 Time To Failure Measurements in Proxmox Server Virtualization Testbed

Software component MTTFs are not readily available, and experiments can be performed to estimate them. We proposed and executed a series of experiment replications aiming at measuring TTFs in the UGC video cache cluster of Proxmox Server Virtualization testbed (Figure 5.1). The goal was to detect how long all the caching processes take to cause a cluster failure.

The adopted methodology to measure the TTFs comprises 2 activities, as depicted in Figure 5.7:

- Activity 1 - Configure workload generation tool: Web-Polygraph (WP) [102] was adopted to excite the video cache VNF cluster (see Figure 5.8). It includes both client and server-side simulators. Two additional machines were added to the Proxmox Testbed with the aim to execute the WP client-side and server-side, as depicted in Figure 5.8.

  The client-side is used to generate and forward the configured workload to video cache VNF cluster, populating it in a balanced way through a round-robin policy. As any traditional caching system, when the required objects are found, the caching system answers the request with the desired object. Otherwise, the request is forwarded to WP server-side, that answers with the requested objects.

  WP natively supports Zipf distribution, so that we could model file popularity behavior through WP $popZipf$ function. Gamma distribution, required to represent file size behavior, is not natively available in WP, but we could represent it using a WP additional resource called

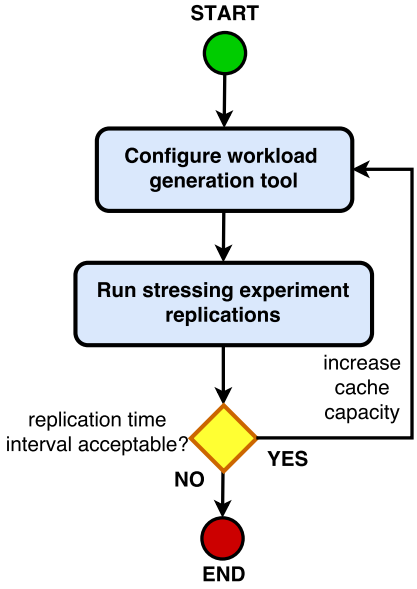


Figure 5.7: Methodology to estimate TTFs

user-defined distribution. Non-native distributions can be modeled as a value-frequency table. We generated value-frequency pairs presented in the Table 5.2 using R [47] *rgamma* function, adopting shape and scale parameters as defined in Table 5.1.

Table 5.2: rgamma results: file sizes and frequencies

File size(MB) [min : max)	**Frequency(%)**
0.128 : 5.12	28.65
5.12 : 10.24	32.31
10.24 : 15.36	19.60
15.36 : 20.48	10.17
20.28 : 25.60	4.98
25.60 : 30.72	2.35
30.72 : 35.84	1.11
35.84 : 40.96	0.45
40.96 : 46.08	0.215
46.08 : 51.20	0.085
51.20 : 56.32	0.03
56.32 : 61.44	0.03
61.44 : 66.56	0.01

Observing Table 5.2, one can notice that the majority of UGC video files (32.31%) have size between 5.12MB and 10.24MB.

- Activity 2 - Run stressing experiment replications: it aims to capture TTFs of VNF video cache executing UGC video workload.

  These experiments are hard to conduct because of time they will take, or yet because of the difficulty of attributing the cause of failures. With the aim of accelerating the results of the

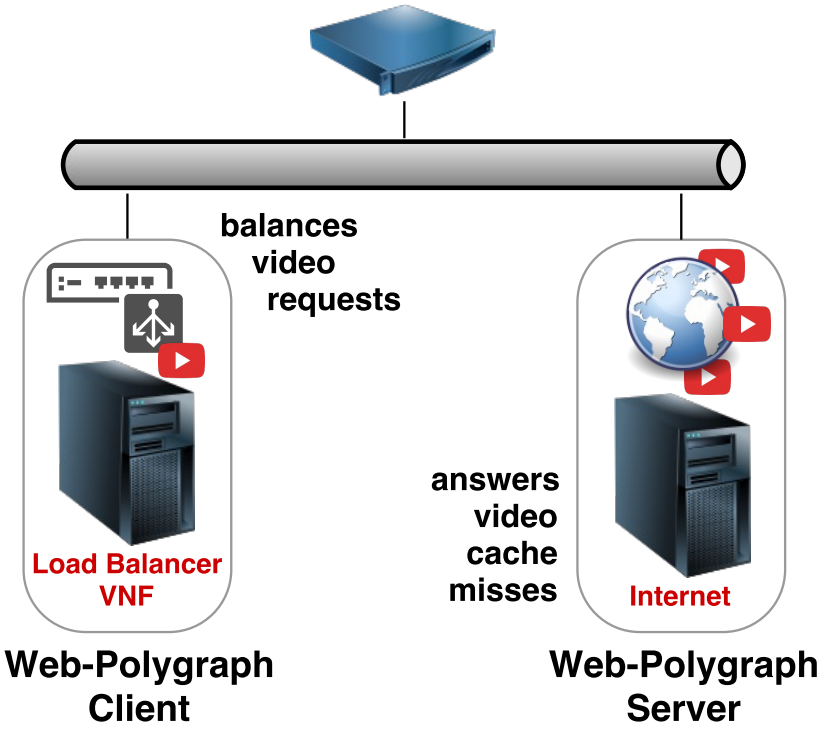


Figure 5.8: Proxmox Testbed: additional machines to execute TTF experiments

experiments, avoiding unpredictable experiments duration time intervals for TTFs estimation, we limited the total cache capacity of the cluster. With UGC workload and a limited cache space, the storage capacity was quickly exhausted, overloading the video cache cluster and accelerating failure events.

Observing the methodology of Figure 5.7, if the experiments replication time is below a predefined threshold, the cache capacity is doubled, and three new replications are executed. The experiments were started with 4GB of cache capacity (1GB per container) and a time threshold of 1-hour. We were able to increase the cache capacity twice without infringing the adopted threshold, reaching 16GB of total cache capacity.

Figure 5.9 exhibits the obtained TTFs. As can be noted, as the storage capacity increases, the failure times also increase. Adopting the configured User Generated Content workload, a capacity storage greater than 16GB does not result in cluster failure in the 1 hour threshold, and we finalize the experiments.

The MTTF=315.33s from 16GB scenario was adopted as input parameter in availability models presented in the next chapter, because it is the most relevant due to higher storage capacity.

5.5 Service Function Chain Migration Experiments in HA Cloud Testbed

The measurement experiments adopted a VNF chain composed of a load balancer, a firewall, and a cache aimed at store User Generated Content videos, as depicted in Figure 5.10.

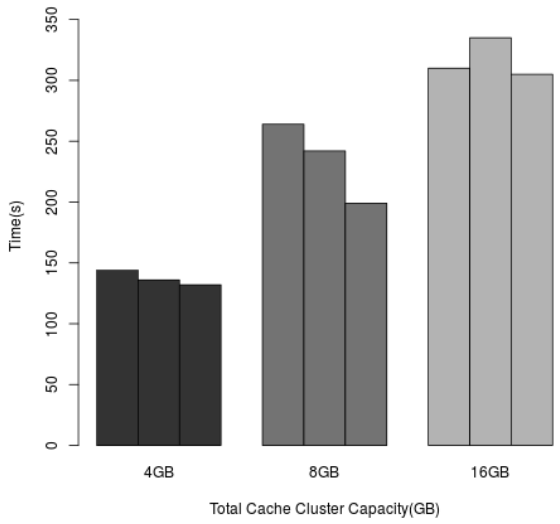


Figure 5.9: Time to Failure (TTF) of the 3 evaluated cache capacities

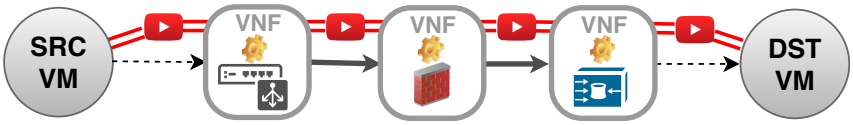


Figure 5.10: UGC video flow through SFC composed of a load balancer, a firewall, and a cache server

The UGC workload (Section 5.1) was also configured in Web-Polygraph for SFC live migration experiments. However, distinctly for Proxmox Testbed, WP server-side was executed in DST VM (Figure 5.10) whereas SRC VM executes WP client-side.

The first VNF to process the UGC packet flow is the load balancer. We adopted the round-robin policy using HA Proxy. After, firewall VNF inspects packets, looking for SYN flooding and ping of death flows. Lastly, the cache VNF looks for requested video files.

We choose KVM as the hypervisor to execute VM instances in openstack. It is one of the most popular hypervisors with openstack deployments, it is also the default configuration option, and it has a low configuration time cost. Moreover, as full virtualization platform presents large migration times in comparison with para-virtualization or container-based virtualization (as can be seen in our previous work [53]), our goal is establishing an upper limit for SFC migration time.

When an instance is booted for the first time, neutron assigns a virtual port to each network interface of the instance. A port for neutron is a logical connection of a vNIC to a subnet (a layer 3 IPv4 or IPv6 network object). A VM instance running on KVM uses its vNIC so that the applications, such as load balancers and cache server, can communicate to the outside world through.

We implemented service function chains using openstack SFC API. When the SFC API executes the Open vSwitch driver, an Ethernet frame will pass through a set of virtual devices, as

depicted in Figure 5.11.

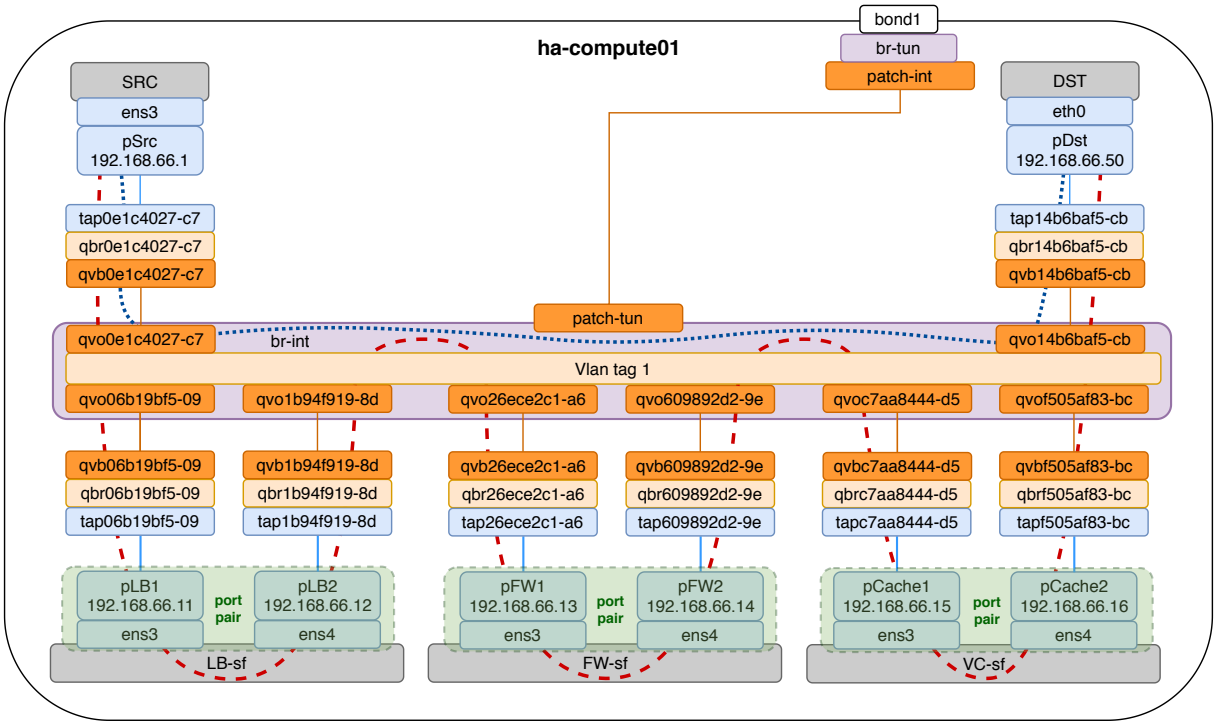


Figure 5.11: Detailed openstack SFC flow

The packets flow starts from Web-Polygraph SRC virtual machine, through its vNIC ens3. But for this vNIC to be operational, it has to be able to connect to something on the other end that gets it to some destination. This is the purpose of the other components of the network architecture depicted in Figure 5.11. So, after vNIC ens3, the next device is the tap software-only interface. Tap devices are the way that KVM implement a vNIC attached to the VMs. The traffic from a vNIC (such as ens3), in a virtual machine instance, can be observed on the respective tap interface in the host Compute server. The tap interface was useful during the execution of the experiments because it enables traffic monitoring in the openstack compute host.

After tap interface in the stack, the Open vSwitch driver requires a Linux Bridge. The virtual interface with prefix qbr shows the Linux Bridge (the "q" stands for quantum, the initial name of neutron project in the openstack community). All the iptables support that implement access rules for virtual machines are configured in the Linux Bridges. The next two layers, qvb and qvo, constitutes the virtual ethernet (veth) cable. The first one, qvb, represents the bridge veth side. The second one, qvo, represents the Open vSwitch veth side, connected to the Open vSwitch integration bridge br-int. The integration bridge is the central virtual switch that most virtual devices are connected to, including instances, DHCP servers, and routers. The multi-tenant feature is implemented by the integration bridges using VLANs. As one can observe in Figure 5.11, all the five VMs used in the migration experiments belongs to the same VLAN, identified by its tag ID 1.

The flow from Web-Polygraph SRC VM is now forward, through OpenFlow rules, to the qvo interface of LB-sf VM, passing through qvb, qbr and tap, reaching LB-sf ens3 vNIC. As all

openstack networking services and openstack Compute instances connect to a virtual network via ports, it is possible to create a traffic steering implementation for service chaining using only openstack ports.

All the VMs belonging to the service function chain were instantiated in a unique compute node (ha-compute-01). In scenarios in which multiple compute nodes would be used, the integration bridge will forward the traffic through the veth composed of patch-tun and patch-int connections. The bond1 interface represents the collections of physical network interfaces.

The SFC API implements the concept of port pair. A port pair represents a specific service function. The port pairs used in the SFC live migration experiment can be observed in Figure 5.11, and are detailed in Table 5.3.

VM/Port Pair Name	Port	IP	vNIC
LB-sf/LBPP_C1	pLB1	192.168.66.11	ens3
	pLB2	192.168.66.12	ens4
FW-sf/FWPP_C1	pFW1	192.168.66.13	ens3
	pFW2	192.168.66.14	ens4
VC-sf/CachePP_C1	pCache1	192.168.66.15	ens3
	pCache2	192.168.66.16	ens4

Table 5.3: Port pairs for SFC live migration experiments

Regarding LB-sf VM, the ports pLB1 and pLB2 form a port pair. The pLB1 port is the ingress port in the pair, whereas the pLB2 port in the egress port in the pair. Similar behavior is presented by port pairs (pFW1, pFW2) and (pCache1, pCache2).

The adopted flow classifier was created using the rules below:

i. IPv4 traffic;
ii. the source IP address equal to Web-polygraph (192.168.66.1);
iii. the destination IP address equal to cache video VNF (192.168.66.15);
iv. TCP protocol;
v. the TCP source ports in the 30000:65535 range;
vi. the TCP destination port equals to the video cache VNF server (3128);
vii. neutron source port equals to WP Source VM (pSrc).

From the dashed line depicted in Figure 5.11, one can observe that the traffic from Web-Polygraph SRC VM will ingress in the port chain through port pLB1. The load balancer VNF will apply its policies in the traffic of vNIC ens3, forwarding the resulting throughout ens4. The traffic flow egress from port pLB2, continuing its journey inside the port chain. All the virtual interfaces and virtual devices steer the flow until it reaches VC-sf ens3 vNIC. The video cache VNF looks up its spools. If the desired object is found, the video cache answers the request, otherwise, it forwards the request to the Web-Polygraph DST VM. The DST VM simulates all the Internet. Any requested object will be found and sent to the video cache VNF

implemented on VC-sf VM. As the answer is unrelated with the port chain, it is sent for the SRC VM outside the port chain, as can be observed by pointed line.

5.5.1 Experiments execution

The TISVEP (Tuning Infrastructure for Server Virtualization Experiment Protocol) [53] was adopted to manage the execution of the experiments. TISVEP automatizes all required configuration steps to replicate experiments.

We collected three time intervals during experiments:

i. deletion of SFC API openvswitch chain rules;

ii. chain migration;

iii. creation of SFC API openvswitch chain rules.

The procedures (i) and (iii) are mandatory for SFC migration in openstack SFC API because the packet flow transmitted through the chain will fail after migration if the Open vSwitch rules were not deleted before VMs migration. As TISVEP is extensible, five new messages were created to capture VNF chain live migration times. They are presented below:

- 312 - createChain: results in the execution of all required commands in a controller node to mount the chain;
- 313 - deleteChain: similar to message 312, but aimed to remove the chain;
- 314 - detectMigration: sent to a compute node, it results in monitoring the VMs hypervisor Process Identification (PID). While the PIDs are not detected, TISVEP waits;
- 315 - fwRulesInjection: results in required netfilter [107] configuration of security rules aimed to allow the chain's traffic;
- 316 - migrateChain: execute the chain's migration commands in the controller node.

All physical and virtual machines run TISVEP server-side with the goal to configure the experiment environment during its execution. Figure 5.12 depicts the TISVEP messages that were sent with the aim to capture the time intervals (i), (ii), and (iii).

As can be noted in Figure 5.12, messages 313, 316, and 312 are delivered for one controller node, whereas messages 314 and 315 are delivered to the migration target compute node. According to TISVEP, all messages must be answered with the STATUS of executed configuration commands, so that the experiments can be monitored in the experimenter's system.

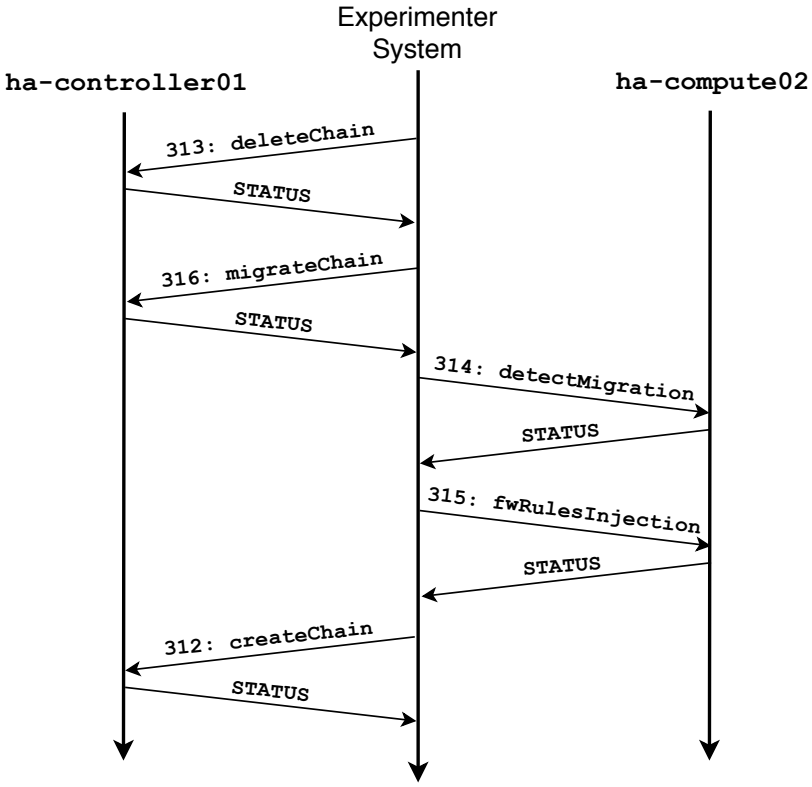


Figure 5.12: TISVEP messages for live migration experiments

Forty migration experiments were performed and Table 5.4 exhibits 95% mean confidence intervals.

Table 5.4: Mean Confidence Intervals

Time interval(s)	LB(s)	Mean(s)	UB(s)
SFC rules deletion	6.119	6.147	6.175
VNF chain migration	108.797	114.511	120.226
SFC rules creation	12.049	12.089	12.129

Specifically to this set of migration experiments, message 316 (migrateChain) conduced VMs back and forth nodes $ha-compute01$ and $ha-compute02$, the more powerful servers of HA Computes cluster. The mean values of Table 5.4 will be inserted in the SPN models presented in the next chapter.

5.6 Final Remarks

This chapter presented the measurement experiments that were conducted during this research. The adopted workload was defined, as well as how it was applied to the assembled testbeds. We automatized the execution of the experiments extending the TISVEP protocol. The results of the measurement experiments are inserted as input parameter values in the proposed models during their evaluation.

6

Availability Models

6.1 Introduction

This chapter presents the models that represent the components aimed at providing VNF chains. Preliminary, a study was conducted with models representing VNFs in the Proxmox VE server virtualization platform. After, Service Function Chains sub-models were proposed for HA openstack cloud.

6.2 Models for VNFs in Proxmox server virtualization infrastructure

For this first proposed models regarding VNFs, the components of the Proxmox platform were modeled using RBDs, as depicted in Figure 5.1. The Proxmox servers, the containers, and the applications provided in containers are the represented components.

6.2.1 Model for VNF Cache Cluster

This model represents the utilization of only two containers running in one node of a Proxmox Server Virtualization Infrastructure. The components of the model, represented by the composition of series and parallel RBDs, are depicted in Figure 6.1. The Server block represents the hardware components of the physical server. The Storage (S) block represents the persistent memory of the server. The OS block represents the Operating System. The CT_i blocks represent the containers, whereas the APP_i blocks represent the VNF cache servers.

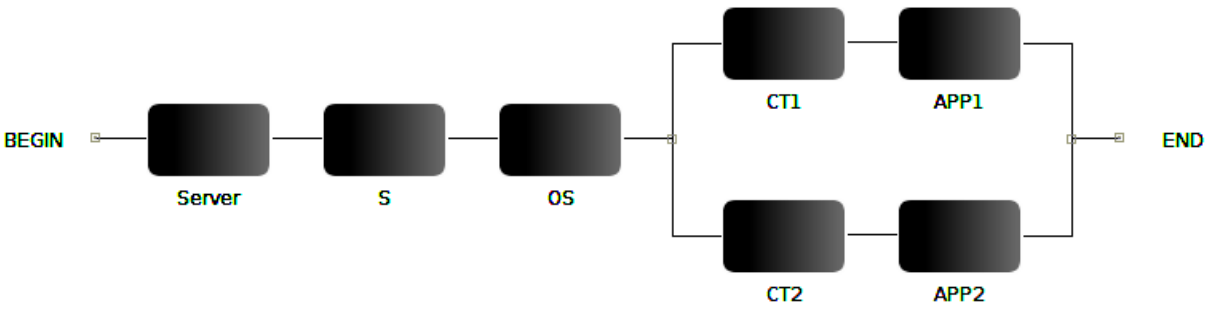


Figure 6.1: RBD for VNF Cache Cluster

The closed-form equation for availability of the non-redundant VNF cluster is expressed as:

$$SSA = A_{Server} \times A_S \times A_{OS} \times (1 - (1 - A_{CT1} \times A_{APP1}) \times (1 - A_{CT2} \times A_{APP2})) \quad (6.1)$$

Table 6.1 report the input parameters adopted in the first case study (Figures 6.1 and 6.2). The parameters represent the mean time to failure and repair of each component. In addition to using manufacturer data for the modeling phase, cloud computing analysts, technicians, and managers can use historical data from their environments. In this way, the values can contribute to the obtention of closer results.

Table 6.1: Dependability parameters for VNF Cache Cluster

Parameter	Description
$MTTF_{SRV}$	Server Mean Time To Failure
$MTTR_{SRV}$	Server Mean Time To Repair
$MTTF_{S}$	Storage Mean Time To Failure
$MTTR_{S}$	Storage Mean Time To Repair
$MTTF_{OS}$	Operating System Mean Time To Failure
$MTTR_{OS}$	Operating System Mean Time To Repair
$MTTF_{C}$	Container Mean Time To Failure
$MTTR_{C}$	Container Mean Time To Repair
$MTTF_{APP}$	Application Mean Time To Failure
$MTTR_{APP}$	Application Mean Time To Repair

6.2.2 Model for Cache VNF and Load Balancer

The interaction among applications was modeled using top-level CTMCs. The details are provided below. This first proposed model represents a cache VNF system chained with a load balancer appliance. We adopted four cache VNF sub-system due to the number of containers in the first assembled testbed (Figure 5.1). Each cache VNF sub-system is composed by one Server(HW), one Storage(S), one OS, one container(CT), and one cache VNF(APP). The series RBD depicted in Figure 6.2 represents the low-level cache VNF model.

Figure 6.2: RBD model for VNF cache sub-system

The reliability of each component ($R_i(t)$) [75] is considered to compute the resulting MTTF of a series RBD:

$$
\begin{aligned}
R_i(t) &= e^{-\lambda_i t} \\
MTTF_i &= \int_0^\infty R_i(t)dt = \int_0^\infty e^{-\lambda_i t}dt = \left[\frac{-e^{-\lambda_i t}}{\lambda_i}\right]\Bigg|_0^\infty \\
MTTF_i &= \lim_{t\to\infty}\frac{-e^{-\lambda_i t}}{\lambda_i} - \lim_{t\to 0}\frac{-e^{-\lambda_i t}}{\lambda_i} = 0 - \frac{-1}{\lambda_i} = \frac{1}{\lambda_i}
\end{aligned}
\tag{6.2}
$$

So, for the entire system,

$$
\begin{aligned}
R(t) &= e^{-\lambda_1 t} \times e^{-\lambda_2 t} \times e^{-\lambda_3 t} \times \ldots \times e^{-\lambda_n t} = e^{-\sum_{i=1}^{n}\lambda_i t} \\
MTTF_{SS} &= \frac{1}{\sum_{i=1}^{n}\lambda_i},
\end{aligned}
\tag{6.3}
$$

where λ_i is the failure rate of each system component. So, the sub-system $MTTF_{SS}$ may be estimated by the Equation 6.4.

$$
MTTF_{SS} = \frac{1}{\lambda_{HW} + \lambda_S + \lambda_{OS} + \lambda_{CT} + \lambda_{APP}} \tag{6.4}
$$

The maintainability of each component($M_i(t)$) is considered to compute the resulting MTTR of a series RBD:

$$
\begin{aligned}
M_i(t) &= 1 - e^{-\mu_i t} \\
MTTR_i &= \int_0^\infty 1 - M_i(t)dt = \int_0^\infty e^{-\mu_i t}dt = \left[\frac{-e^{-\mu_i t}}{\mu_i}\right]\Bigg|_0^\infty \\
MTTR_i &= \lim_{t\to\infty}\frac{-e^{-\mu_i t}}{\mu_i} - \lim_{t\to 0}\frac{-e^{-\mu_i t}}{\mu_i} = 0 - \frac{-1}{\mu_i} = \frac{1}{\mu_i}
\end{aligned}
\tag{6.5}
$$

So, for the entire system,

$$
\begin{aligned}
M(t) &= (1 - e^{-\mu_1 t}) \times (1 - e^{-\mu_2 t}) \times (1 - e^{-\mu_3 t}) \times \ldots \times (1 - e^{-\mu_n t}) = (e^{-\sum_{i=1}^{n}\mu_i t}) \\
MTTR_{SS} &= \frac{1}{\sum_{i=1}^{n}\mu_i},
\end{aligned}
\tag{6.6}
$$

where μ_i is the repair rate of each system component. So, the sub-system $MTTR_{SS}$ may be estimated by the Equation 6.7.

$$
MTTR_{SS} = \frac{1}{\mu_{HW} + \mu_S + \mu_{OS} + \mu_{CT} + \mu_{APP}} \tag{6.7}
$$

The top-level model representing the whole system, i.e., the joint behavior of the cache VNF and the load balancer, was modeled through a CTMC. This hierarchical ans heterogeneous model is represented in Figure 6.3. The system will be available when at least one cache VNF sub-system is working and the load balancer is working.

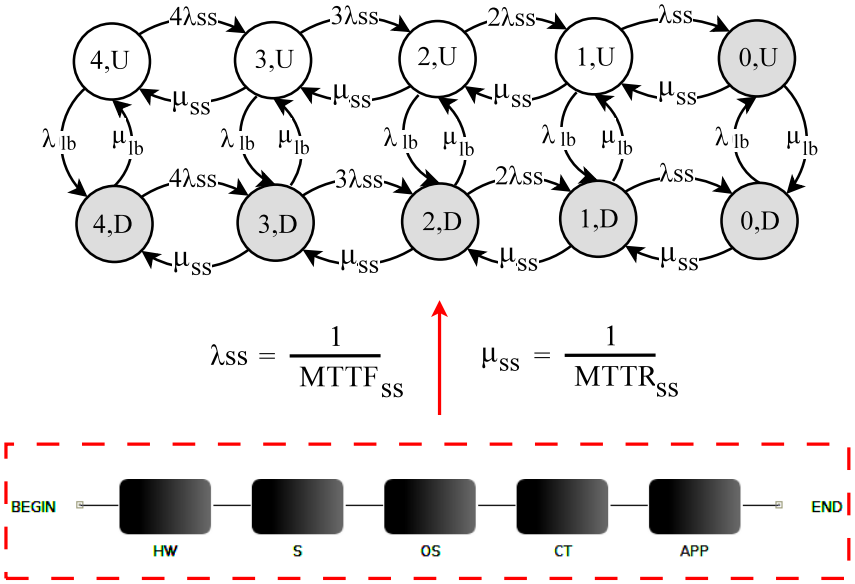


Figure 6.3: Hierarchical Composition: top-level CTCM of system's availability model; bottom level RBD of cache node sub-system model

We adopted a notation for the CTMC model states based on the current condition of each component. The first character represents the number of working (up) cache VNFs. The second character represents the state of the load balancer: up (U) or down (D). We shaded the states in which the system is down.

The system is at full capacity at state $4,U$. From this state, the failure of one working cache VNF sub-system brings the system to state $3,U$, at a $4\lambda_{SS}$ rate. From this latter state, when a failure occurs in load balancer at λ_{lb} rate, the system goes to inactive state $3,D$. From this state, the system can be repaired at μ_{lb} rate, returning to working $3,U$ state. Likewise, from $3,U$ state, the failed cache VNF sub-system can be repaired at mu_{SS} rate, and the system returns to its full capacity at state $4,U$. The remaining states and transitions have a similar understanding. The sub-system failure rate λ_{SS} and repair rate $4\mu_{SS}$, obtained through low-level RBD sub-system, are inserted in the top-level CTMC, as showed by expressions $\lambda_{SS} = 1/MTTF_{SS}$ and $\mu_{SS} = 1/MTTR_{SS}$ in Figure 6.3.

The SSA and COA of Figure 6.3 model are given respectively by:

$$SSA = \pi_{4,U} + \pi_{3,U} + \pi_{2,U} + \pi_{1,U}, \tag{6.8}$$

and

$$COA = \frac{\sum_{i=1}^{4} i \times \pi_{(i,U)}}{4}, \tag{6.9}$$

where i is the number of available cache sub-systems and $\pi_{(i,U)}$ is the long-run steady-state probability of i cache sub-systems and a load balancer were available.

6.2.3 Model for Cache VNF and Load Balancer without SPOFs

With the aim of claim conformance with the elimination of Single Points Of Failure (SPOF), defined in the resiliency objective of European Telecommunications Standards Institute (ETSI) [37], a redundant load balancer is represented by one additional character U or D in the model states depicted in Figure 6.4. As a result, the new model is five states bigger than the first CTMC.

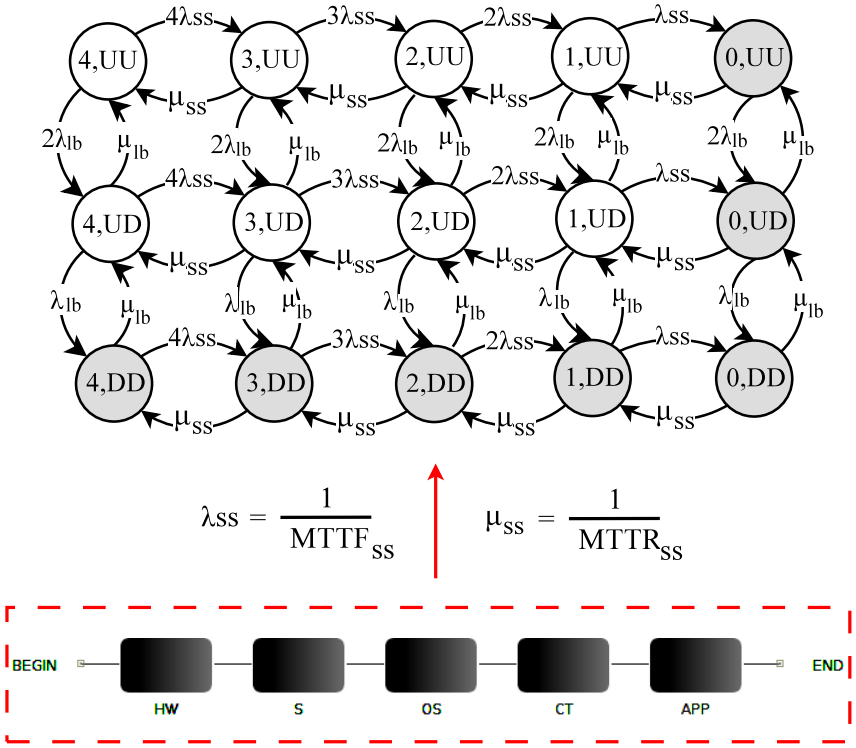


Figure 6.4: Hierarchical Composition: no load balancer SPOF

The SSA and COA for this scenario are given respectively by:

$$SSA = \sum_{i=1}^{4} \pi_{i,UU} + \pi_{i,UD}, \tag{6.10}$$

and

$$COA = \frac{\sum_{i=1}^{4} i \times (\pi_{i,UU} + \pi_{i,UD})}{4}, \tag{6.11}$$

where i is the number of available cache sub-systems and $\pi_{i,UU}$ and $\pi_{i,UD}$ is the long-run steady-state probability of i cache sub-systems were available with both or one load balancer, respectively. Again, the sub-system failure rate λ_{SS} and repair rate μ_{SS}, obtained through low-level RBD sub-system, are inserted in the top-level CTMC.

6.3 Models for SFCs in openstack cloud infrastructure

Our work adopts regarding SFC in openstack cloud infrastructure adopts time-based live migration rejuvenation with conditions that may postpone migration, leveraging dependability attributes. The rejuvenation is performed proactively through preventive maintenance. Figure 6.5 exhibits the system architecture in which the proposed rejuvenation is applied.

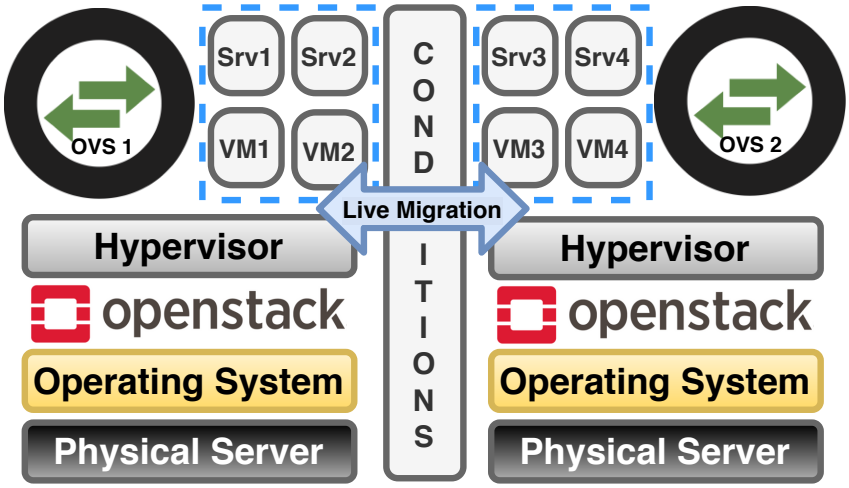


Figure 6.5: Architecture for nodes providing VNFs: the software rejuvenation is implemented by VM live migration and conditions

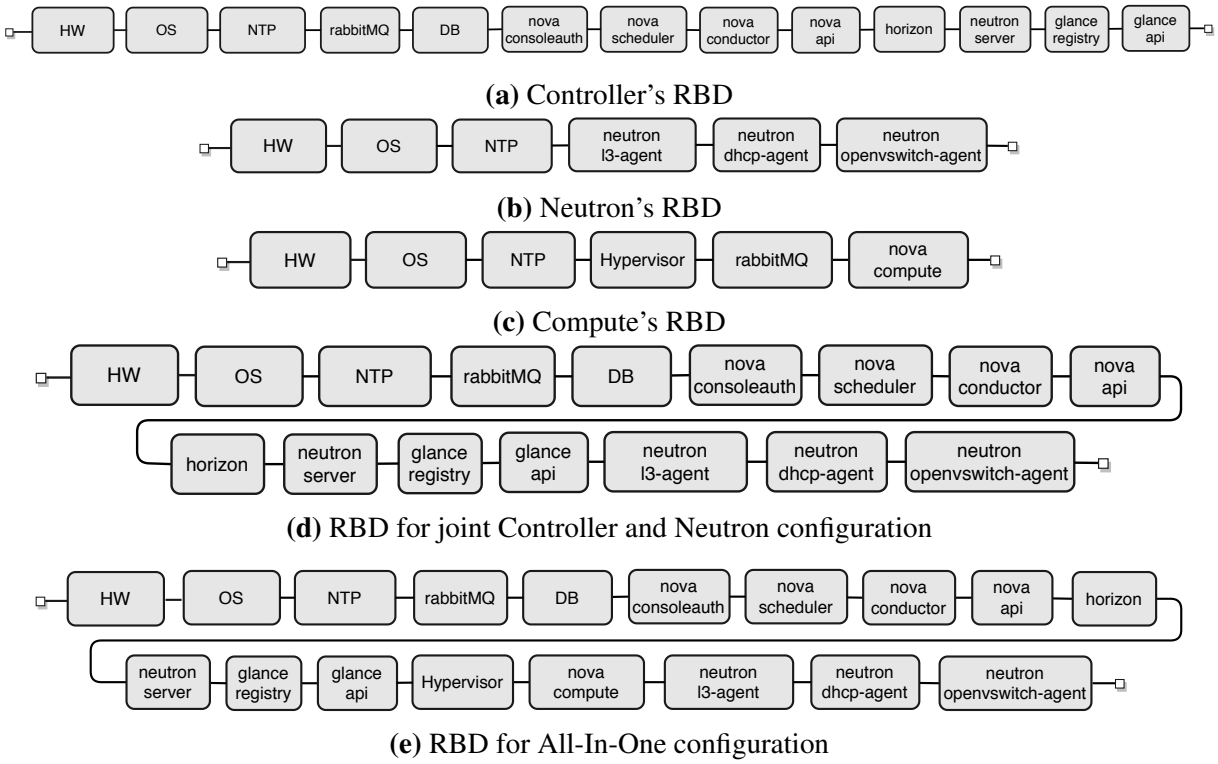
In Figure 6.5, two physical servers are presented. They host their own operating system (OS), the openstack services, the hypervisor, the VMs running its services, and the Open vSwitch. The VNFs are provided by each (VM_i,Srv_i) pair. Our models aim at representing the hardware and software components of Figure 6.5.

Regarding HA openstack cloud infrastructure, its services and the SFCs are modeled using RBDs. The interaction among nodes, the service chains, and their interconnections, as well as the aging and rejuvenation phenomena, are modeled using SPN. The presented modeling is generic and can be used in any cloud infrastructure.

6.4 Low-level RBDs for openstack deployment modes

We model the openstack services (as presented in Figures 2.5, and 2.6) as low-level RBDs, as depicted in Figure 6.6.

By using these RBDs, one can estimate the MTTF and/or MTTR of each openstack server configuration. These values will also be injected as input parameters in the top-level SPN models. The Figures. 6.6a, 6.6b, and 6.6c correspond to openstack servers roles provided in distinct nodes. Figure 6.6d depicts the joint roles of controller and neutron provided in one single Node, whereas Figure 6.6e depicts the RBD for All-In-One assembling, in which controller, neutron, and compute roles are provided in one physical server.



(a) Controller's RBD

(b) Neutron's RBD

(c) Compute's RBD

(d) RBD for joint Controller and Neutron configuration

(e) RBD for All-In-One configuration

Figure 6.6: Low-level RBDs models: the MTTF of each low-level sub-system is computed and injected in the top-level SPN models

Table 6.2 report the input parameters adopted in the low-level sub-models representing openstack deployment modes (Figure 6.6).

Table 6.2: Dependability parameters for low-level

Parameter	Description
$MTTF_{SRV}$	Server Mean Time To Failure
$MTTR_{SRV}$	Server Mean Time To Repair
$MTTF_{OS}$	Operating System Mean Time To Failure
$MTTR_{OS}$	Operating System Mean Time To Repair
$MTTF_{HV}$	Hypervisor Mean Time To Failure
$MTTR_{HV}$	Hypervisor Mean Time To Repair
$MTTF_{DB}$	Database Mean Time To Failure
$MTTR_{DB}$	Database Mean Time To Repair
$MTTF_{RS}$	Required Services Mean Time To Failure
$MTTR_{RS}$	Required Services Mean Time To Repair

6.5 Low-level RBDs for service function chains

The Service Function Chains are modelled as series RBD, as depicted in Figure 6.7. A failure in any component results in the chain failure, regardless if the failure occurs in a VM or

in a service being provided in its correspondent VM.

Figure 6.7: Low-level RBD for Service Function Chains

Similarly to low-level RBDs of openstack deployment server roles, it is also possible to estimate the MTTF and/or MTTR of each chain represented by an RBD.

6.6 Top-level SPN sub-models for openstack nodes

For Service Function Chains provided in openstack infrastructure, besides redundancy, we also consider the aging phenomenon as well as the rejuvenation countermeasure mechanism. Furthermore, we also modeled the service chain without rejuvenation to be used as a baseline comparison. The aim was to state the impact of aging and rejuvenation process in the service chain availability and capacity.

6.6.1 SPN sub-model for openstack nodes without rejuvenation

Figure 6.8 depicts the SPN that represents openstack's nodes without rejuvenation. The presence of a token at place *Nup* indicates the working state of an openstack node.

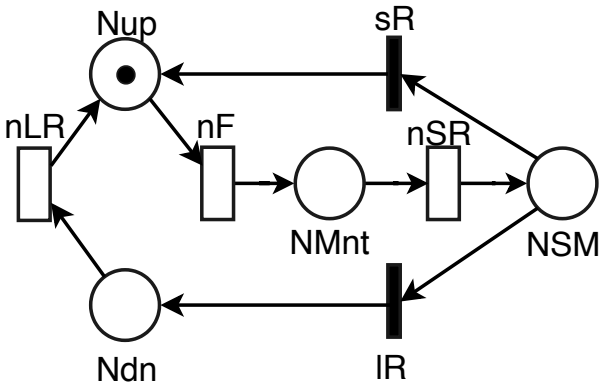


Figure 6.8: SPN sub-model for node

The expression $P\{(\#Nup > 0)\}$, meaning the number of tokens probability at place *Nup* is greater than zero, is used to estimate the component's availability. Only the *nF* transition can fire at the beginning, representing a node failure. When *nF* fires, it deposits a token at *NMnt* place. At *NMnt*, a fast repair representing a simple node reboot is performed by the *nSR* firing, depositing a token at place *NSM*. From *NSM*, there are two alternatives: *(i)* the fast repair solves the issue, then the *sR* is fired with a certain probability *psR*, and the token is deposited back at working place *Nup*; *(ii)* the fast repair procedure is unsuccessful, then *lR* is fired with a $(1 - psR)$ probability, the token reaches *Ndn* place and a troubleshooting repair, represented by

the transition *nLR*, is performed. When *nLR* fires, the token is deposited back at working place Nup.

The nodes are modeled with redundancy using two or more tokens at place Nup. As all nodes may fail simultaneously, the transition *nF* has infinite server semantic(iss) [17] concurrency. Table 6.3 presents the descriptions of the transitions applied in the Node SPN, whereas Table 6.4 describes the transitions attributes for that SPN. The models' transitions can be tuned to represent service repair policies. So, it is possible to analyze conditions where the average repair time is shorter or longer, considering service repair policies.

Table 6.3: Dependability parameters for Node SPN

Parameter	Description
$MTTF_N$	Node Mean Time To Failure
$MTTR_N$	Node Mean Time To Repair
TRBT	Node Mean Time To Reboot
sR	Short Repair
lR	Large Repair

Table 6.4: Transitions attributes for Node SPN

Transition	Type	Semantics	Weight	Priority
nF	Timed	Infinite Server	-	-
nSR	Timed	Single Server	-	-
sR	Immediate	-	psR	1
lR	Immediate	-	1-psR	1
nLR	Timed	Single Server	-	-

6.6.2 SPN sub-model for openstack nodes with rejuvenation

Figure 6.9 depicts the SPN sub-model for the openstack nodes adopting rejuvenation.

We used a k-phases Erlang sub-net to model the node aging phenomenon. This proposed sub-net prevents the immediate transition to a failure state after a single firing, adequately modeling the aging phenomenon. For an aging time interval of $nodeMTTF$, each of the k Erlang phases is exponentially distributed with mean $nodeMTTF/k$. The places $Ndup$ and $Nddn$ represent the working and the failure states, respectively. When $ndAg1$ fires, the $Ndup$ token is taken from this place, and $(k-1)$ tokens are deposited at place $Ag1nd$, representing the first aging period. The transition $ndAg1$ also redeposits a token at $Ndup$, as the node remains working. So, a node reaches failure $Nddn$ place, only after $ndAg2$ fires (k-1) times, representing the remaining aging periods. When there are $(k-1)$ tokens at place $Ag2nd$, the immediate transition dNd fires, the $(k-1)$ tokens are taken from $Ag2nd$, the $Ndup$ token is also taken, and one token is deposited at place $Nddn$, representing the node failure. The rejuvenation will occur when the service chain migrates from a node to another. After the VMs migration, all the

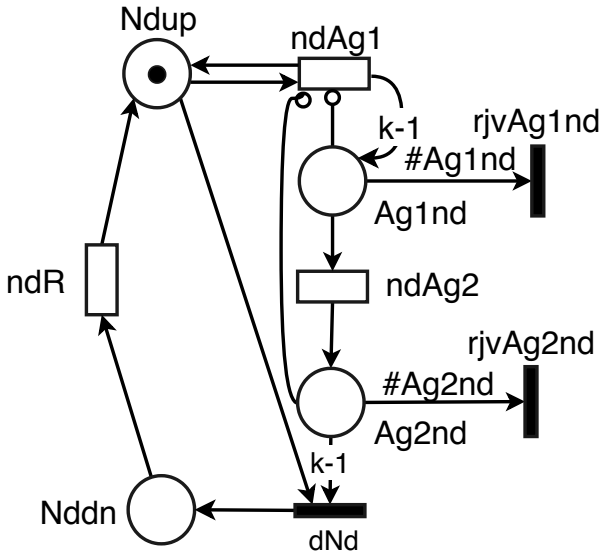


Figure 6.9: SPN sub-model for nodes with rejuvenation

tokens at places $Ag1nd$ and $Ag2nd$ will be consumed by the immediate transitions $rjvAg1nd$ and $rjvAg2nd$, respectively, completing the rejuvenation. The inhibitor arcs from the places $Ag1nd$ and $Ag2nd$ to the transition $ndAg1$ avoid $ndAg1$ to fire after the start of the aging process.

If one needs to represent redundancy of openstack nodes, it is just required to add two or more sub-models presented in Figure 6.9. Tables 6.5 and 6.6 report the parameters and attributes of the transitions belonging to SPN of Figure 6.9.

Table 6.5: Dependability parameters for Node SPN with Rejuvenation

Parameter	Description
$MTTF_N$	Node Mean Time To Failure
$MTTR_N$	Node Mean Time To Repair
RJV_N	Node Rejuvenation

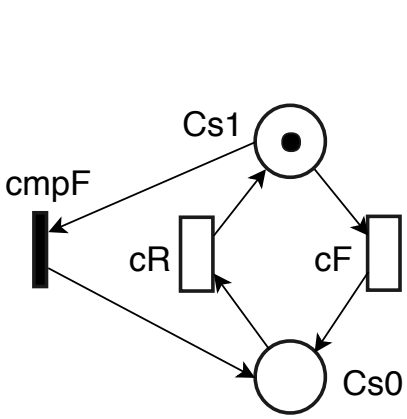
Table 6.6: Transitions attributes for Node SPN with Rejuvenation

Transition	Type	Semantics	Weight	Priority
ndAg1	Timed	Single Server	-	-
ndAg2	Timed	Single Server	-	-
rjvAg1nd	Immediate	-	1	1
rjvAg1nd	Immediate	-	1	1
dNd	Immediate	-	1	1

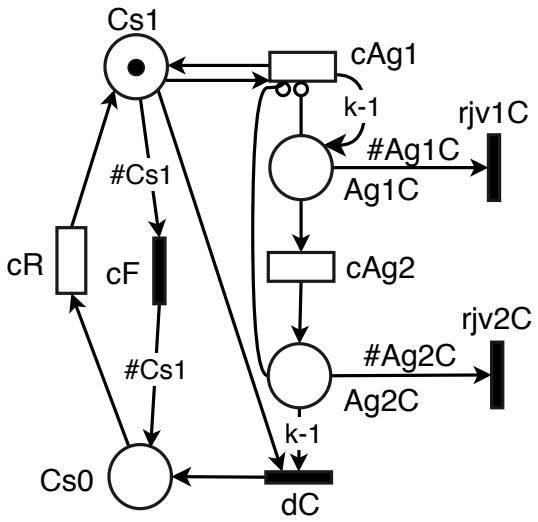
6.7 Top-level SPN sub-models for service chain

Figure 6.10a depicts the chain sub-model without rejuvenation. Initially, two distinct transitions can be fired: *(i)* the cF transition fires to represent any failure in a VM or in a service

belonging to the chain, depositing the token at place *Cs*0; and *(ii)* the *cmpF* fires to represent a failure in the compute node hosting the chain, depositing the *Cs*1 token at *Cs*0. The chain repair is represented by the firing of the *cR* transition, redepositing the token at working place *Cs*1.



(a) SPN for service chain without rejuvenation



(b) SPN for service chain with rejuvenation

Figure 6.10: SPN sub-models for service chains

On the other hand, Figure 6.10b depicts the chain sub-model considering aging and rejuvenation. It has a similar behavior of node sub-model (Figure 6.9). After a chain migration (see Section 6.8), the place *Cs*1 will contain two tokens representing two service chains. The immediate transition *cF* was added to model the failure of the compute node hosting the service chain. It fires whenever the compute node fails, depositing the token at place *Cs*0.

Tables 6.7 and 6.8 report the parameters and attributes of the transitions belonging to Chain SPN (Figure 6.10a).

Table 6.7: Dependability parameters for Service Chain SPN

Parameter	Description
$MTTF_C$	Chain Mean Time To Failure
$MTTR_C$	Chain Mean Time To Repair
$DACT_C$	Chain Deactivation

Table 6.8: Transitions attributes for Service Chain SPN

Transition	Type	Semantics	Weight	Priority
cR	Timed	Single Server	-	-
cF	Timed	Single Server	-	-
cmpF	Immediate	-	1	1

Following, Tables 6.9 and 6.10 report the parameters and attributes of the transitions belonging to Chain SPN with rejuvenation (Figure 6.10b).

Table 6.9: Dependability parameters for Service Chain SPN with Rejuvenation

Parameter	Description
$MTTF_C$	Chain Mean Time To Failure
$MTTR_C$	Chain Mean Time To Repair
RJV_C	Chain Rejuvenation
$DACT_C$	Chain Deactivation
$DACT_N$	Node Deactivation

Table 6.10: Transitions attributes for Service Chain SPN with Rejuvenation

Transition	Type	Semantics	Weight	Priority
cAg1	Timed	Single Server	-	-
cAg2	Timed	Single Server	-	-
rjv1C	Immediate	-	1	1
rjv2C	Immediate	-	1	1
dC	Immediate	-	1	1
cF	Immediate	-	1	1

6.8 Top-level SPN sub-model for service chain live migration

Figure 6.11 exhibits the SPN sub-model for service chain live migration. There are two chains in operation, represented by the tokens at places $C1s1$ and $C2s1$. The transition $ch1m2$ fires, with an $MTBPM$, takes the $C1s1$ token and depositing it at $m12$. As soon as the migration conditions in $mC1C2$ are satisfied, the $mC1C2$ transition becomes enabled and fires with an $MTTCLM + MTTPPM$ mean time. The transition $mC1C2$ adds a token at $C2s1$, completing the chain migration. As soon as the source node is rejuvenated, $mC2C1r$ is enabled and fires with an $MTTCLM$ mean time, depositing one token from $C2s1$ at $C1s1$. Similar behavior occurs when the preventive maintenance starts at $C2s1$ and continues with the firing of transitions $ch2m1$, $mC2C1$, and $mC1C2r$.

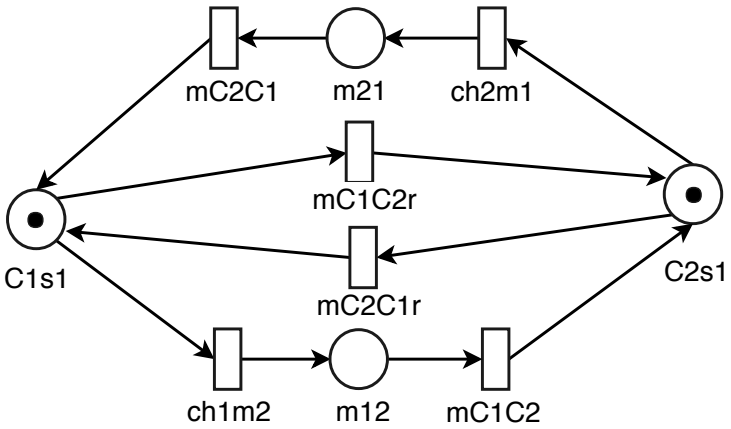


Figure 6.11: SPN sub-model for VNF chain live migration

Tables 6.11 and 6.12 report the parameters and attributes of the transitions belonging to Chain Live Migration SPN (Figure 6.11).

Table 6.11: Dependability parameters for Chain Live Migration SPN

Parameter	Description
MTBPM	Mean Time Between Preventive Maintenance
MTTPPM	Mean Time To Perform Preventive Maintenance
MTTCLM	Mean Time To Chain Live Migration

Table 6.12: Transitions attributes for Chain Live Migration SPN

Transition	Type	Semantics	Weight	Priority
ch1m2	Timed	Single Server	-	-
mC1C2	Timed	Single Server	-	-
mC2C1r	Timed	Single Server	-	-
ch2m1	Timed	Single Server	-	-
mC2C1	Timed	Single Server	-	-
mC1C2r	Timed	Single Server	-	-

6.9 Top-level SPN sub-models for chains interconnection

The chain interconnection sub-model represents the Open vSwitch containing the open-stack SFC API rules. Such rules are required to redirect the packet flow through the service chain. The sub-model behavior of Figure 6.12 is equal to the service chain sub-model (Figure 6.10a) for non-rejuvenation approach.

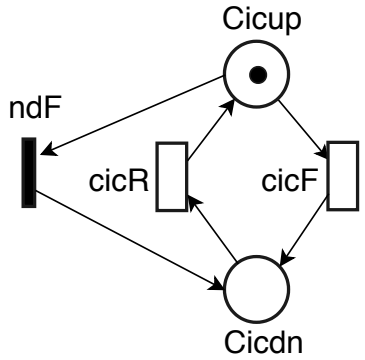


Figure 6.12: SPN for chain interconnection without rejuvenation

Tables 6.13 and 6.14 report the parameters and attributes of the transitions belonging to Chain Interconnection SPN (Figure 6.12).

Table 6.13: Dependability parameters for Chain Interconnection SPN

Parameter	Description
$MTTF_{CI}$	Chain Interconnection Mean Time To Failure
$MTTR_{CI}$	Chain Interconnection Mean Time To Repair
$DACT_{CI}$	Chain Interconnection Deactivation

Table 6.14: Transitions attributes for Chain Interconnection SPN

Transition	Type	Semantics	Weight	Priority
cicR	Timed	Single Server	-	-
cicF	Timed	Single Server	-	-
ndF	Immediate	-	1	1

Regarding chains interconnection sub-models applied in rejuvenation scenarios, we adopted two SPN sub-model, depicted in Figures 6.14 and6.13. In the first sub-model, the transition *cicD* can fire, enabling the service chain migration, depositing the *Cicup* token at place *CicMnt* (the *Mnt* suffix means maintenance), indicating the deletion of SFC API rules and, as a consequence, the interruption of Open vSwitch rules required for the packets flow routing throughout the chain.

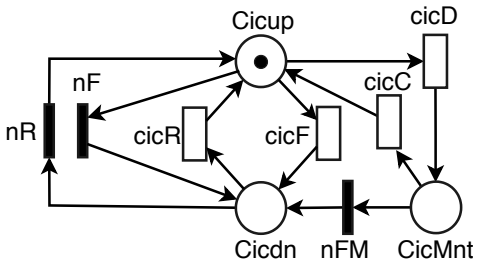


Figure 6.13: SPN for chain interconnection adopted in rejuvenation model: the chain interconnection without SAR

At *CicMnt*, two alternatives are possible: *(i)* the transition *cicC* fires, representing the recreation of SFC API rules, after the service chain migration; *(ii)* the *nFM* fires, indicating the failure of the compute node that is hosting the service chain, depositing the *CicMnt* token at place *Cicdn*. As soon as the compute node is recovered, the *nR* transition fires, depositing the *Cicdn* token at working place *Cicup*. The mean times adopted in the *cicD* and *cicC* exponential transitions were obtained from the measurement experiments, presented in Chapter 5. In addition, the immediate transition *nFM* will fire if during a live migration rejuvenation process, the Node containing the Open vSwitch fails. The token is taken from place *CicMnt* and is deposited in place *Cicdn*.

Tables 6.15 and 6.16 report the parameters and attributes of the transitions belonging to Chain Interconnection SPN with Rejuvenation, adopted in the First Scenario (Figure 6.13).

Table 6.15: Dependability parameters for Chain Interconnection SPN with Rejuvenation - First Scenario

Parameter	Description
$MTTF_{CI}$	Chain Interconnection Mean Time To Failure
$MTTR_{CI}$	Chain Interconnection Mean Time To Repair
$MTTD_{CI}$	Chain Interconnection Mean Time To Deletion
$MTTC_{CI}$	Chain Interconnection Mean Time To Creation
$DACT_{CI}$	Chain Interconnection Deactivation
$RACT_{CI}$	Chain Interconnection Reactivation

Table 6.16: Transitions attributes for Chain Interconnection SPN with Rejuvenation - First Scenario

Transition	Type	Semantics	Weight	Priority
cicR	Timed	Single Server	-	-
cicF	Timed	Single Server	-	-
cicD	Timed	Single Server	-	-
cicC	Timed	Single Server	-	-
nF	Immediate	-	1	1
nR	Immediate	-	1	1
nFM	Immediate	-	1	1

In the second sub-model, we consider that the proper chain interconnection software, i.e., the Open vSwitch, ages. So, in Figure 6.14 one can observe a similar SAR behavior as one adopted for service chain (Figure 6.10b).

Tables 6.17 and 6.18 report the parameters and attributes of the transitions belonging to Chain Interconnection SPN with Rejuvenation, adopted in the Second Scenario (Figure 6.14).

Table 6.17: Dependability parameters for Chain Interconnection SPN with Rejuvenation - Second Scenario

Parameter	Description
$MTTF_{CI}$	Chain Interconnection Mean Time To Failure
$MTTR_{CI}$	Chain Interconnection Mean Time To Repair
$MTTD_{CI}$	Chain Interconnection Mean Time To Deletion
$MTTC_{CI}$	Chain Interconnection Mean Time To Creation
RJV_{CI}	Chain Interconnection Rejuvenation
$DACT_{CI}$	Chain Interconnection Deactivation
$RACT_{CI}$	Chain Interconnection Reactivation

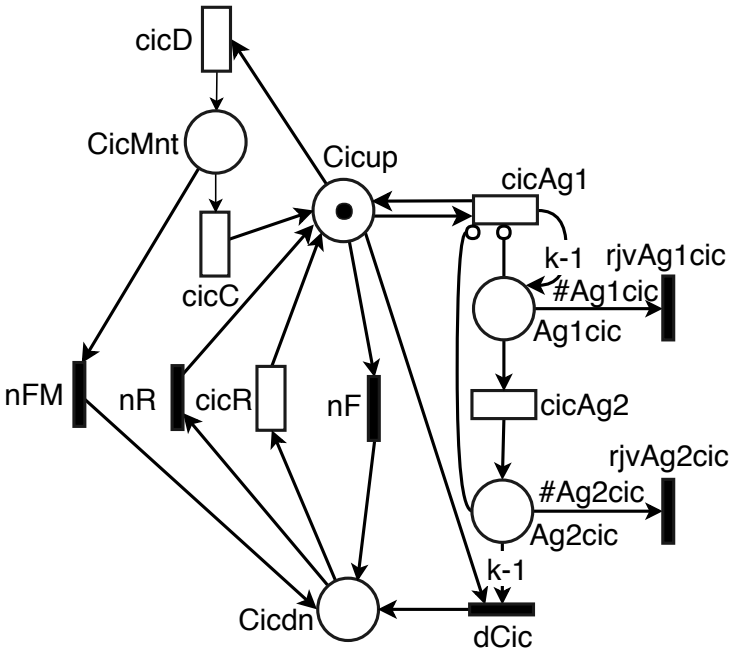


Figure 6.14: SPN for chain interconnection with rejuvenation

Table 6.18: Transitions attributes for Chain Interconnection SPN with Rejuvenation - Second Scenario

Transition	Type	Semantics	Weight	Priority
cicR	Timed	Single Server	-	-
cicAg1	Timed	Single Server	-	-
cicAg2	Timed	Single Server	-	-
cicD	Timed	Single Server	-	-
cicC	Timed	Single Server	-	-
nF	Immediate	-	1	1
nR	Immediate	-	1	1
nFM	Immediate	-	1	1
dCic	Immediate	-	1	1

6.10 Final Remarks

This chapter presented a series of dependability models that enable the evaluation of steady-state availability and capacity oriented availability of VNF chains and Service Function chains considering redundancy and software rejuvenation mechanisms. They are used in the case studies presented in the next chapter, where several scenarios are studied with the aim to state which infrastructures can provide high available VNF chains and service function chains environments.

7

Case Studies

7.1 Introduction

This chapter presents practical experiments divided into five cases studies. Preliminary, two case studies analyze the Steady-State Availability and Capacity Oriented Availability of cache VNFs and load balancers provided in Proxmox server virtualization environment.

The third case study aims at analyzing how reasonable is our proposed modeling approach and our results to estimate the availability of Service Function Chains in openstack cloud through a comparison with previous literature. The fourth case study analyzes the benefits of the rejuvenation adoption in a 3N redundant SFC provided in openstack cloud. Finally, the fifth case study analyzes the behavior of SSA and COA facing the reduction for a 2N redundant environment in an openstack cloud.

7.2 VNF Cache Clusters

The analyzed scenarios of this first case study are presented in Table 7.1. Along with the first case study, we adopted failure and repair rates mentioned in literature (exhibited in Table 7.2) and from conduced stressing experiments (see Section 5.4).

Table 7.1: Scenarios of First Case Study

Scenario	Model
1	Non-Redundant VNF Cache Cluster
2	Redundant VNF Cache Cluster

In order to compute the measures of interest, we used Mercury Tool [109]. Mercury provides a Graphical User Interface (GUI) for intuitive modeling of RBDs. As RBDs provide closed-form equations, Mercury solves them and presents the computed metrics.

In addition to values listed in Table 7.2, the MTTF of 315.33 seconds for cache servers were obtained from previous experiments (see Section 5.4) and an MTTR of 10 seconds was adopted. It is a sufficient time interval to restart the cache cluster services. These values are inserted in hours on Mercury, as presented in Table 7.3.

Table 7.2: Applied times in the RBD models

Component	MTTF	MTTR
Server	8760h	1.67h
S	4380h	5 min
OS	2893h	0.25h
CT	2990h	1h

Table 7.3: MTTF and MTTR, in hours, for cache servers

Component	MTTF	MTTR
APP	0.0875925926h	0.0027777778h

7.2.1 Non-redundant VNF Cache Cluster

This scenario represents the utilization of only two containers running in one node of Proxmox Server Virtualization. It was modeled using one RBD for VNF Cache Cluster (Figure 6.1). It is representation can be observer in Figure 7.1.

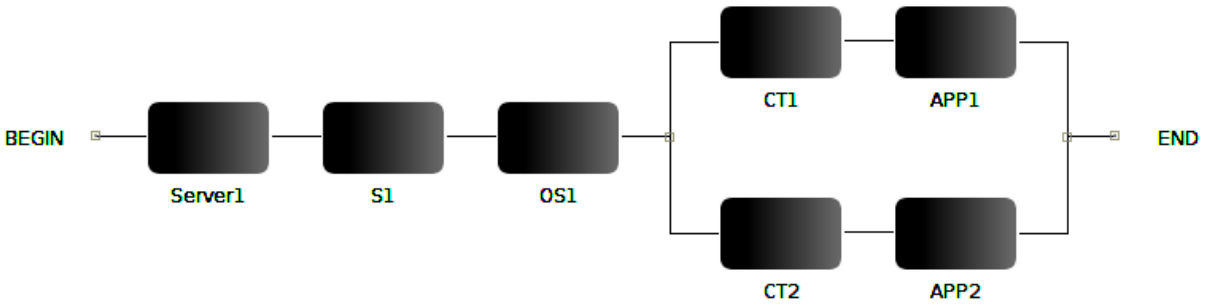


Figure 7.1: RBD for Non-Redundant Cluster

The steady-state availability, number of 9's, and annual downtime were the measures of interest for this scenario. The results are exhibited in Table 7.4.

Table 7.4: Availability measures for Non-redundant VNF Cache Cluster

Measure	Results
SSA (%)	99.87394%
SSA (Number of 9's)	2.89943
Annual Downtime	11.049853h

Following the classification of Table 2.2, two 9's of availability not even result in a fault-tolerant system type, with more than 10 hours of downtime per year, or 50 minutes per month. Due to users requirements of telecommunication service providers, such downtime is not suitable for NFV compatible infrastructures.

Aiming at reveal the most impactful parameters over SSA, a parametric sensitivity analysis was conducted. The closed-form equation for availability of non-redundant VNF cluster is expressed as:

$$SSA = A_{Server1} \times A_{S1} \times A_{OS1} \times (1 - (1 - A_{CT1} \times A_{APP1}) \times (1 - A_{CT2} \times A_{APP2})) \quad (7.1)$$

This closed-form equation was used to obtain the partial derivatives and compute the complete sensitivity ranking, as described in [80]. Table 7.5 shows the results, where the parameters are presented in decreasing order of the sensitivity index.

Table 7.5: Ranking of Sensitivities for SSA of Non-Redundant VNF Cluster

Parameter	S(SSA)
$Server_1$	0.998929836
OS_1	0.998825744
S_1	0.998758439
APP_n	0.031042188
CT_n	0.030098085

Each value in the ranking of sensitivities represents how changes in any particular block affect systems' steady-state availability.

The ranking points Server component as the most important when A is the measure of interest, nearly followed by OS and S components. The results also show that APP and CT are more than thirty times less influential than the other components. So, Server, OS, and S should receive priority when improvements to the system availability are considered on the NFV video cache cluster.

7.2.2 Redundant VNF Cache Cluster

Motived by the results of sensitivity analysis in the first scenario, as Server, OS, and S are the most important components regarding availability improvements, we replicated them. Clearly, a directed advantage of such components replication is the possibility of insert CT and APP blocks, as they model software components running under Server, OS, and S replicas. Thus, in order to improve availability, we propose the implementation of active-active parallel redundancy for VNF cache service, representing the utilization of four containers running in two nodes of Proxmox Server Virtualization testbed (2 containers per node). The resulting RBD model is shown in Figure 7.2, where servers 1 and 2 were connected in parallel to represent the proposed active-active redundancy.

Analysis results are summarized in Table 7.6.

Table 7.6: Availability measures of Redundant VNF Cache Cluster

Measure	Result
SSA (%)	99.99984%
SSA (Number of 9's)	5.79887
Annual Downtime	0.013929h

The Steady-State Availability was improved to 99.99984%, meaning an annual downtime below 5 minutes. According to the ranking of Table 2.2, the high availability classification was

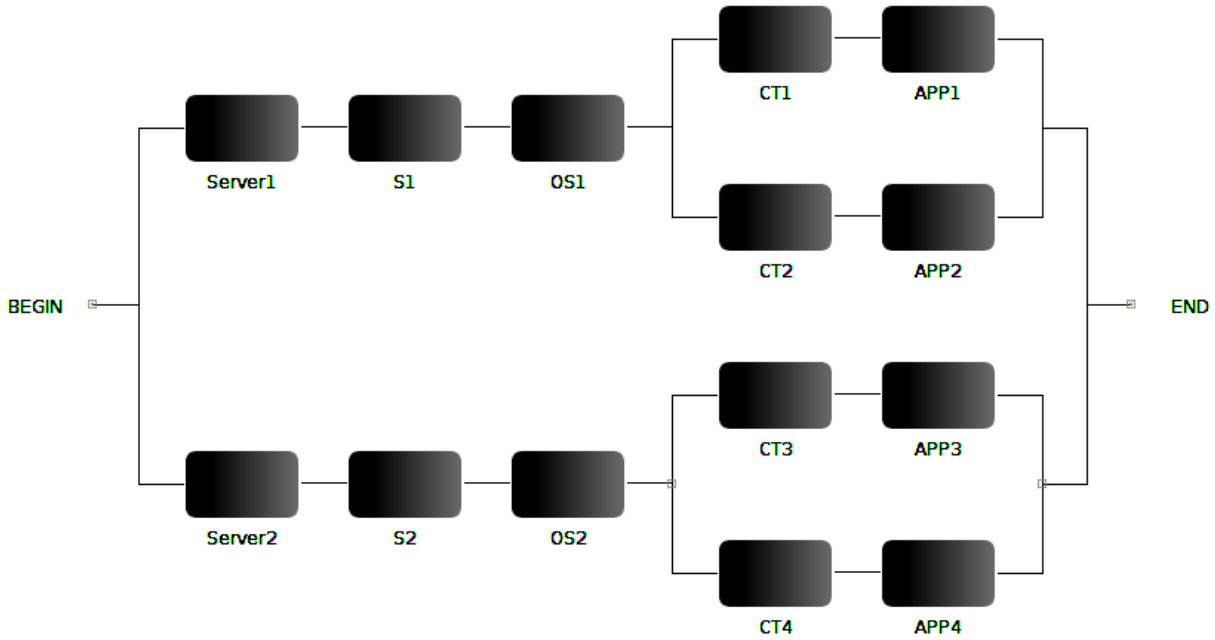


Figure 7.2: RBD for Redundant VNF Cache Cluster

reached. Such an increase in availability, in comparison with a simplex scheme, is an evidence of the improvements in the adoption of the proposed active-active redundancy strategy.

Regarding sensitivity analysis, the closed-form equation for Steady-Sate Availability of redundant cluster is expressed as:

$$\begin{aligned} SSA = 1 - (1 - A_{Server1} \times A_{S1} \times A_{OS1} \times (1 - (1 - A_{CT1} \times A_{APP1}) \times (1 - A_{CT2} \times A_{APP2}))) \\ \times (1 - A_{Server2} \times A_{S2} \times A_{OS2} \times (1 - (1 - A_{CT3} \times A_{APP3}) \times (1 - A_{CT4} \times A_{APP4}))) \end{aligned} \quad (7.2)$$

The results of parametric sensitivity analysis are presented in decreasing order of the sensitivity index in Table 7.7.

Table 7.7: Ranking of Sensitivities for SSA in Redundant VNF Cluster

Parameter	S(SSA)
$Server_n$	0.0012592133431
OS_n	0.0012590821281
S_n	0.0012589972864
APP_n	0.0000391306135
CT_n	0.0000379278885

The ranking shows that the Server components remains as the most relevant component when A is the measure of interest, again followed by OS and S components. However, in comparison with the non-redundant scenario, the failure of these components had less influence on availability, as sensitivity coefficients are smaller due to components replication.

7.3 Redundant VNF Caches and Load Balancers

In this section, we analyzed the models presented in Sections 6.2.2 and 6.2.3. They are summarized in Table 7.8. In this case study, besides rates mentioned in literature (previously reported in Table 7.2), estimated guesses were applied for VNF cache application.

Table 7.8: Scenarios of Second Case Study

Scenario	Model
1	Redundant VNF Cache and Load Balancer
2	Redundant VNF Cache and Load Balancer without SPOFs

Hierarchical and heterogeneous models were adopted, using RBD in the low-level and Continuous Time Markov Chains (CTMC) in the top-level. We also evaluated Capacity Oriented Availability (COA) in this scenario.

7.3.1 Redundant VNF Cache and Load Balancer

The analysis results for the system composed by VNF caches and one load balancer are presented in Table 7.9.

Table 7.9: Steady-State Availability and COA for CTCM

$MTTF_{APP}$(h)	SSA	COA
4380	99.9520205%	99.8851661%
8760	99.9520180%	99.8870576%
13140	99.9520185%	99.8876941%
26280	99.9520190%	99.8882247%
43800	99.9520192%	99.8885130%
87600	99.9520193%	99.8887930%

The estimated guesses for $MTTF_{APP}$ presented in the first column of Table 7.9 correspond to six months, one year, one year and six months, three years, five years, and ten years.

The application MTTF variability did not result in a significant impact over Steady-State Availability: the resulting difference among MTTFs over SSA is smaller than 1 second, all of them with an Unavailability (UA) nearly *4h12m11s* per year, or 252.18 minutes (1 year is equivalent to 525,600 minutes). Regarding Capacity Oriented Unavailability (COUA), the difference between $COUA_{MTTF_{APP}=87600}$ and $COUA_{MTTF_{APP}=4380}$ was *1h16m12s*.

In order to compare the source of downtime minutes, consider, for instance, the scenario with $MTTF_{APP}$=87600h, in which the system is up during 99.9520193% of the time. Consequently, UA=0.0479807%. The product of UA, one year of functional minutes and the number of cache sub-systems ($0.000479807 \times 525600 \times 4 = 1008.74$) reveals that the VNF cache cluster can be expected to be out of operation during 1008.74 minutes. Similarly, as

$COA_{MTTF_{APP}=87600}$=99.8887930%, $COUA_{MTTF_{APP}=87600}$=0.0526357%, and 2338.01 of cache-minutes were not delivered. This downtime consists of 1008.74 cache-minutes of system downtime (UA) and the remaining $2338.01 - 1008.74 = 1329.27$ cache-minutes of downtime due to degraded capacity (COUA).

The percentage of downtime due to UA and COUA for all considered $MTTF_{APP}$ can be observed in Figure 7.3.

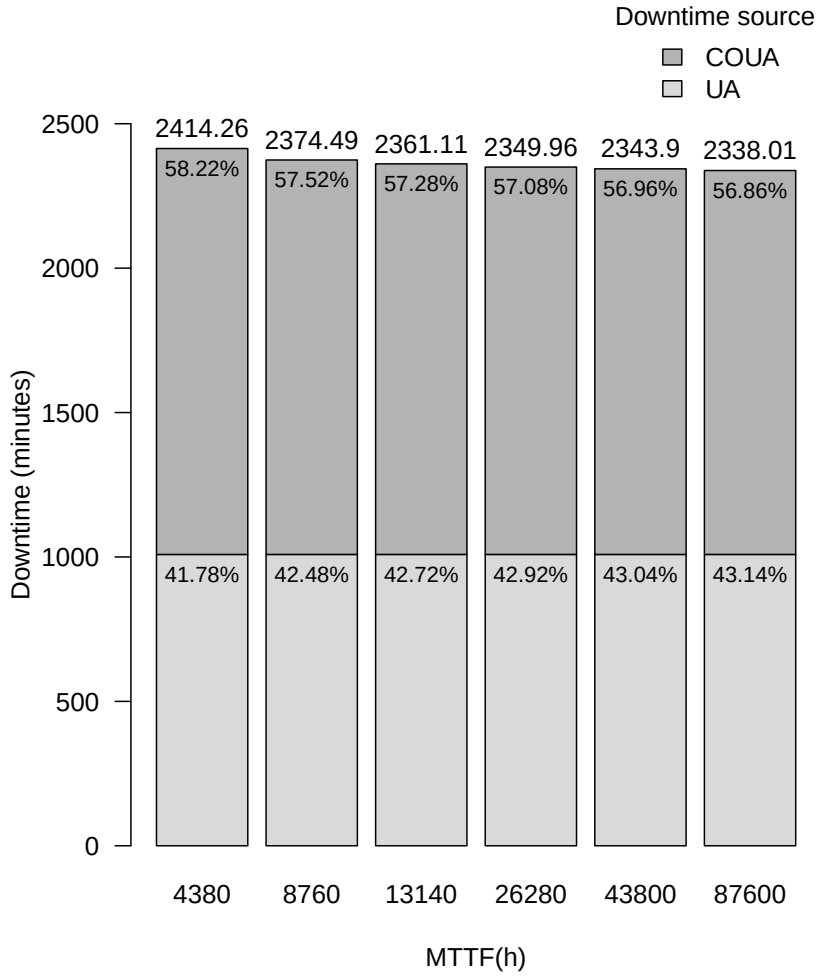


Figure 7.3: Percentage of annual downtime due to UA and COUA

With the aim to state which of the VNF cache system parameters are most influential in the presented downtime, we proceeded with the SSA and COA sensitivity analysis for the top-level CTMC of the hierarchical model. Closed-form formulas were obtained using StateDiagram Software Package available for Mathematica [128].

The insertion of CTMC infinitesimal generator matrix enables StateDiagram Package to obtain the respective closed-form formulas for SSA and COA:

$$SSA = \frac{\mu_{lbn}(\frac{24\lambda_{ss}^4}{\beta_1})}{(\lambda_{lbn} + \mu_{lbn})} \tag{7.3}$$

Table 7.10: Ranking of Sensitivities for SSA and COA

δ	S_δ(SSA)
λ_{LB}	-2.39·10[1]
μ_{LB}	1.15·10[-2]
λ_{HW}	2.77·10[-17]
λ_{OS}	2.77·10[-17]
λ_{S}	2.77·10[-17]
λ_{CT}	2.77·10[-17]
λ_{APP}	2.77·10[-17]
μ_{HW}	0.0
μ_{S}	0.0
μ_{OS}	0.0
μ_{CT}	0.0
μ_{APP}	0.0

(a) $MTTF_{APP}$=4380

δ	S_δ(SSA)
λ_{LB}	-2.39·10[1]
μ_{LB}	1.15·10[-2]
λ_{HW}	0.0
λ_{OS}	0.0
λ_{S}	0.0
λ_{CT}	0.0
λ_{APP}	0.0
μ_{HW}	0.0
μ_{S}	0.0
μ_{OS}	0.0
μ_{CT}	0.0
μ_{APP}	0.0

(b) $MTTF_{APP}$=26280

δ	S_δ(SSA)
λ_{LB}	-2.39·10[1]
μ_{LB}	1.15·10[-2]
λ_{HW}	0.0
λ_{OS}	0.0
λ_{S}	0.0
λ_{CT}	0.0
λ_{APP}	0.0
μ_{HW}	0.0
μ_{S}	0.0
μ_{OS}	0.0
μ_{CT}	0.0
μ_{APP}	0.0

(c) $MTTF_{APP}$=87600

δ	S_δ(COA)
λ_{LB}	-2.39·10[1]
λ_{HW}	-4.24·10[-2]
λ_{OS}	-4.24·10[-2]
λ_{S}	-4.23·10[-2]
λ_{CT}	-4.24·10[-2]
λ_{APP}	-4.24·10[-2]
μ_{LB}	1.15·10[-2]
μ_{HW}	2.25·10[-6]
μ_{S}	2.25·10[-6]
μ_{OS}	2.25·10[-6]
μ_{CT}	2.25·10[-6]
μ_{APP}	2.25·10[-6]

(d) $MTTF_{APP}$=4380

δ	S_δ(COA)
λ_{LB}	-2.39·10[1]
λ_{HW}	-4.24·10[-2]
λ_{OS}	-4.24·10[-2]
λ_{S}	-4.23·10[-2]
λ_{CT}	-4.24·10[-2]
λ_{APP}	-4.24·10[-2]
μ_{LB}	1.15·10[-2]
μ_{HW}	1.63·10[-6]
μ_{S}	1.63·10[-6]
μ_{OS}	1.63·10[-6]
μ_{CT}	1.63·10[-6]
μ_{APP}	1.63·10[-6]

(e) $MTTF_{APP}$=26280

δ	S_δ(COA)
λ_{LB}	-2.39·10[1]
λ_{HW}	-4.24·10[-2]
λ_{OS}	-4.24·10[-2]
λ_{S}	-4.23·10[-2]
λ_{CT}	-4.24·10[-2]
λ_{APP}	-4.24·10[-2]
μ_{LB}	1.15·10[-2]
μ_{HW}	1.58·10[-6]
μ_{S}	1.58·10[-6]
μ_{OS}	1.58·10[-6]
μ_{CT}	1.58·10[-6]
μ_{APP}	1.58·10[-6]

(f) $MTTF_{APP}$=87600

$$COA = \frac{\mu_{lbn}\mu_{ss}\beta_2}{(\lambda_{lbn}+\mu_{lbn})\beta_1} \tag{7.4}$$

where:

$\beta_1 = 24\lambda_{ss}^4 + 24\lambda_{ss}^3\mu_{ss} + 12\lambda_{ss}^2\mu_{ss}^2 + 4\lambda_{ss}\mu_{ss}^3 + \mu_{ss}^4$,

$\beta_2 = 6\lambda_{ss}^3 + 6\lambda_{ss}^2\mu_{ss} + 3\lambda_{ss}\mu_{ss}^2 + \mu_{ss}^3$,

$\lambda_{ss} = \lambda_{HW} + \lambda_S + \lambda_{OS} + \lambda_{CT} + \lambda_{APP}$, and

$\mu_{ss} = \mu_{HW} + \mu_S + \mu_{OS} + \mu_{CT} + \mu_{APP}$.

The parametric sensitivity analysis was computed using equation (2.21) in the closed-form formulas (7.3) and (7.4). The full methodology can be observed in [80]. We selected the shorter, the intermediate, and the higher values of $MTTF_{APP}$, respectively 4380h, 26280h, and 87600h, to proceed with sensitivity analysis. The sensitivities coefficients $S_\delta(SSA)$ and $S_\delta(COA)$

with respect to system's failure and recovery parameters are presented in Table 7.10.

The sensitivity rankings of Table 7.10 are ordered from the most influential parameters to less ones. It reveal that the load balancer failure rate has the greatest influence when SSA and COA are the metrics of interest, one order of magnitude more influential than the second parameters, that are μ_{LB} for steady-state availability and all other failure rate parameters (λ_{HW} λ_{OS}, λ_S, λ_{CT}, and λ_{APP}) for COA.

Due to the replication of their components, the other sensitivity coefficients for *SSA* did not present critical importance, because they had 15 orders of magnitude less (failure rates when MTTF_{APP}=4380) than μ_{LB} or were null.

Regarding COA, μ_{LB} presented an influence level over system degradation of the same order of magnitude to λ_{HW} λ_{OS}, λ_S, λ_{CT}, and λ_{APP}, but with only nearly a quarter of impact. The remaining recovery rates were 4 orders of magnitude less impactful on the system degradation.

Yet according to Table 7.10, note that failure rates present a negative sensitivity coefficient because when they increase, SSA and COA decreases, and recovery rates have positive values because when they increase, SSA and COA also increases.

According to these sensitivity analysis results, a second load balancer was inserted in the next scenario aiming to improve SSA and COA.

7.3.2 No Load Balancer SPOF

Besides the recommendation for Single Point Of Failures elimination, the indication of sensitivity analysis results, presented in the previous scenario, also drives to the addition of a second load balancer.

The prevention of load balancer VNF SPOF results in availability improvement from three 9's of the previous scenario (Table 7.9) to six 9's, as can be noted in Table 7.12. It is an evidence that justifies the ETSI attention to prevent SPOF in their resiliency objectives [37] for NFV architectures.

Similarly to the previous scenario, the variability of application MTTF did not result in a significant impact over steady-state availability. System unavailability, according to second column of Table 7.12, was smaller than 15 seconds for all the $MTTF_{APP}$ variations. The resulting annual capacity degradation, represented by not delivered cache-minutes, can be observed in COUA column.

Regarding sensitivity analysis, an identical approach to previous scenario was adopted and the closed-form formulas are presented below:

$$SSA = \frac{\mu_{lbn}\mu_{ss}(2\lambda_{lbn} + \mu_{lbn})\beta_1}{\beta_3\beta_4} \tag{7.5}$$

$$COA = \frac{\mu_{lbn}\mu_{ss}(2\lambda_{lbn} + \mu_{lbn})\beta_2}{\beta_3\beta_4} \tag{7.6}$$

Table 7.11: Ranking of Sensitivities for SSA and COA: no LB SPOF

δ	S_δ(SSA)
λ_{LB}	$-4.60\cdot10^{-2}$
μ_{LB}	$2.21\cdot10^{-5}$
λ_{HW}	0.0
λ_{OS}	0.0
λ_{S}	0.0
λ_{CT}	0.0
λ_{APP}	0.0
μ_{HW}	0.0
μ_{S}	0.0
μ_{OS}	0.0
μ_{CT}	0.0
μ_{APP}	0.0

(a) $MTTF_{APP}$=4380

δ	S_δ(SSA)
λ_{LB}	$-4.60\cdot10^{-2}$
μ_{LB}	$2.21\cdot10^{-5}$
λ_{HW}	$2.00\cdot10^{-12}$
λ_{OS}	$2.00\cdot10^{-12}$
λ_{S}	$2.00\cdot10^{-12}$
λ_{CT}	$2.00\cdot10^{-12}$
λ_{APP}	$2.00\cdot10^{-12}$
μ_{HW}	0.0
μ_{S}	0.0
μ_{OS}	0.0
μ_{CT}	0.0
μ_{APP}	0.0

(b) $MTTF_{APP}$=26280

δ	S_δ(SSA)
λ_{LB}	$-4.60\cdot10^{-2}$
μ_{LB}	$2.21\cdot10^{-5}$
λ_{HW}	0.0
λ_{OS}	0.0
λ_{S}	0.0
λ_{CT}	0.0
λ_{APP}	0.0
μ_{HW}	0.0
μ_{S}	0.0
μ_{OS}	0.0
μ_{CT}	0.0
μ_{APP}	0.0

(c) $MTTF_{APP}$=87600

δ	S_δ(COA)
λ_{LB}	$-4.60\cdot10^{-2}$
λ_{HW}	$-4.24\cdot10^{-2}$
λ_{OS}	$-4.24\cdot10^{-2}$
λ_{S}	$-4.24\cdot10^{-2}$
λ_{CT}	$-4.24\cdot10^{-2}$
λ_{APP}	$-4.24\cdot10^{-2}$
μ_{LB}	$2.21\cdot10^{-5}$
μ_{HW}	$2.04\cdot10^{-6}$
μ_{S}	$2.04\cdot10^{-6}$
μ_{OS}	$2.04\cdot10^{-6}$
μ_{CT}	$2.04\cdot10^{-6}$
μ_{APP}	$2.04\cdot10^{-6}$

(d) $MTTF_{APP}$=4380

δ	S_δ(COA)
λ_{LB}	$-4.60\cdot10^{-2}$
λ_{HW}	$-4.24\cdot10^{-2}$
λ_{OS}	$-4.24\cdot10^{-2}$
λ_{S}	$-4.24\cdot10^{-2}$
λ_{CT}	$-4.24\cdot10^{-2}$
λ_{APP}	$-4.24\cdot10^{-2}$
μ_{LB}	$2.21\cdot10^{-5}$
μ_{HW}	$1.90\cdot10^{-6}$
μ_{S}	$1.90\cdot10^{-6}$
μ_{OS}	$1.90\cdot10^{-6}$
μ_{CT}	$1.90\cdot10^{-6}$
μ_{APP}	$1.90\cdot10^{-6}$

(e) $MTTF_{APP}$=26280

δ	S_δ(COA)
λ_{LB}	$-4.60\cdot10^{-2}$
λ_{HW}	$-4.24\cdot10^{-2}$
λ_{OS}	$-4.24\cdot10^{-2}$
λ_{S}	$-4.24\cdot10^{-2}$
λ_{CT}	$-4.24\cdot10^{-2}$
λ_{APP}	$-4.24\cdot10^{-2}$
μ_{LB}	$2.21\cdot10^{-5}$
μ_{HW}	$1.86\cdot10^{-6}$
μ_{S}	$1.86\cdot10^{-6}$
μ_{OS}	$1.86\cdot10^{-6}$
μ_{CT}	$1.86\cdot10^{-6}$
μ_{APP}	$1.86\cdot10^{-6}$

(f) $MTTF_{APP}$=87600

where:

$\beta_1 = 24\lambda_{ss}^3 + 12\lambda_{ss}^2\mu_{ss} + 4\lambda_{ss}\mu_{ss}^2 + \mu_{ss}^3$,

$\beta_2 = 6\lambda_{ss}^3 + 6\lambda_{ss}^2\mu_{ss} + 3\lambda_{ss}\mu_{ss}^2 + \mu_{ss}^3$,

$\beta_3 = 24\lambda_{ss}^4 + 24\lambda_{ss}^3\mu_{ss} + 12\lambda_{ss}^2\mu_{ss}^2 + 4\lambda_{ss}\mu_{ss}^3 + \mu_{ss}^4$,

$\beta_4 = 2\lambda_{lbn}^2 + 2\lambda_{lbn}\mu_{lbn} + \mu_{lbn}^2$,

$\lambda_{ss} = \lambda_{HW} + \lambda_S + \lambda_{OS} + \lambda_{CT} + \lambda_{APP}$, and

$\mu_{ss} = \mu_{HW} + \mu_S + \mu_{OS} + \mu_{CT} + \mu_{APP}$.

The sensitivity rankings of Table 7.11 show that the load balancer failure rate λ_{LB} is again the most influential parameter for SSA and COA. For A, λ_{LB} is 3 orders of magnitude more influential than μ_{LB}, the second most important parameter. However, both parameters had their influence level reduced in comparison to SPOF case: λ_{LB} was one order of magnitude smaller while μ_{LB} reduced 3 orders of magnitude. These values reveal an attenuation of the load

Table 7.12: Steady-State Availability, COA and COUA for CTCM without LB SPOF

$MTTF_{APP}$	SSA	COA	COUA (minutes)
4380	99.99995374%	99.93306726%	1407.19
8760	99.99995381%	99.93496226%	1367.35
13140	99.99995383%	99.93559851%	1353.97
26280	99.99995385%	99.93612895%	1342.82
43800	99.99995365%	99.93641699%	1336.77
87600	99.99995358%	99.93673706%	1330.88

balancers influence over SSA. The other failure rates were 7 orders of magnitude smaller than μ_{LB} for $MTTF_{APP}$=26280h and null for the other MTTFs. The remaining recovery rates were null. So, the load balancers are the critical component for Steady-State Availability. For COA, in comparison with the previous scenario: λ_{LB} reduced one order of magnitude, μ_{LB} reduced 3 orders of magnitude, and the other parameters presented similar orders of magnitude.

Figs. 7.4 and 7.5 plot a comparison of failure and recovery rates, respectively, for the scenario without SPOF. They demonstrate that maintenance services should be focused on failures mitigation, mainly in load balancers, whereas the attention of recovery team should be directed to the load balancer recovery, aiming to mitigate capacity degradation issues.

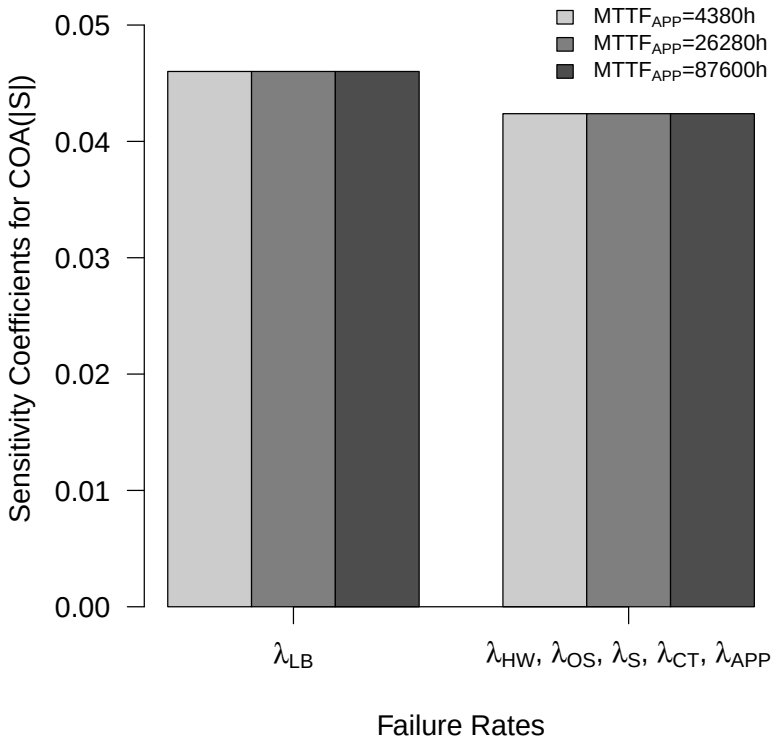


Figure 7.4: Load balancer failure rate is the most influential parameter for COA, but with similar magnitude order

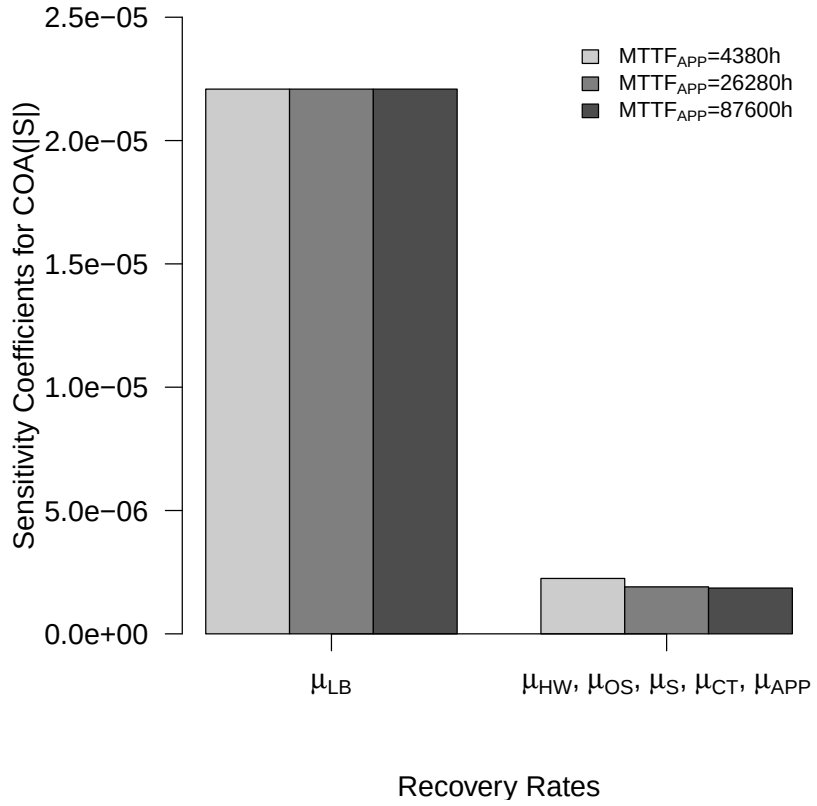


Figure 7.5: Load balancer recovery rate is the most influential parameter for COA, but one order of magnitude higher than other parameters

7.4 Comparison with Reference Work

This case study aims at analyzing how reasonable are our proposed modeling approach and results to estimate the availability of VNF chains through a comparison with previous literature. We performed a comparison with the Di Mauro et al. [33] work, because they also perform an availability study of VNFs chains. The authors address an availability evaluation of a chain of network nodes implementing an SFC managed by openstack Virtual Infrastructure Manager (VIM). A double-layer model was also adopted, where an RBD describes the top-level dependencies among the architecture components and Stochastic Reward Networks (SRN) model the probabilistic behavior of each component. Firstly, we present one instance of our proposed model for 3N redundant VNF chain. Next, we present Di Mauro et al. model. Finally, we compare the results regarding steady-state availability.

7.4.1 3N Redundant Baseline Model

Figure 7.6 depicts simultaneously: (i) a 3N redundant model, representing three openstack VIM nodes, based on the SPN sub-model of Figure 6.8 and (ii) a 3N redundant VNF model, based on sub-models Node (Figure 6.8), Chain(Figure 6.10b), and Chain Interconnection (Figure 6.12). Each openstack VIM node represents the joint operation of the controller and neutron deployment

modes. The VNF sub-system represents three redundant instances of computer deployment mode providing their chains and chains interconnections.

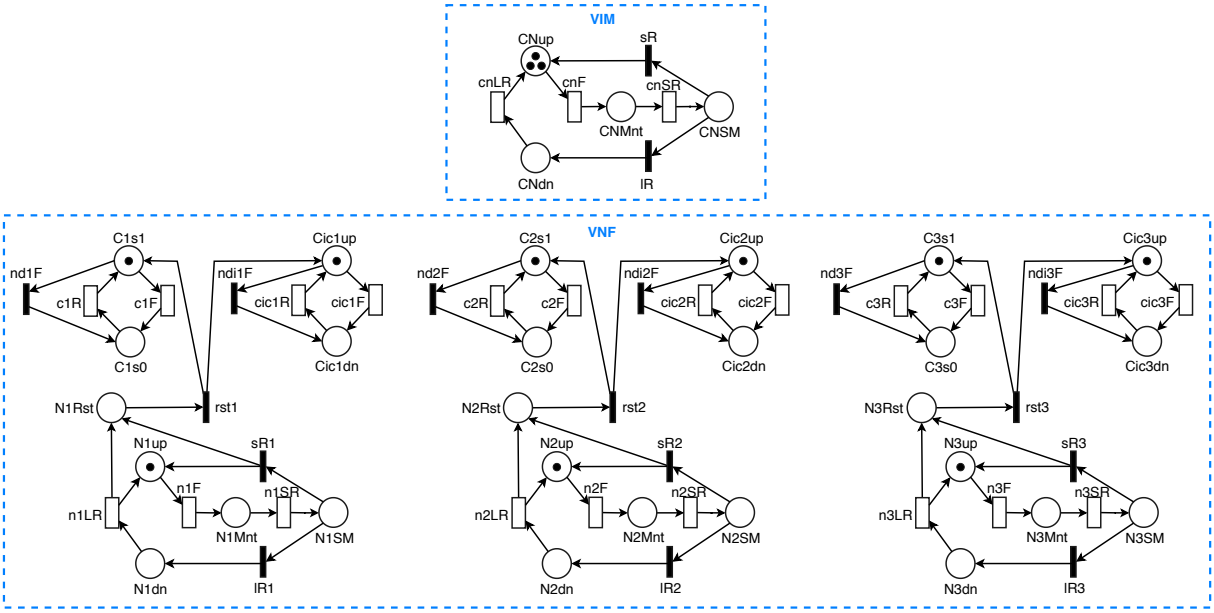


Figure 7.6: 3N redundant baseline model

The sub-models representing computer nodes interact with chains and chain interconnection sub-models through the N_iRst place and the rst_i transition $(1 <= i <= 3)$. One token is deposited at place N_iRst whenever a Node is recovered, firing rst_i. We consider that all software (Figure 6.6c) is working when its underlying Node is restarted: the token is taken from place N_iRst and deposited at working places C_iS1 and Cic_iup.

Using hierarchical modeling, each token in place $CNup$ of top-level SPN represents a Controller/Neutron joint deployment mode node, as depicted in the low-level RBD of Figure 7.7.

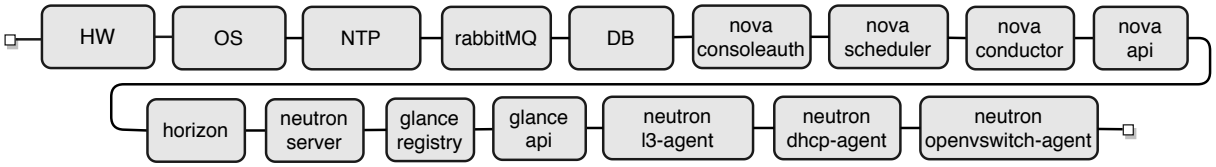


Figure 7.7: RBD for joint Controller and Neutron deployment modes Node

Similarly, each token in places N_iup of top-level SPN represents a Compute node, as depicted in the low-level RBD of Figure 7.8.

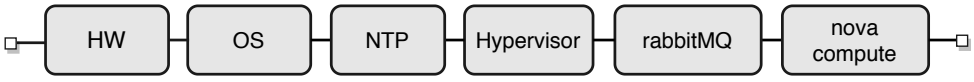


Figure 7.8: RBD for Compute deployment mode node

Finally, each token in places C_is1 of top-level SPN represents a Service Function Chain, as depicted in the low-level RBD of Figure 7.9.

Figure 7.9: RBD for Compute deployment mode node

The system is considered active when at least one VIM sub-system is active and one VNF sub-system is also active. The expression

$$P\{(((\#C1s1 = 1)AND(\#Cic1up = 1))OR((\#C2s1 = 1)AND(\#Cic2up = 1))OR((\#C3s1 = 1) \\ AND(\#Cic3up = 1)))AND(\#CNup > 0)\} \tag{7.7}$$

was adopted to estimate the Steady-State Availability of our 3N redundant baseline model (Figure 7.6).

7.4.2 Reference Work for Comparison

Figure 7.10 depicts the top-level RBD model representing the dependencies among the VIM and VNF sub-systems. The VIM sub-system is composed of N redundant VIM nodes. The comparison with our previously proposed model requires $N = 3$.

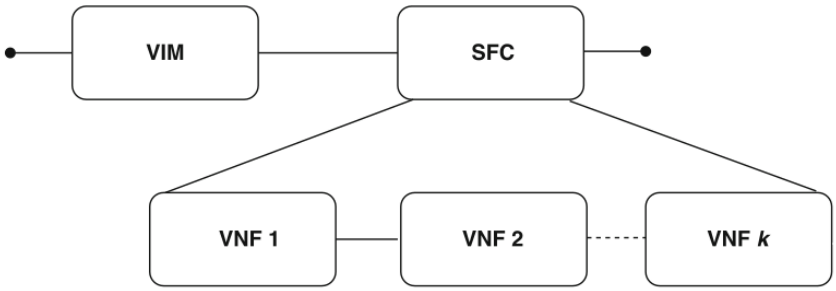


Figure 7.10: Top-level model of reference work

The Figure 7.11 depicts the proposed model for openstack VIM, whereas the Figure 7.12 depicts the authors' model for VNFs. We describe the model behavior of the reference work below.

Places P_{upDB}, P_{upFB}, P_{upHA}, P_{upVMM}, and P_{upHW} indicate the conditions where the database, the functional blocks, the HAproxy, the hypervisor, and the hardware of the VIM are working. Places with index f indicate the conditions where such components are down. Figure 7.11 shows the fully working condition of VIM sub-system. As an exemplary case, consider the failure of the database. When DB fails, T_{fDB} fires and the token removed from P_{upDB} is deposited into P_{fDB} . The immediate transition t_{DB} fires when the transition T_{fVMM} is fired, meaning that a hypervisor failure implies virtual modules failure as well. Similarly, the immediate transition t_{VMM} accounts for a hypervisor failure as a consequence of a hardware failure. The inhibitory arc between P_{upHW} and t_{VMM} compels the hypervisor failure in case of

hardware failure. The inhibitory arc between P_{fHW} and t_{rVMM} forbids the hypervisor repair in case of hardware failure.

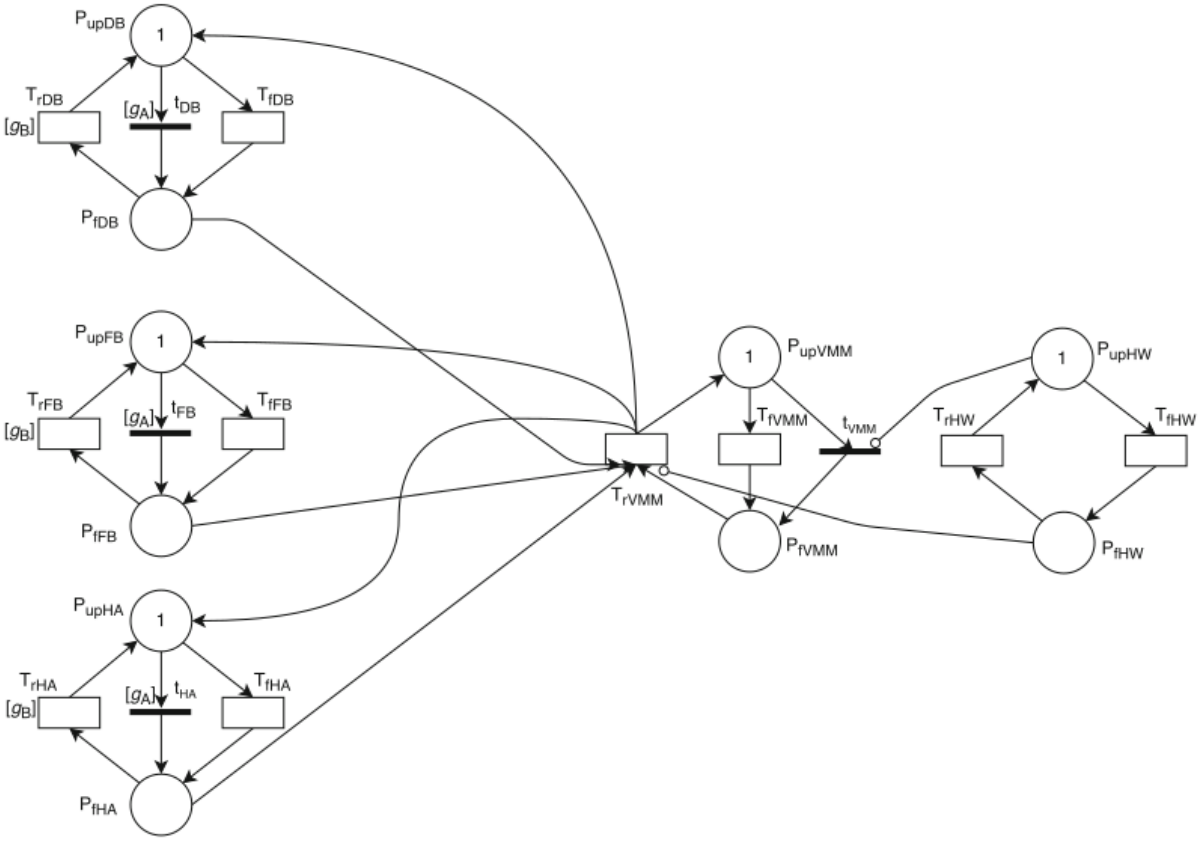


Figure 7.11: VIM of reference work

Furthermore, the first guard expressions of Table 7.13 enables t_{DB}, t_{FB}, t_{HA} when hypervisor fails, namely, when a token is moved from P_{upVMM} to P_{fVMM}, whereas $\#P_{upVMM} = 1$ inhibits the repair of the virtual modules in case of hypervisor failure.

Table 7.13: Guard expressions for the VIM of the reference work model

Transition	Guard Expressions
t_{DB}, t_{FB}, t_{HA}	$\#P_{fVMM} = 1$
t_{rDB}, t_{rFB}, t_{rHA}	$\#P_{upVMM} = 1$

Figure 7.12 depicts the VNF sub-system. The place P_{upVNF} indicates the VNF working state, implying that the hardware, software and, virtual resources are correctly working. The token value inside P_{upVNF} amounts to $L+M$, namely, the number of initial working VNFs replicas. Originally, the authors of reference work consider the assumption that the number of deployed VNFs can vary dynamically with time. So, an elasticity functionality was developed through the sub-net composed by the place Pp and transitions t_{So}, t_{Si}, T_p, and through the sub-net composed by the place P_L, and transitions T_rem, T_{add}. The Scale-Out (t_{So}) and Scale-In(t_{Si}) operations corresponding to a provisioning phase (deploying replicas) and a de-provisioning

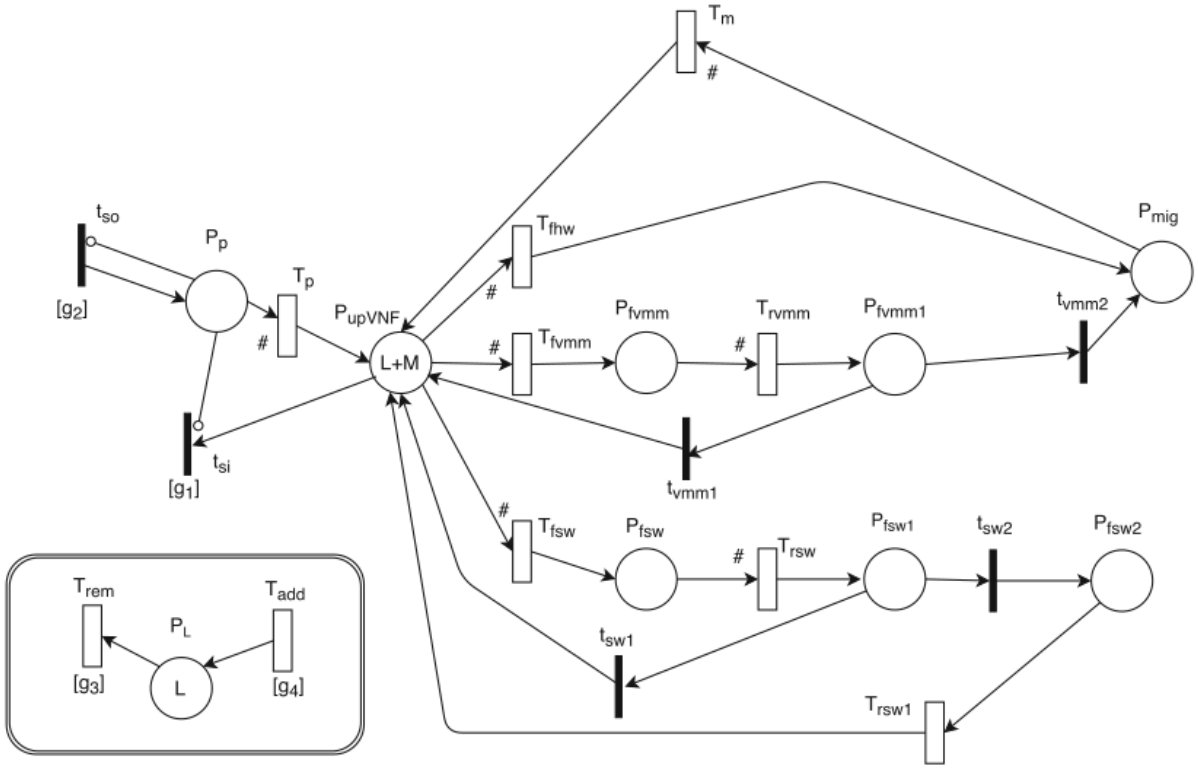


Figure 7.12: VNF of reference work

phase (un-deploying replicas), respectively. Thus, when t_{So} fires, a token is deposited in place P_p, modeling the condition of a replica requested but not working yet, until the token enters P_{upVNF} after T_p firing. The inhibitory arc from P_p to t_{So} models the impairment of multiple provisioning stages. On the contrary, a Scale-In operation happens when t_{Si} fires. The inhibitory arc from P_p to t_{So} preventing Scale-In operations during provisioning stages. A complete explanation can be found in the authors' work. The reference work models were implemented on SHARPE (Symbolic Hierarchical Automated Reliability and Performance Evaluator) tool [118].

The Steady-State Availability of reference work is estimated by

$$A_{NS} = A_{VIM} \prod_{m=1}^{3} A_{VNF}^{(m)} \tag{7.8}$$

which consider a 3N redundant system.

Aiming at enabling a comparison between the reference work and our models, the adopted approach was based on 4 steps:

i. Reproduce the reference work model and its results on our adopted Mercury tool;
ii. Remove the elasticity sub-nets of the model;
iii. Execute the analysis of the steady-state availability metric adopting our input values in the reference work;
iv. compare the results obtained in step iii with the analysis of our model.

Step i: Figure 7.13 represents the 3N redundant VIM model of reference work im-

plemented by us in Mercury tool. Regarding VNF, we also reproduce the model exhibited in Figure 7.12 in Mercury tool.

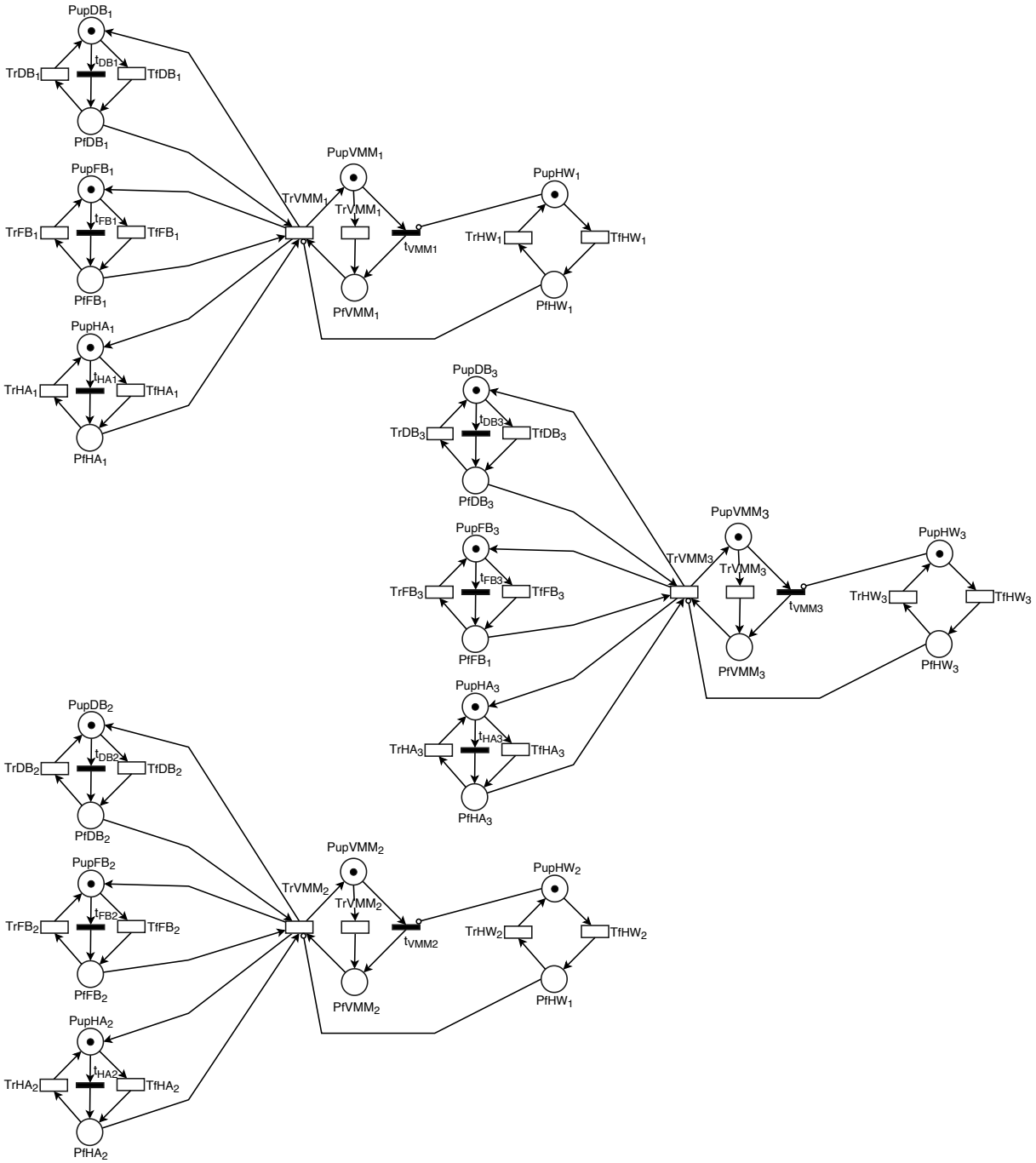


Figure 7.13: VIM of reference work

The VIM sub-system is in working state when at least two instances of database are active and at least one of instance of functional blocks and HAproxy is active. This state is represented by the equation:

$$A_{VIM} = P\{(((\#PupDB1 = 1)AND(\#PupDB2 = 1))OR((\#PupDB1 = 1)AND(\#PupDB3 = 1))$$
$$OR((\#PupDB2 = 1)AND(\#PupDB3 = 1)))$$
$$AND((\#PupFB1 = 1)OR(\#PupFB2 = 1)OR(\#PupFB3 = 1))$$
$$AND((\#PupHA1 = 1)OR(\#PupHA2 = 1)OR(\#PupHA3 = 1))\} \quad (7.9)$$

The VNF sub-system is in working state when the number total replicas is no less than the number of regular replicas. This state is represented by the equation:

$$A_{VNF} = P\{\#P_{upVNF} > \#P_L\} \quad (7.10)$$

The estimation for Steady-State Availability from the reference work, using Equation 7.8 at SHARPE tool, is equal to $SSA = 99.998444\%$, whereas our result, using Mercury tool, was equal to 99.998618%. We consider these results close enough to accept them as equivalents, associating the difference in the order of 10^{-6} due to the adoption of distinct tools.

Step ii: Figure 7.14 depicts the VNF model of reference work without elasticity sub-net.

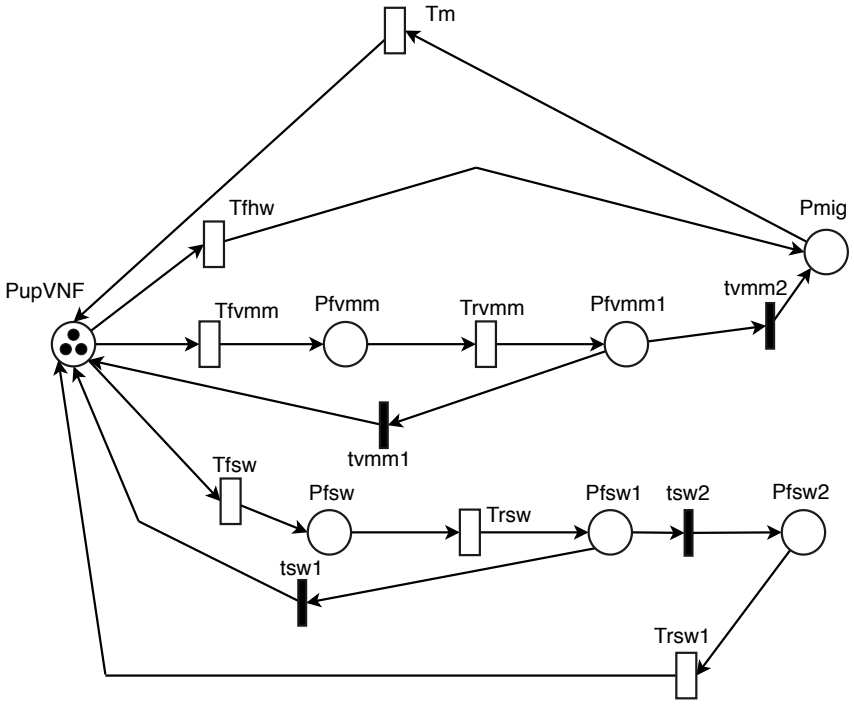


Figure 7.14: VNF without elasticity

As our goal is to perform a fair comparison with our model, we now estimate the Steady-State Availability adopting an identical approach used in our model. The VNF sub-system now is considered in working state when the number of VNFs are not null. This state is represented by the expression:

$$P\{(\#PupVNF > 0)\} \tag{7.11}$$

Step iii: Aiming at perform the comparison between our model and the reference work model, we consider the working state of VIM sub-system without elasticity as a not null number of required services. So, the SSA for the model of Figure 7.14 is estimated by:

$$\begin{aligned} &P\{((\#PupDB1 = 1)OR(\#PupDB2 = 1)OR(\#PupDB3 = 1)) \\ &AND((\#PupFB1 = 1)OR(\#PupFB2 = 1)OR(\#PupFB3 = 1)) \\ &AND((\#PupHA1 = 1)OR(\#PupHA2 = 1)OR(\#PupHA3 = 1))\} \end{aligned} \tag{7.12}$$

Furthermore, the inputted parameters for failure and repair rates for functional blocks and hardware were originated from RBDs numerical analysis, as depicted in Figure 7.15. Using Mercury tool, we can compute the MTTF and MTTR of all RBD. The failure and repair rates λ_{FB} and μ_{FB} were inserted in transitions $TfFB_i$ and $TrFB_i$, respectively, whereas λ_{CN} and μ_{CN} were inserted in transitions $TfHW_i$ and $TrHW_i$, respectively.

Similarly, for VNF SPN, the failure rate for the hardware and for the chain, as well as the chain repair rate, were also originated from their correspondent RBDs, as depicted in Figure 7.16.

Table 7.14: Input mean times for cloud components

Component	**MTTF**	**MTTR**
HW	60000h	8h
Hypervisor	5000h	10min
OS	2893h	0.25h
DB, NTP, rabbitMQ, nova-consoleauth, nova-scheduler, nova-conductor, nova-api, horizon, neutron-server, glance-registry, glance-api, neutron-l3-agent, neutron-dhcp-agent nova-compute	3000h	1h
VMFW, VMLB, VMCache	2160h	0.25h
FW, LB, Cache	2160h	0.25h

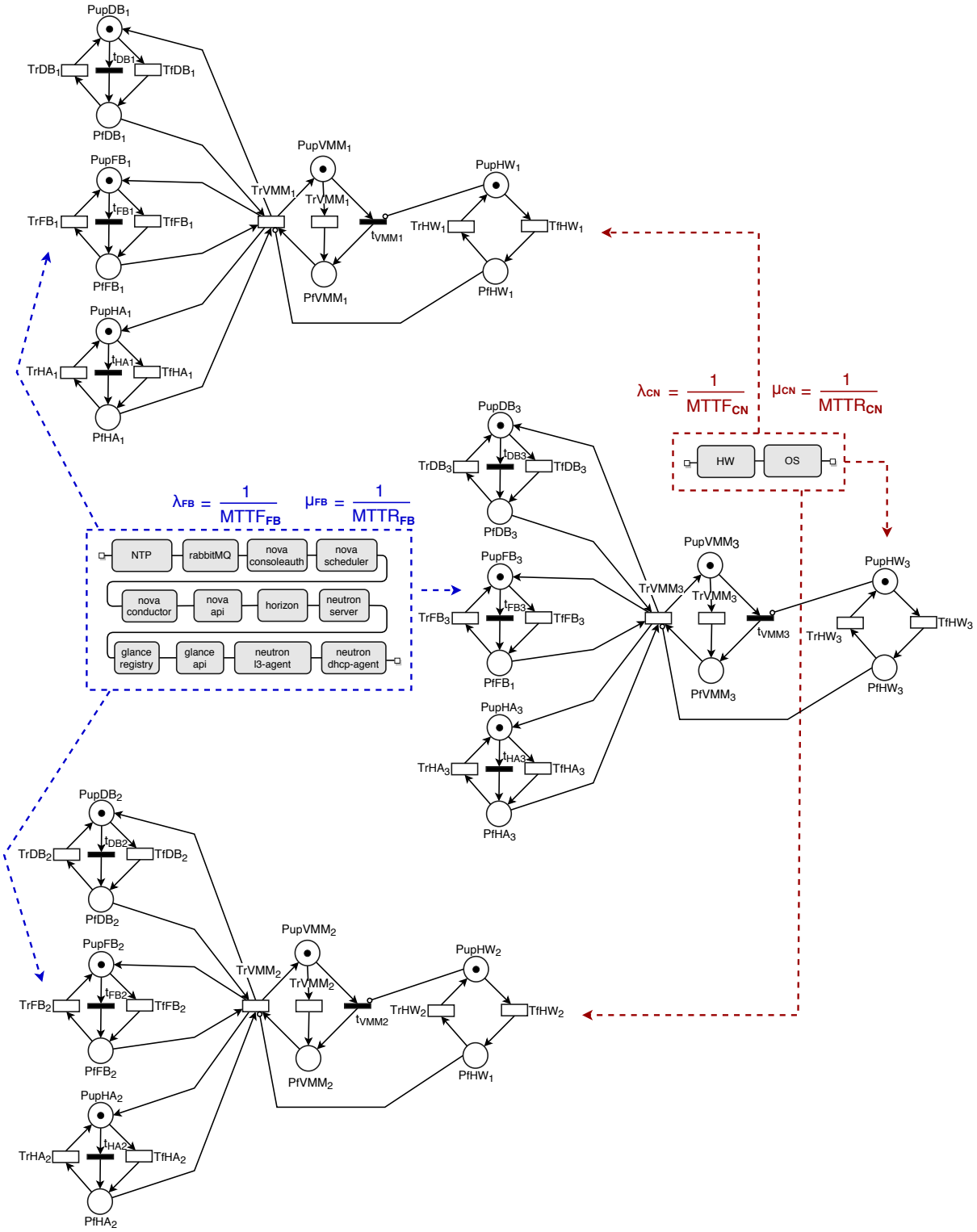


Figure 7.15: Customized VIM of reference work

Table 7.14 exhibits the parameters' values [64] used as input in the RBDs of Figures 7.15 and 7.16. We consider a three months for MTTF and 15 minutes for MTTR to VMs and its hosted services.

Step iv: Finally, aiming at comparing the results, we performed an identical approach regarding our model, inserting resulting failure and repair rates from RBDs numerical analysis.

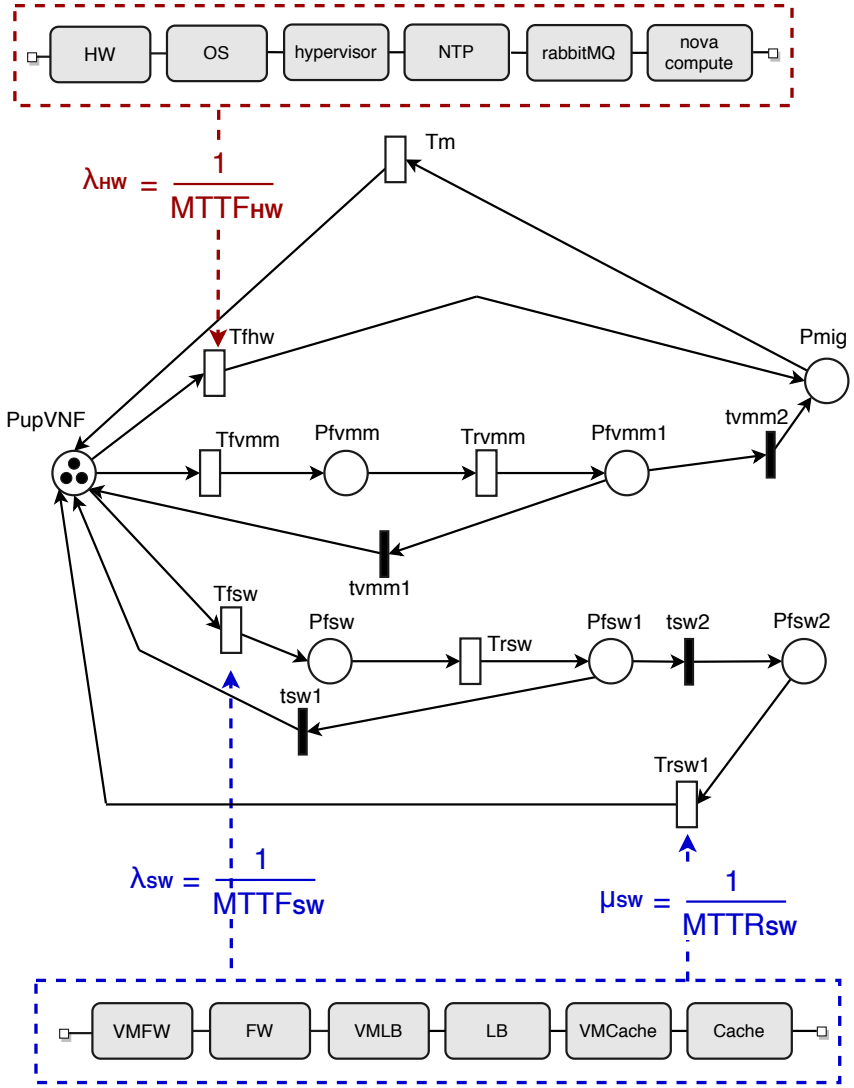


Figure 7.16: Customized VNF without elasticity

The failure and repair rates λ_c and μ_c were inserted in transitions c_iF and c_iR, respectively. Also, the failure and repair rates λ_{CN} and μ_{CN} were inserted in transitions cnF and $cnLR$, respectively. Finally, the failure and repair rates λ_N and μ_N were inserted in transitions n_iF and n_iLR.

Table 7.14 exhibits the parameters' values [64] used as input in the RBDs of Figure 7.17. The input parameters of the RBDs depicted in Figure 7.17 were also in Table 7.14.

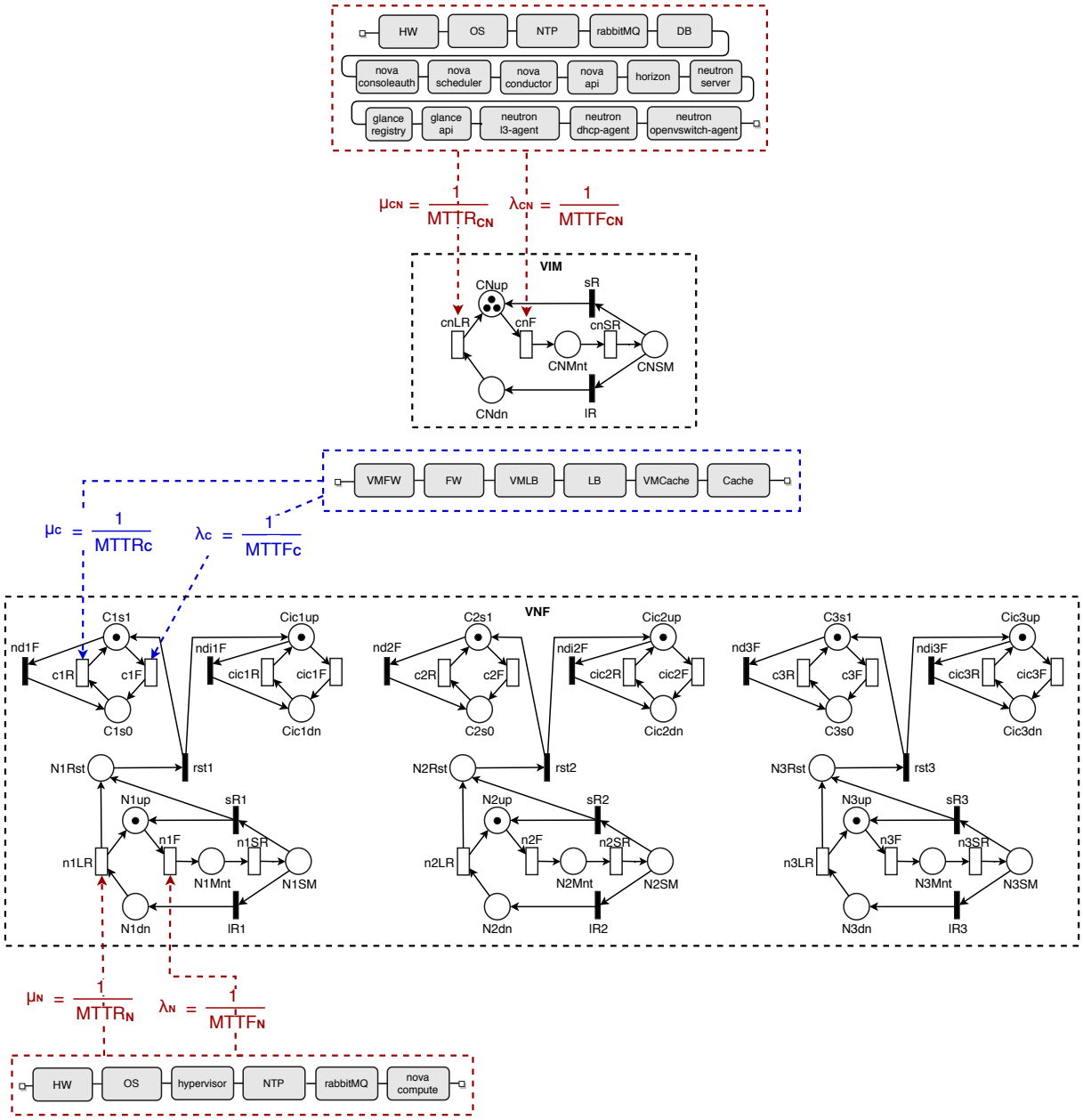


Figure 7.17: 3N redundant baseline model: the insertion of input parameters in chain and compute node were represented only in the first instances of there sub-models to do not overload the figure

Table 7.15 shows the analysis results for both models. As one can observe, the difference between results is in the order of 10^{-7}. Such a distinction results in a daily downtime difference of 1.15s between the models' SSA estimations.

Table 7.15: Steady-State Availability Comparison

Model	SSA(%)
Reference Model	99.99997229308%
Proposed Model	99.99991759069%

The evidence of such close results does not allow to claim that the models are different

enough to consider them as heterogeneous. So, we consider our results effectual.

7.5 Rejuvenation of Service Function Chains

The studied service chain is composed of firewalling, load balancing, and video caching services. Figure 7.18 depicts the RBD of the adopted service chain. It has a series design because a failure in any component means a service chain downtime.

Figure 7.18: Service Chain's RBD

The resulting MTTF of Figure 7.18 RBD, adopting Eq. (6.2), is computed as:

$$MTTF_{SFC} = \frac{1}{\lambda_{vmfw} + \lambda_{fw} + \lambda_{vmlb} + \lambda_{lb} + \lambda_{vmcache} + \lambda_{cache}} \tag{7.13}$$

Table 7.14 exhibits the parameters' values [64] used as input in the RBDs of Figs. 6.6 and 7.18. We consider a three month for MTTF and 15 minutes for MTTR to VMs and its hosted services.

Table 7.16: Input mean times for cloud components

Component	MTTF	MTTR
HW	60000h	8h
Hypervisor	5000h	10min
OS	2893h	0.25h
DB, NTP, rabbitMQ, nova-consoleauth, nova-scheduler, nova-conductor, nova-api, horizon, neutron-server, glance-registry, glance-api, neutron-l3-agent, neutron-dhcp-agent nova-compute	3000h	2h
VMFW, VMLB, VMCache	2160h	0.25h
FW, LB, Cache	2160h	0.25h

Table 7.17 exhibits the resulting MTTF of all components present in the scenarios described in Table 7.23. These values are used as input of the high-level SPN models and are referred by their IDs.

7.6 3N Redundant Service Function Chain

This case study analyzes an All-In-One openstack deployment. We consider a 3N redundant SFC composed of 3 instances of the chain depicted in Figure 7.18. The first scenario (Sec-

Table 7.17: Results from low-level RBDs analysis

RBD	$MTTF_{ID}$	MTTF(h)
Controller	cMTTF	248.20083
Neutron	nMTTF	589.74019
Compute	ndMTTF	640.07020
Joint Controller and Neutron	cnMTTF	198.84687
All-In-One	allMTTF	179.78082
Service Chain	chainMTTF	500.00000

tion 7.6.1) is the baseline one, without rejuvenation, whereas the second scenario (Section 7.6.2) presents the 3N redundant model with rejuvenation based on VM live migration.

7.6.1 Baseline model

Figure 7.19 depicts the 3N redundant model, representing three All-In-One openstack nodes.

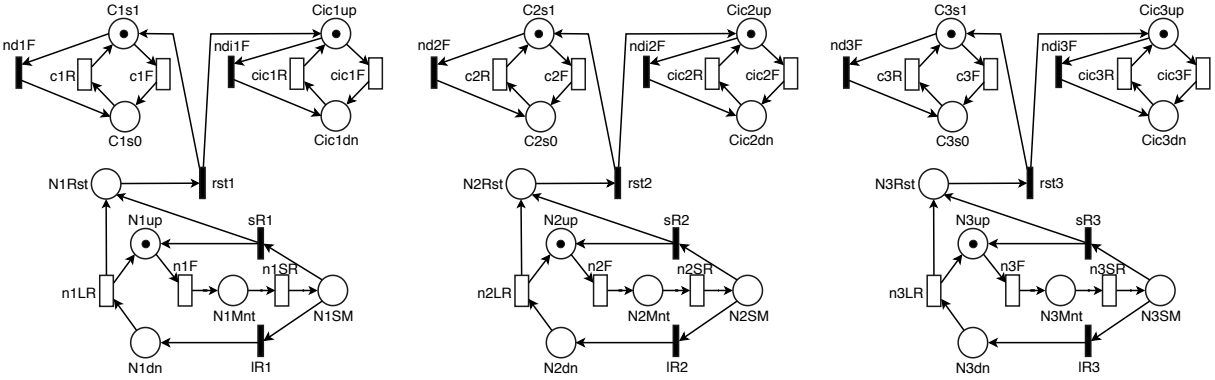


Figure 7.19: 3N redundant baseline model

The sub-models Node, Chain, and Chain Interconnection interact through N_iRst place and rst_i transition $(1 <= i <= 3)$. One token is deposited at place N_iRst whenever a Node is recovered, firing rst_i. We consider that all software (Figure 6.6e) is working when its underlying Node is restarted. The restarting is represented when the token is taken from place N_iRst and deposited at working places C_iS1 and Cic_iup.

Table 7.18: Guard expressions and mean times for transitions in the 3N baseline model

Transition	Guard Expressions	Mean Time
c1F	-	500 h
cic1F	-	3000 h
c1R, cic1R	#N1up>0	2 h
nd1F, ndi1F	#N1Mnt>0	-
n1F	-	179.780 h
n1SR	-	7 min
n1LR	-	8 h
rst1, sR1, lR1	-	-

7.6.2 Rejuvenation model

The model exhibited in Figure 7.20 adopts VM live migration as rejuvenation technique to improve the system steady-state availability. Figure 7.20 depicts the 3N redundant model in which preventive maintenance is performed in the three modeled openstack All-In-One nodes.

If there is a Node failure during a live migration, the model put the VMs back to the original place. For example, if Node 01 fails during a migration, $m12F$ fires and the token in $m12$ is taken and deposited at place $C1s0$. An identical behavior occurs when $m13F$, $m21F$, $m23F$, $m31F$, or $m32F$ fire.

Several requirements must be accomplished to allow preventive maintenance. They were mapped to the model through guard expressions. All the procedures are similar when considering preventive maintenance from any openstack node to another. We presented them only between Nodes 01 and 02 for brevity.

Table 7.19 presents the guard expressions for transitions involved in the preventive maintenance from Node 01 to Node 02. The transition ch1m2 fires with an MTBPM, representing the initiation of a preventive maintenance, if:

i. the destination Node 02 is working (#N2up>0);
ii. there is no other migration between these Nodes in process ((#m21=0)AND(#m12=0));
iii. the destination chain interconnection is working (#Cic2up=1);
iv. there are no failure chains is the destination Node (#C2s0=0);
v. there is only one chain in Node 01 (#C1s1<2);
vi. there are no failure chains in the source Node (#C1s0=0).

Such requirements aim to minimize the downtime, allowing the initiation of maintenance with as many operational resources as possible. From place m12, the model verifies again if destination Node is active (#N2up>0) and if the openstack SFC API mandatory requirement of delete Open vSwitch rules before migration is valid (#Cicup=0).

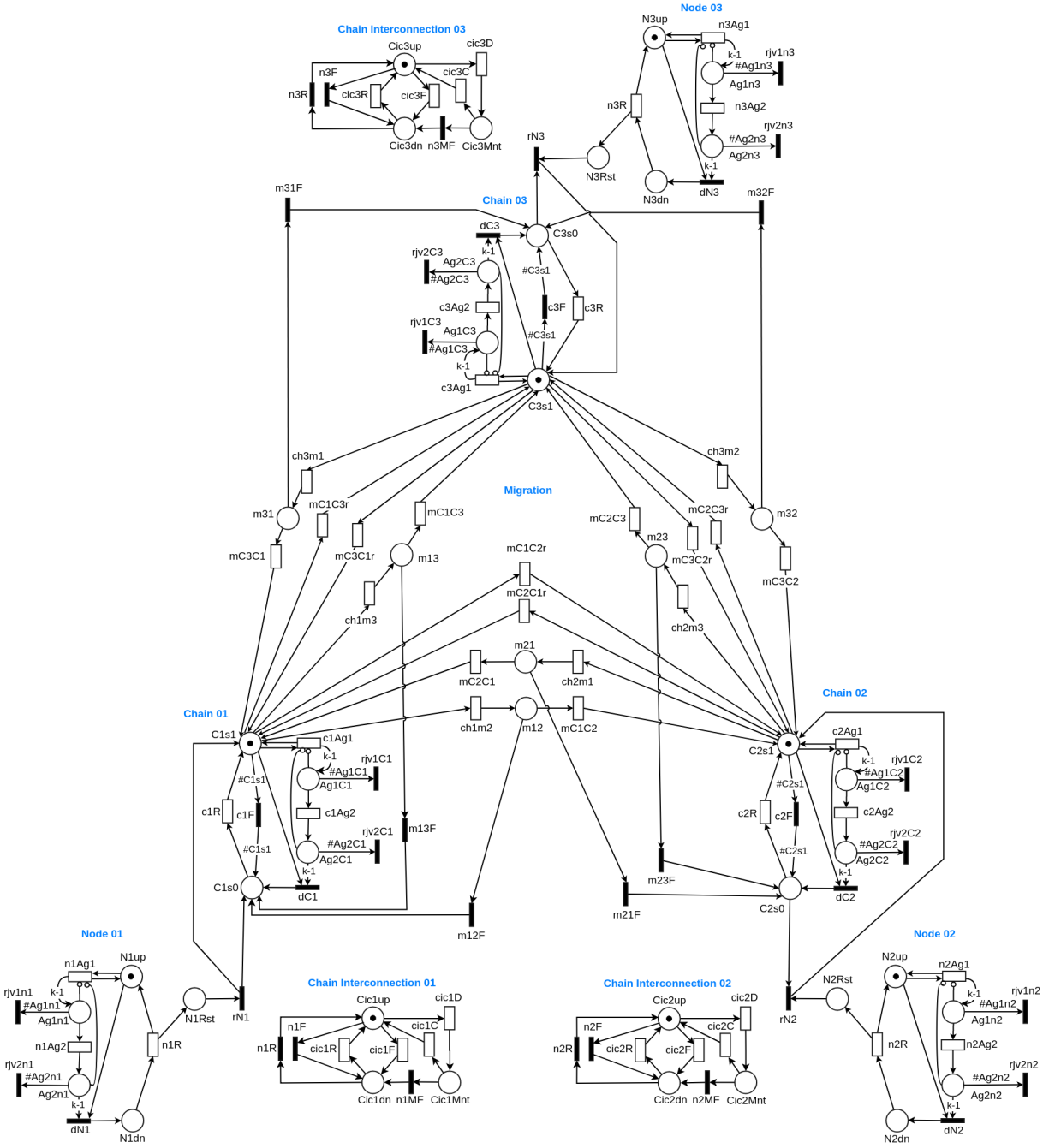


Figure 7.20: 3N redundant model with rejuvenation based on VM live migration

After the preventive maintenance execution, the chain return will occur if

i. the Open vSwitch rules were not configured (#Cic1up=0);
ii. the Node 01 is active (#N1up>0);
iii. there is no chains in Node 01 (#C1s1+#C1s0=0).

An identical behavior is true for the firing of transition triads: ch2m1, mC2C1, mC1C2r; ch1m3, mC1C3, mC3C1r; ch3m1, mC3C1, mC1C3r; ch2m3, mC2C3, mC3C2r; and ch3m2, mC3C2, mC2C3r.

Table 7.20 shows the adopted guard expressions for Node. The guard expressions

Table 7.19: Guard expressions in 3N rejuvenation model: migration sub-model

Transition	Guard expressions	Mean time(h)
ch1m2	(#N2up>0)AND(#m21=0)AND(#m12=0)AND(#Cic2up=1) AND(#C2s0=0)AND(#C1s1<2)AND(#C1s0=0)	MTBPM
mC1C2	(#N2up>0)AND(#Cic1up=0)	MTTCLM+ MTTPPM
mC2C1r	(#Cic1up=0)AND(#N1up>0)AND (#C2s1>1)AND(#C1s1+#C1s0=0)	MTTCLM

(#Ag1N1>0) and (#Ag2N1>0) prevents the firing of rjv1n1 and rjv1n2, respectively, without tokens in place Ag1n1 and Ag2n1. The model detects a live migration from Node 01 by:

i. the sum of two tokens at places C2s1 and C2s0 and the sum of one token at places C3s1 and C3s0 ((#C2s1+#C2s0=2)AND(#C3s1+#C3s0=1));
ii. the sum of one token at places C2s1 and C2s0 and the sum of two tokens at places C3s1 and C3s0 ((#C2s1+#C2s0=1)AND(#C3s1+#C3s0=2));
iii. the sum of three tokens at places C2s1 and C2s0 or at places C3s1 and C3s0 ((#C2s1+#C2s0=3)) OR((#C3s1+#C3s0=3)).

In any of these conditions, as the live migration occurred, the tokens at places Ag1n1 and Ag2n1 are removed, realizing the rejuvenation. An identical behavior is true for the rejuvenation of Nodes 02 and 03, with their respective guard expressions.

Table 7.20: Guard expressions in 3N rejuvenation model: node sub-model

Transition	Guard expressions	Mean time(h)
c1Ag1, c1Ag2	-	179.780/k
rjv1n1	(#Ag1N1>0)AND (((#C2s1+#C2s0=2)AND(#C3s1+#C3s0=1)) OR((#C2s1+#C2s0=1)AND(#C3s1+#C3s0=2)) OR((#C2s1+#C2s0=3))OR((#C3s1+#C3s0=3)))	-
rjv2n1	(#Ag2N1>0)AND (((#C2s1+#C2s0=2)AND(#C3s1+#C3s0=1)) OR((#C2s1+#C2s0=1)AND(#C3s1+#C3s0=2)) OR((#C2s1+#C2s0=3))OR((#C3s1+#C3s0=3)))	-
n1R	-	8
dN1	-	-

Table 7.21 shows the guard expressions for the Chain sub-model.

The requirements for chain rejuvenation are :

i. the presence of tokens in places m12 (#m12>0) or m13 (#m13>0);
ii. the presence of tokens at places C1s1 or C1s0 (#C1s1+#C1s0>0) while Node 01 is inactive (#N1dn>0).

Table 7.21: Guard expressions in 3N rejuvenation model: chain sub-model

Transition	Guard expressions	Mean time(h)
c1Ag1, c2Ag1	-	500/k
rjv1c1	(#Ag1C1>0)AND((#m12>0)OR (#m13>0))OR((#C1s1+#C1s0>0)AND(#N1dn>0))	-
rjv2c1	(#Ag2C1>0)AND((#m12>0)OR (#m13>0))OR((#C1s1+#C1s0>0)AND(#N1dn>0))	-
dC1	(#Ag2C1=(k-1))	-
c1F	(#N1dn=1)AND(#Cic1up=0)AND(#C1s1>0)	-
c1R	(#N1up>0)	2

because when a Node returns of a downtime period its software will be rejuvenated. The conditions for rejuvenation are identical for Chain 02 and Chain 03.

Table 7.22 shows the guard expressions for the Chain interconnection sub-model. The transition $cic1D$ will be enabled after the initiation of a migration process $((\#m12 = 1)OR(\#m13 = 1))$ or after its conclusion, enabling the return of a chain to its original Node. The guard expressions for transitions cic1C, cic1R, and, n1R state the conditions in which a down chain can return to working state, whereas the guard expressions for transitions n1F and n1MF state the conditions in which an up chain must be conducted to a down state. This behavior is identical for the Chain Interconnections 02 and 03.

Table 7.22: Guard expressions and mean times in 3N rejuvenation model: chain interconnection sub-model

Transition	Guard expressions	Mean time(h)
cic1D	(#m12=1)OR(#m13=1)OR((#C2s1=2)AND (#C3s1+C3s0=1))OR((#C3s1=2) AND(#C2s1+C2s0=1)) OR(#C2s1+#C2s0=3)OR(#C3s1+#C3s0=3)	0.0017075
cic1C, cic1R	((#C1s1+#C1s0>0)AND(#N1up>0)) OR((#C2s1+#C2s0=2)AND(#C3s1+#C3s0=1)AND(#N2up>0)) OR((#C3s1+#C3s0=2)AND(#C2s1+#C2s0=1)AND(#N3up>0)) OR((#C2s1+#C2s0=3)AND(#N2up>0)) OR((#C3s1+#C3s0=3)AND(#N3up>0))	0.0033580, 2
cic1F	-	3000
n1F, n1MF	((#C1s1+#C1s0>0)AND(#N1dn>0)) OR((#C2s1+#C2s0=2)AND(#C3s1+#C3s0=1)AND(#N2dn>0)) OR((#C3s1+#C3s0=2)AND(#C2s1+#C2s0=1)AND(#N3dn>0)) OR((#C2s1+#C2s0=3)AND(#N2dn>0)) OR((#C3s1+#C3s0=3)AND(#N3dn>0))	-
n1R	((#N1Rst=1)AND(#C1s0>0)) OR((#N2Rst=1)AND(#C2s1+#C2s0=2)AND(#C3s1+#C3s0=1)) OR((#N3Rst=1)AND(#C3s1+#C3s0=2)AND(#C2s1+#C2s0=1)) OR((#N2Rst=1)AND(#C2s1+#C2s0=3)) OR((#N3Rst=1)AND(#C3s1+#C3s0=3))	-

Similar to the baseline model, the system is active if at least one chain and its corre-

sponding chain interconnection are active. But for the rejuvenation model, there are additional conditions of working states: when there are two or three chains in an openstack Node due to VM live migrations. So, we adopted Equation 7.14 to compute SSA for the 3N rejuvenation model. It encompasses all the additional conditions in which the system is considered up.

$$
\begin{aligned}
SSA = P\{(((\#C1s1 = 3) OR (\#C2s1 = 3) OR (\#C3s1 = 3)) \\
AND \\
((\#Cic1up = 1) OR (\#Cic2up = 1) OR (\#Cic3up = 1))) \\
OR \\
(((\#C1s1 = 2) AND ((\#Cic1up = 1) OR (\#Cic2up = 1)) OR ((\#Cic1up = 1) OR (\#Cic3up = 1)) OR \\
((\#Cic2up = 1) OR (\#Cic3up = 1)))) \\
OR \\
(((\#C2s1 = 2) AND ((\#Cic1up = 1) OR (\#Cic2up = 1)) OR ((\#Cic1up = 1) OR (\#Cic3up = 1)) OR \\
((\#Cic2up = 1) OR (\#Cic3up = 1)))) \\
OR \\
(((\#C3s1 = 2) AND ((\#Cic1up = 1) OR (\#Cic2up = 1)) OR ((\#Cic1up = 1) OR (\#Cic3up = 1)) OR \\
((\#Cic2up = 1) OR (\#Cic3up = 1)))) \\
OR \\
((\#C1s1 = 1) AND (\#Cic1up = 1)) OR ((\#C2s1 = 1) AND (\#Cic2up = 1)) \\
OR ((\#C3s1 = 1) AND (\#Cic3up = 1))\}
\end{aligned}
\tag{7.14}
$$

7.6.3 Experimental Results

Figures 7.21, 7.22, and 7.23 present the analysis results for 3N redundant baseline and rejuvenation models. We performed experiments with varying MTBPM over from one to seven days, and with a $MTTPPM$ of seven minutes, aiming at investigating which value maximizes availability. Figure 7.21 shows that the Steady-State Availability is higher than 99.999% (dashed line) for MTBPM values from 24 to 96 hours (i.e., from one to four days). The comparison with the baseline model is also depicted in Figure 7.21. The BL column exhibits the SSA of the baseline model: 99.99872%. The SSA of the baseline strategy is lower than ones where preventive maintenances are performed in the range from one to seven days.

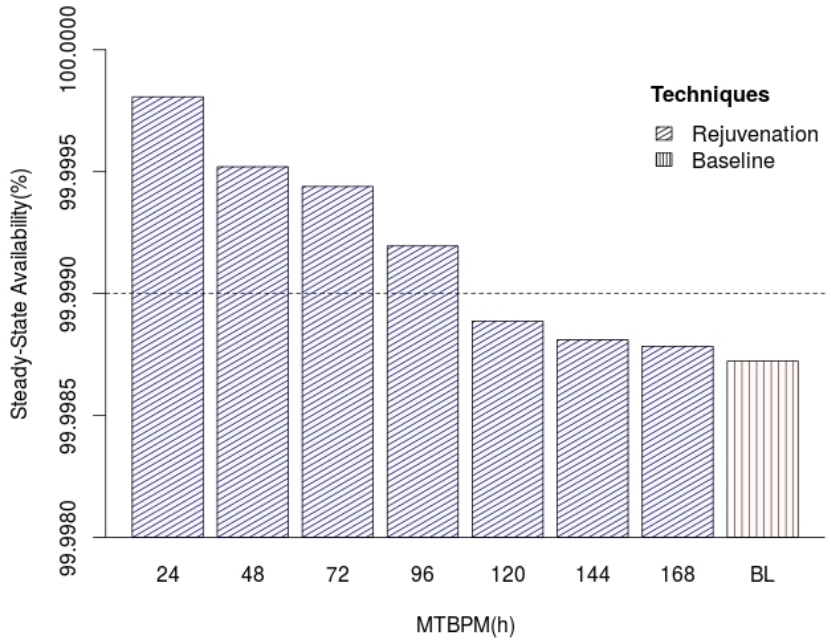


Figure 7.21: Daily MTBPM and corresponding availability

From these results, we explored the SSA behavior around the highest result, i.e., with MTBPM around 24 hours. Figure 7.22 shows that if preventive maintenance occurs in mean time intervals of 23 hours, very high availability can be reached. The availability is higher than 99.9999% (dashed line), with the estimated value of 99.9999131%, which corresponds to 27.404 seconds of annual downtime. So, the optimal point of SSA occurs with a preventive maintenance occurring in a time interval of 23 hours.

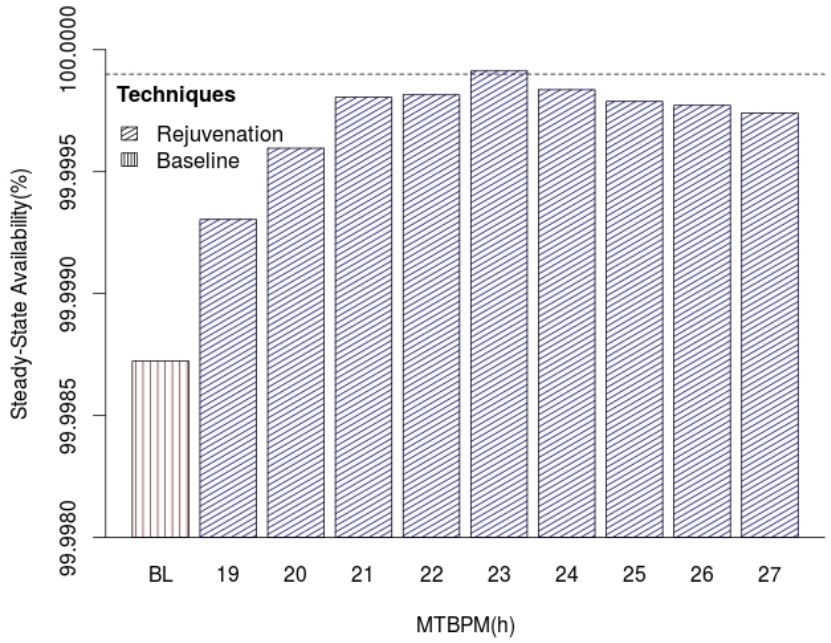


Figure 7.22: Very high availability for MTBPM=23hs

Finally, Fig 7.23 depicts the lower end in which the required five 9's stands. The preventive maintenance should be performed at least every eight hours.

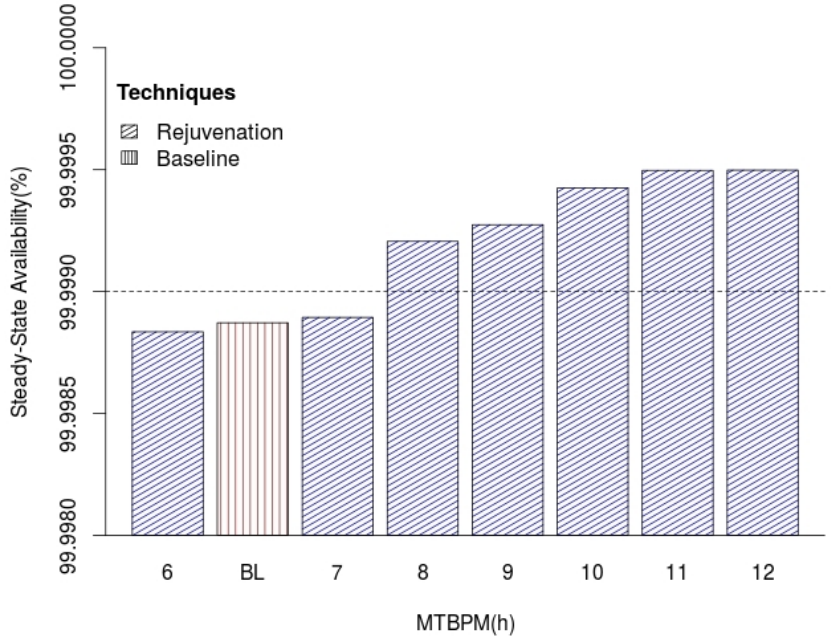


Figure 7.23: High availability with minimum MTBPM=8hs

So, if all the requirements mapped to guard expressions of rejuvenation model were accomplished, and the MTBPM was equal to 23 hours, the annual downtime can be imperceptible

to cloud SFC users. Furthermore, if cloud operators with similar SFCs perform preventive maintenance based on rejuvenation at every four days, they should experience high availability from their openstack Service Function Chains.

7.7 2N Redundant Service Function Chain

This final case study was designed to analyze the benefits of rejuvenation mechanism in a 2N redundant openstack cloud providing Service Function Chains. So, we also propose baseline models without rejuvenation to enable a comparison and state the possible benefits. Six distinct scenarios, listed in Table 7.23, were analyzed.

Scenarios 1 and 4 require two servers as all deployment modes are implemented in all servers; scenarios 2 and 5 require four servers, since the Controller and Neutron deployment modes are implemented together in the same server; and finally, scenarios (3 and 6) requires six servers, one pair for each deployment mode.

Table 7.23: Analyzed scenarios

Id	Configured Openstack Deployment Modes	Rejuvenation	# of Nodes
1	All-In-One	No	2
2	Controller + Neutron and Compute		4
3	Controller, Neutron and Compute		6
4	All-In-One	Yes	2
5	Controller + Neutron and Compute		4
6	Controller, Neutron and Compute		6

7.7.1 Baseline models

The SPN models presented in this subsection does not consider the adoption of rejuvenation mechanism. Figure 7.24 depicts the baseline SPN model for scenario 1.

The places Nd_iRst and the immediate transitions rst_i were added to represent the interaction at recovery occurrences between *(i)* nodes (Nd_iup) and their hosted chains; and *(ii)* nodes and its hosted Open vSwitch.

As soon as a node is recovered, a token is deposited at its correspondent place Nd_iRst. So, the immediate transition rst_i becomes enabled and fires, depositing one token at working places C_iS1 and Cic_iup. The transitions $c1R$, $cic1R$, $c2R$, and $cic2R$ will be enabled only if their correspondent nodes were working, i.e., the places $Nd1up$ and $Nd2up$ have tokens. The guard expressions reflecting these conditions are presented in Table 7.24.

Regarding scenario 1, the system is working when at least one service chain ($C1s1$ or $C2s1$) and its correspondent chain interconnection ($Cic1up$ or $Cic2up$) are working. The working condition is represented in the model by the concomitant presence of tokens at these places. The guard expressions presented in Table 7.24 assure the dependency between the service chain and

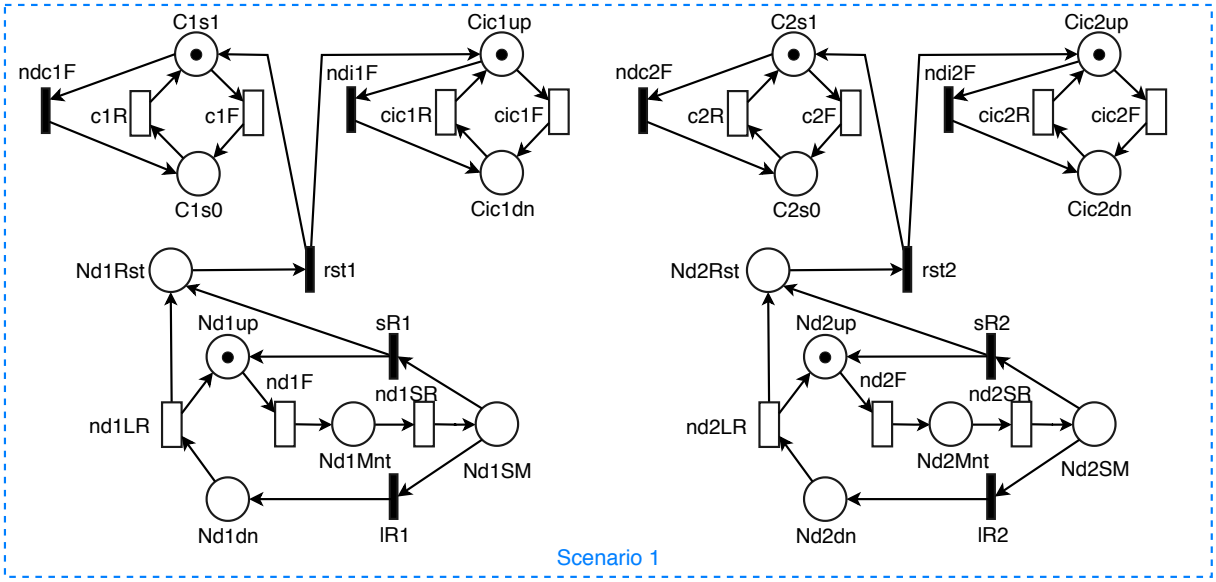


Figure 7.24: SPNs for baseline All-In-One scenario

Table 7.24: Guard expressions for the All-In-One baseline model

Transition	Guard Expressions	Mean time (h)
c1F, c2F	-	chainMTTF
cic1F, cic2F	-	3000
c1R, cic1R	$\#Nd1up > 0$	chainMTTR, 2
c2R, cic2R	$\#Nd2up > 0$	chainMTTR, 2
ndc1F,ndi1F	$\#Nd1Mnt > 0$	-
ndc2F,ndi2F	$\#Nd2Mnt > 0$	-
rst1,rst2	-	-
nd1F	$\#N1Mnt > 0$	allMTTF
nd2F	$\#N2Mnt > 0$	allMTTF
nd1SR, nd2SR	-	0.1166666667
sR1, lR1, sR2, lR2	-	-
nd1LR, nd2LR	-	8

its underlying node and also between the chain interconnection and its underlying node. So, it is not required to consider them to state the system working state.

The Equation

$$SSA_{scn1} = P\{((\#C1s1 = 1)AND(\#Cic1up = 1))OR((\#C2s1 = 1)AND(\#Cic2up = 1))\} \tag{7.15}$$

computes the SSA for scenario 1. Regarding COA, the Equation

$$\begin{aligned} COA_{scn1} = (2 * (P(\#C1s1 = 1)AND(\#Cic1up = 1)AND(\#C2s1 = 1)AND(\#Cic2up = 1)) \\ +1 * ((P(\#C1s1 = 1)AND(\#Cic1up = 1)AND(\#C2s1 = 0)AND(\#Cic2up = 0)) \\ +(P(\#C2s1 = 1)AND(\#Cic2up = 1)AND(\#C1s1 = 0)AND(\#Cic1up = 0))))/2 \end{aligned} \tag{7.16}$$

was adopted to compute it.

Figure 7.25 depicts the baseline SPN model for scenario 2. The places Nd_iup represent Compute nodes. The place $CNup$ represents nodes with the joint controller and neutron deployment modes. The system is working when at least: *(i)* one compute node ($Nd1up$ or $Nd2up$) and their associated service chain ($C1s1$ or $C2s1$) and chain interconnection ($Cic1up$ or $Cic2up$), respectively, are working; *(ii)* one controller/neutron node ($CNup$) is working. The working condition is represented in the model by the concomitant presence of tokens at these places.

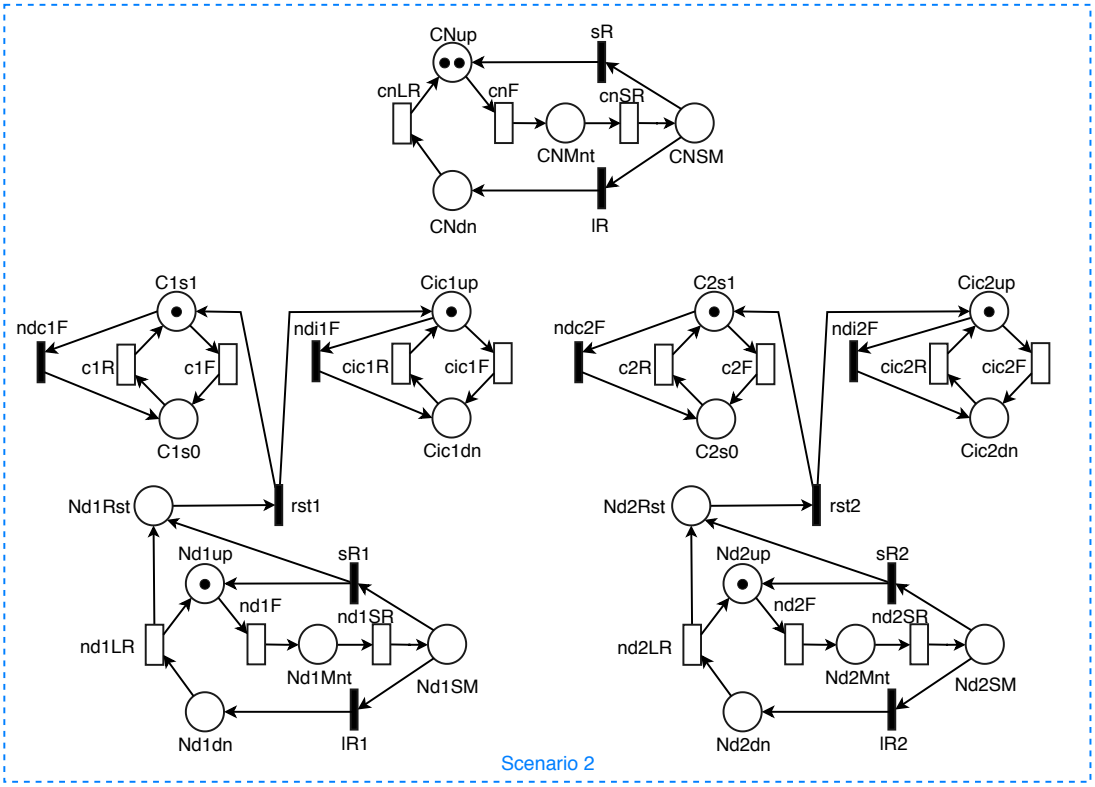


Figure 7.25: SPNs for baseline scenario 2

There were no guard expressions in Controller/Neutron sub-net. Table 7.25 presents the mean times adopted in the timed transitions of the Controller/Neutron sub-net.

Table 7.25: Mean times of timed transitions in scenario 2

Transition	Mean time (h)
cnF	cnMTTF
cnSR	0.1166666667
cnLR	8

The Equation

$$SSA_{scn2} = P\{(\#CNup > 0)AND(((\#C1s1 = 1)AND(\#Cic1up = 1)) \\ OR((\#C2s1 = 1)AND(\#Cic2up = 1)))\} \quad (7.17)$$

computes the SSA for scenario 2. Regarding COA, the Equation

$$\begin{aligned} COA_{scn2} = (2 * (P\{(\#C1s1 = 1)AND(\#Cic1up = 1)AND(\#C2s1 = 1) \\ AND(\#Cic2up = 1)AND(\#CN1up > 0)\}) \\ +1 * ((P\{(\#C1s1 = 1)AND(\#Cic1up = 1)AND(\#C2s1 = 0)AND(\#Cic2up = 0) \\ AND(\#CN1up > 0)\}) \\ +(P\{(\#C2s1 = 1)AND(\#Cic2up = 1)AND(\#C1s1 = 0)AND(\#Cic1up = 0) \\ AND(\#CN1up > 0)\}))/2 \end{aligned} \quad (7.18)$$

was adopted to compute it.

Figure 7.26 depicts the baseline SPN model for scenario 3. The system is working when at least: *(i)* one compute node ($Nd1up$ or $Nd2up$) and their associated service chain ($C1s1$ or $C2s1$) and chain interconnection ($Cic1up$ or $Cic2up$) are working; *(ii)* one controller node is working (Cup); and *(iii)* one neutron node (Nup) is working.

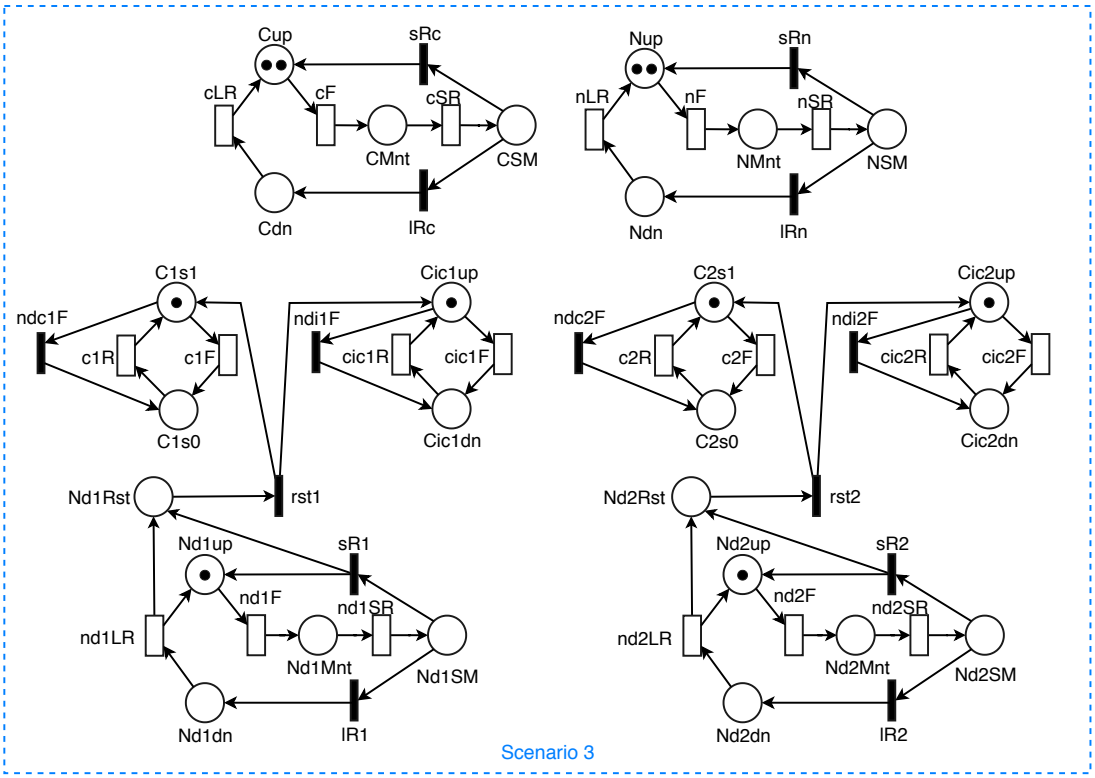


Figure 7.26: SPNs for baseline scenario 3

There were no guard expressions in both Controller and Neutron sub-nets. Table 7.26 presents the mean times adopted in the timed transitions of the Controller and Neutron sub-nets.

The Equation

$$\begin{aligned} SSA_{scn3} = P\{(\#Cup > 0)AND(\#Nup > 0)AND(((\#C1s1 = 1)AND(\#Cic1up = 1))OR \\ ((\#C2s1 = 1)AND(\#Cic2up = 1)))\} \end{aligned} \quad (7.19)$$

Table 7.26: Mean times of timed transitions in scenario 3

Transition	Mean time (h)
cF	cMTTF
nF	nMTTF
cSR, nSR	0.1166666667
cLR, nLR	8

computes the SSA for scenario 3. Regarding COA, the Equation

$$\begin{aligned} COA_{scn3} = (2 * (P\{(\#C1s1 = 1)AND(\#Cic1up = 1)AND(\#C2s1 = 1)AND(\#Cic2up = 1) \\ AND(\#Cup > 0)AND(\#Nup > 0)\}) \\ +1 * ((P\{(\#C1s1 = 1)AND(\#Cic1up = 1)AND(\#C2s1 = 0)AND(\#Cic2up = 0) \\ AND(\#Cup > 0)AND(\#Nup > 0)\}) \\ +(P\{(\#C2s1 = 1)AND(\#Cic2up = 1)AND(\#C1s1 = 0)AND(\#Cic1up = 0) \\ AND(\#Cup > 0)AND(\#Nup > 0)\}))/2 \end{aligned} \tag{7.20}$$

was adopted to compute it.

The numerical results of SSA and COA for scenarios 1, 2, and 3 are presented in Table 7.27.

Table 7.27: SSA and COA for 2N baseline scenarios

Scenario	SSA	COA
1	99.99676967%	98.92080977%
2	99.99747647%	99.04539234%
3	99.99739319%	99.04530916%

These values will be used as a comparison basis to the results considering rejuvenation, presented in the next section.

7.7.2 Rejuvenation models

The models exhibited in Figure 7.27 represent the scenarios 4, 5, and 6. These models adopt VM live migration as rejuvenation technique. The goal is to improve the Steady-State Availability and Capacity Oriented Availability of the 2N redundant openstack infrastructure providing a Service Function Chain.

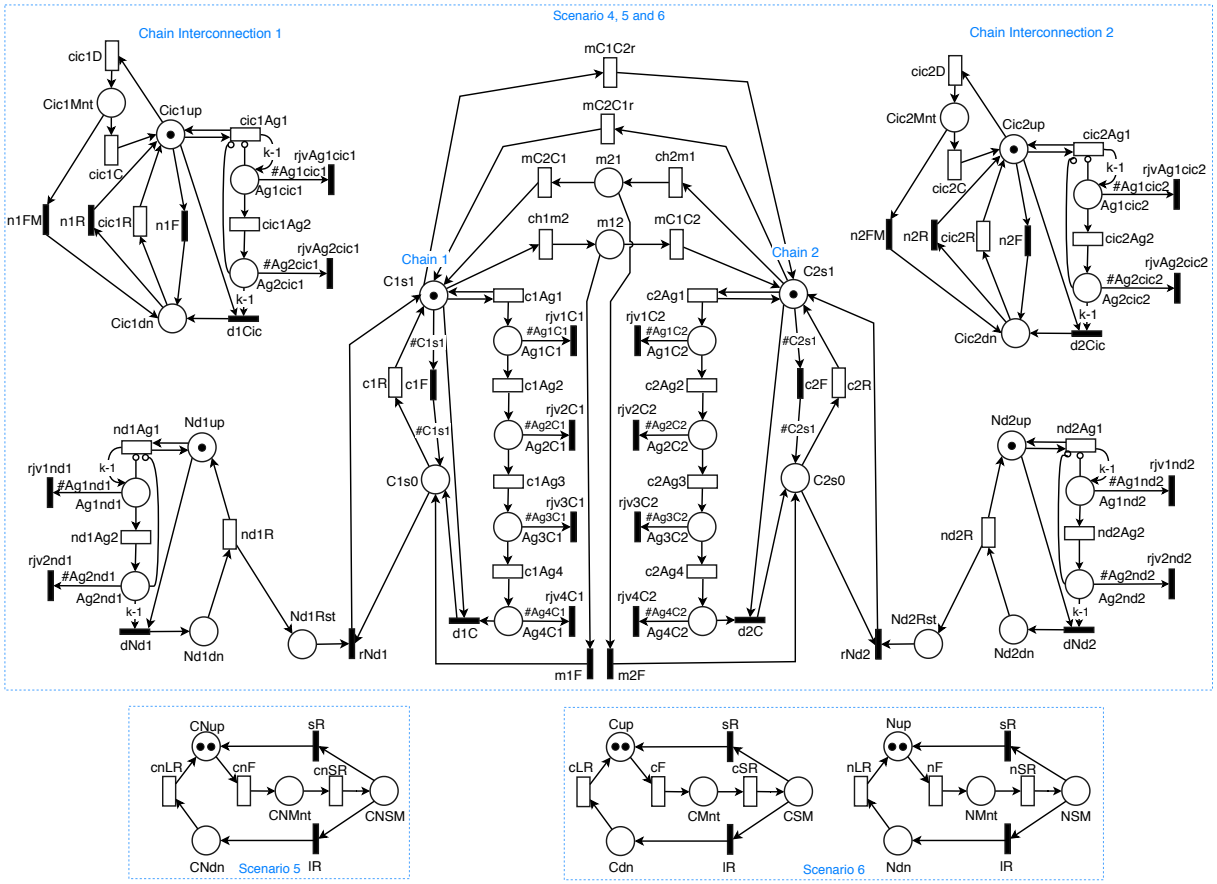


Figure 7.27: SPN Model with Rejuvenation technique

We now consider the SAR of chain interconnections. The sub-model of Figure 6.14 was adopted to represent the Open vSwitch.

During any chain migration, if the source node fails, the transition m_iF fires and any token at places $m12$ or $m21$ is deposited at places $C1s0$ or $C2s0$, respectively. After some migration, the places C_is1 will contain two tokes. So, we isolate aging places, $Ag1C1$, $Ag2C1$, $Ag3C1$, and $Ag4C1$, for chain provided in node 1, and $Ag1C2$, $Ag2C2$, $Ag3C2$, and $Ag4C2$ for chain provided in node 2, enabling the aging detection of the distinct service chains.

The requirements that must be accomplished to allow preventive maintenance through VM live migration in 2N scenarios are identical for those required in 3N scenarios. The list below describes the requirements for a SFC live migration from Node 02 to Node 01:

i. the destination Node 01 is working (#N1up>0);
ii. there is no other migration between these Nodes in process ((#m21=0)AND(#m12=0));
iii. the destination chain interconnection is working (#Cic1up=1);
iv. there are no failure chains is the destination Node (#C1s0=0);
v. there is only one chain in Node 02 (#C2s1<2);
vi. there are no failure chains in the source Node (#C2s0=0).

Senario 4 represents the All-In-One deployment mode. The tokens in places $Nd1up$ and

$Nd2up$ represent the two physical servers and all the associated required softwares, as depicted in RBD of Figure 6.6e.

Scenario 5 represents the Controller + Neutron and Compute deployment mode. For this scenario, the tokens in the place $CNup$ represent the two controller+neutron servers and all the associated required softwares, as depicted in RBD of Figure 6.6d, while the tokens in places $Nd1up$ and $Nd2up$ represent the two compute servers and all the associated required softwares, as depicted in Figure 6.6c.

Finally, scenario 6 represents the Controller, Neutron, and Compute deployment mode. The tokens presented in the place Cup represent the two controller servers and all the associated required softwares, as depicted in RBD of Figure 6.6a, whereas he tokens presented in the place Nup represent the two neutron servers and all the associated required softwares, as depicted in RBD of Figure 6.6b. The tokens in places $Nd1up$ and $Nd2up$ have an equivalent meaning of scenario 5.

Table 7.28 summarizes the guard expressions and the mean time of the transitions present in the rejuvenation models.

Table 7.28: Guard expressions for rejuvenation models

Transition	**Guard Expression**	**Mean time (h)**
nd1Ag1	$(\#C2s1+\#C2s0<2)$	{nd,all}MTTF/k
nd1Ag2	-	{nd,all}MTTF/k
nd1R	-	{nd,all}MTTR
rjv1nd1	$(\#Ag1Nd1>0)AND(\#C2s1>1)$	-
rjv2nd1	$(\#Ag2Nd1>0)AND(\#C2s1>1)$	-
dNd1	-	-
c1Ag1	$((\#C1s1+\#C1s0==1)AND$ $(\#Ag1C1+\#Ag2C1+\#Ag3C1+\#Ag4C1==0))OR$ $((\#C1s1+\#C1s0==2)AND$ $(\#Ag1C1+\#Ag2C1+\#Ag3C1+\#Ag4C1<2)$ $AND(\#C1s0==0))$	((k-1)* chainMTTF)/k
c1Ag$_{(2<=i<=4)}$	-	chainMTTF/k
c1R	$\#Nd1up>0$	chainMTTR
c1F	$(\#Nd1dn=1)AND(\#Cic1up=0)AND(\#C1s1>0)$	-
d1C	-	-
rjv$_i$C1	$(\#AgiC1>0)AND((\#C2s1>1)OR$ $((\#C1s1+\#C1s0>0)AND(\#Nd1dn>0))OR$ $((\#C2s1+\#C2s0=2)AND(\#Nd2dn>0)))$	-
cic1Ag$_j$	-	chainMTTF/k
rjvAg$_j$cic1	$(\#Agjcic1>0)AND((\#C2s1>1)OR$ $((\#C1s1+\#C1s0>0)AND(\#Nd1dn>0))OR$ $((\#C2s1+\#C2s0=2)AND(\#Nd2dn>0)))$	-

Continued on next page

Table 7.28 – *Continued from previous page*

Transition	**Guard Expression**	**Mean time (h)**
d1Cic	-	-
cic1D	$(\#m12 = 1)OR(\#C2s1 > 1)$	0.0017075
cic1C, cic1R, n1R	$((\#C1s1 + \#C1s0 > 0)AND(\#Nd1up > 0))OR$ $((\#C2s1 + \#C2s0 = 2)AND(\#Nd2up > 0))$	0.00335805556
n1F, n1FM	$((\#C1s1 + \#C1s0 > 0)AND(\#Nd1dn > 0))OR$ $((\#C2s1 + \#C2s0 = 2)AND(\#Nd2dn > 0))$	cicMTTF
nd2Ag1	$(\#C1s1 + \#C1s0 < 2)$	{nd,all}MTTF/k
nd2Ag2	-	{nd,all}MTTF/k
nd2R	-	{nd,all}MTTR
rjv1nd2	$(\#Ag1Nd2 > 0)AND(\#C1s1 > 1)$	-
rjv2nd2	$(\#Ag2Nd2 > 0)AND(\#C1s1 > 1)$	-
dNd2	-	-
c2Ag1	$((\#C2s1 + \#C2s0 == 1)AND$ $(\#Ag1C2 + \#Ag2C2 + \#Ag3C2 + \#Ag4C2 == 0))OR$ $((\#C2s1 + \#C2s0 == 2)AND$ $(\#Ag1C2 + \#Ag2C2 + \#Ag3C2 + \#Ag4C2 < 2)$ $AND(\#C2s0 == 0))$	((k-1)* chainMTTF)/k
$c2Ag_{(2<=i<=4)}$	-	chainMTTF/k
c2R	$\#Nd2up > 0$	chainMTTR
c2F	$(\#Nd2dn = 1)AND(\#Cic2up = 0)AND(\#C2s1 > 0)$	-
d2C	-	-
$rjv_{(1<=i<=4)}C1$	$(\#AgiC2 > 0)AND((\#C1s1 > 1)OR$ $((\#C2s1 + \#C2s0 > 0)AND(\#Nd2dn > 0))$ $OR((\#C1s1 + \#C1s0 = 2)AND(\#Nd1dn > 0)))$	-
$cic2Ag_j$	-	chainMTTF/k
$rjvAg_jcic2$	$(\#Ag1cic2 > 0)AND((\#C1s1 > 1)OR$ $((\#C2s1 + \#C2s0 > 0)AND(\#Nd2dn > 0))$ $OR((\#C1s1 + \#C1s0 = 2)AND(\#Nd1dn > 0)))$	-
d2Cic	-	-
cic2D	$(\#m21 = 1)OR(\#C1s1 > 1)$	0.0017075
cic2C, cic2R, n2R	$((\#C2s1 + \#C2s0 > 0)AND(\#Nd2up > 0))OR$ $((\#C1s1 + \#C1s0 = 2)AND(\#Nd1up > 0))$	0.00335805556
n2F, n2FM	$((\#C2s1 + \#C2s0 > 0)AND(\#Nd2dn > 0))OR$ $((\#C1s1 + \#C1s0 = 2)AND(\#Nd1dn > 0))$	cicMTTF
ch1m2	$(\#Nd2up > 0)AND(\#m21 = 0)AND(\#m12 = 0)AND$ $(\#Cic2up = 1)AND(\#C1s1 < 2)AND(\#C1s0 = 0)$	MTBPM
mC1C2	$(\#Nd2up > 0)AND(\#Cic1up = 0)$	0.1484863888
mC2C1r	$(\#Cic1up = 0)AND(\#Nd1up > 0)AND(\#C2s1 > 1)$	0.0318197222

Continued on next page

Table 7.28 – *Continued from previous page*

Transition	Guard Expression	Mean time (h)
ch2m1	$(\#Nd1up > 0)AND(\#m12 = 0)AND$ $(\#m21 = 0)AND(\#Cic1up = 1)$ $AND(\#C2s1 < 2)AND(\#C2s0 = 0)$	MTBPM
mC2C1	$(\#CMP1up > 0)AND(\#Cic2up = 0)$	0.1484863888
mC1C2r	$(\#Cic2up = 0)AND(\#CMP2up > 0)AND(\#C1s1 > 1)$	0.0318197222

Table 7.29 presents the SSA and COA equations of the analyzed aging/rejuvenation configurations.

Table 7.29: SSA e COA equations for rejuvenation models

Scn.	Mtr.	Equation
4	SSA	P{((#C1s1=2)AND((#Cic1up=1)OR(#Cic2up=1)))OR((#C2s1=2) AND((#Cic1up=1)OR(#Cic2up=1))) OR((#C1s1=1)AND(#Cic1up=1))OR((#C2s1=1)AND(#Cic2up=1))}
5		P{(((#C1s1=2)AND((#Cic1up=1)OR(#Cic2up=1)))OR ((#C2s1=2) AND((#Cic1up=1)OR(#Cic2up=1))) OR((#C1s1=1)AND(#Cic1up=1)) OR ((#C2s1=1)AND(#Cic2up=1)))AND(#CNup>0)}
6		P{(((#C1s1=2)AND((#Cic1up=1)OR(#Cic2up=1)))OR((#C2s1=2) AND((#Cic1up=1)OR(#Cic2up=1)) OR((#C1s1=1)AND(#Cic1up=1)) OR((#C2s1=1)AND(#Cic2up=1))) AND(#Cup>0)AND(#Nup>0)}
4	COA	(2*((P{(#C1s1=1)AND(#Cic1up=1)AND(#C2s1=1)AND(#Cic2up=1)}) +(P{(#C1s1=2)AND(#Cic1up=1)AND(#Cic2up=1)})+(P{(#C2s1=2) AND(#Cic1up=1)AND(#Cic2up=1)}) +1*((P{(#C1s1=1)AND(#Cic1up=1)AND(#C2s1=0)AND(#Cic2up=0)}) +(P{(#C2s1=1)AND(#Cic2up=1)AND(#C1s1=0)AND(#Cic1up=0)})))/2
5		(2*((P{(#C1s1=1)AND(#Cic1up=1)AND(#C2s1=1)AND(#Cic2up=1) AND(#CNup>0)}) +(P{(#C1s1=2)AND(#Cic1up=1)AND(#Cic2up=1)AND(#CNup>0)}) +(P{(#C2s1=2)AND(#Cic1up=1)AND(#Cic2up=1)AND(#CNup>0)})) +1*((P{(#C1s1=1)AND(#Cic1up=1)AND(#C2s1=0)AND(#Cic2up=0) AND(#CNup>0)}) +(P{(#C1s1=0)AND(#Cic1up=0)AND(#C2s1=1)AND(#Cic2up=1) AND(#CNup>0)})))/2
6		(2*((P{(#C1s1=1)AND(#Cic1up=1)AND(#C2s1=1)AND(#Cic2up=1) AND(#Cup>0)AND(#Nup>0)}) +(P{(#C1s1=2)AND(#Cic1up=1)AND(#Cic2up=1) AND(#Cup>0)AND(#Nup>0)}) +(P{(#C2s1=2)AND(#Cic1up=1)AND(#Cic2up=1) AND(#Cup>0)AND(#Nup>0)})) +1*((P{(#C1s1=1)AND(#Cic1up=1)AND(#C2s1=0)AND(#Cic2up=0) AND(#Cup>0)AND(#Nup>0)}) +(P{(#C1s1=0)AND(#Cic1up=0)AND(#C2s1=1)AND(#Cic2up=1) AND(#Cup>0)AND(#Nup>0)})))/2

7.7.3 Experimental results

This section presents the results regarding the scenarios defined previously in Table 7.23. The main goal is to identify and measure how a rejuvenation technique can improve the

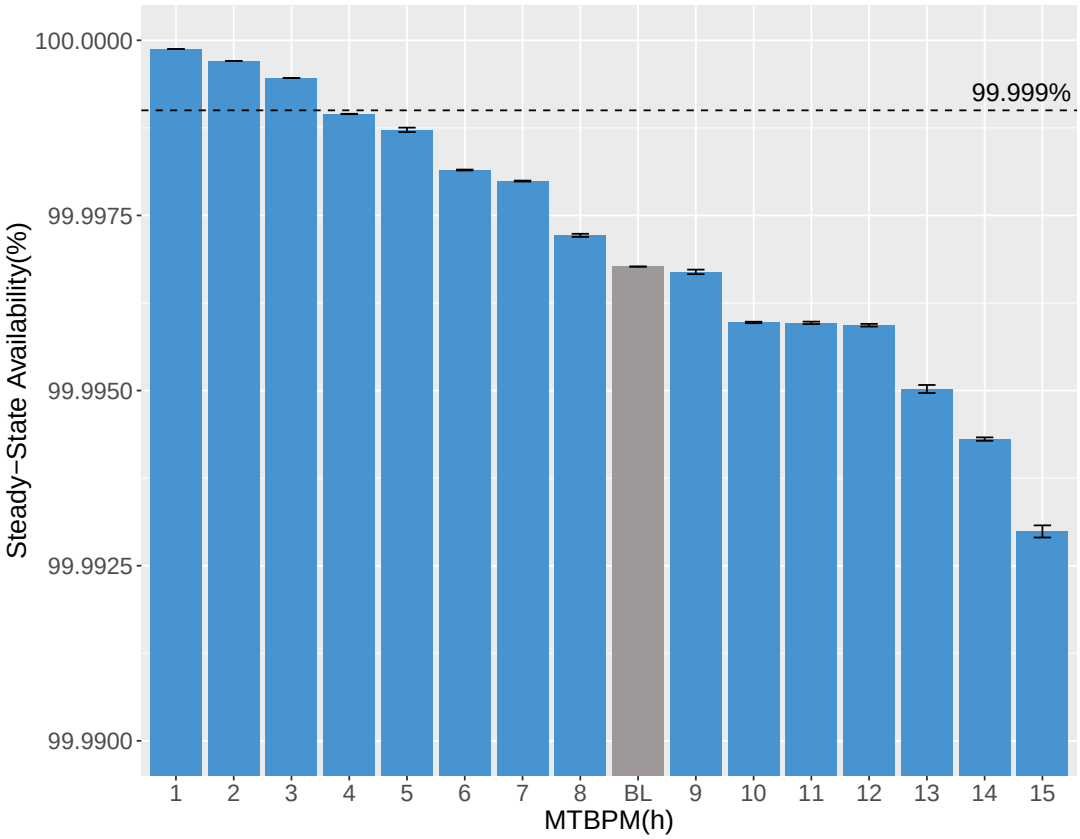


Figure 7.28: SSA for All-In-One configuration (scenarios 3 and 6)

availability of service chains and understand how the variation of the MTBPM helps to achieve high availability.

Figure 7.28 shows the SSA of All-In-One configuration. In this case, the high availability (five 9's) is reached when the preventive maintenance occurs at intervals of 3 hours at maximum. When using the baseline approach, the configuration did not achieve high availability, but reached a good availability (about 99.9967696%), whereas with rejuvenation, adopting MTBPM = 3 hours, the SSA reaches 99.9994627%. The goal of high availability is reached with the adoption of VM live migration rejuvenation.

Figure 7.29 illustrates the COA for All-In-One configuration.

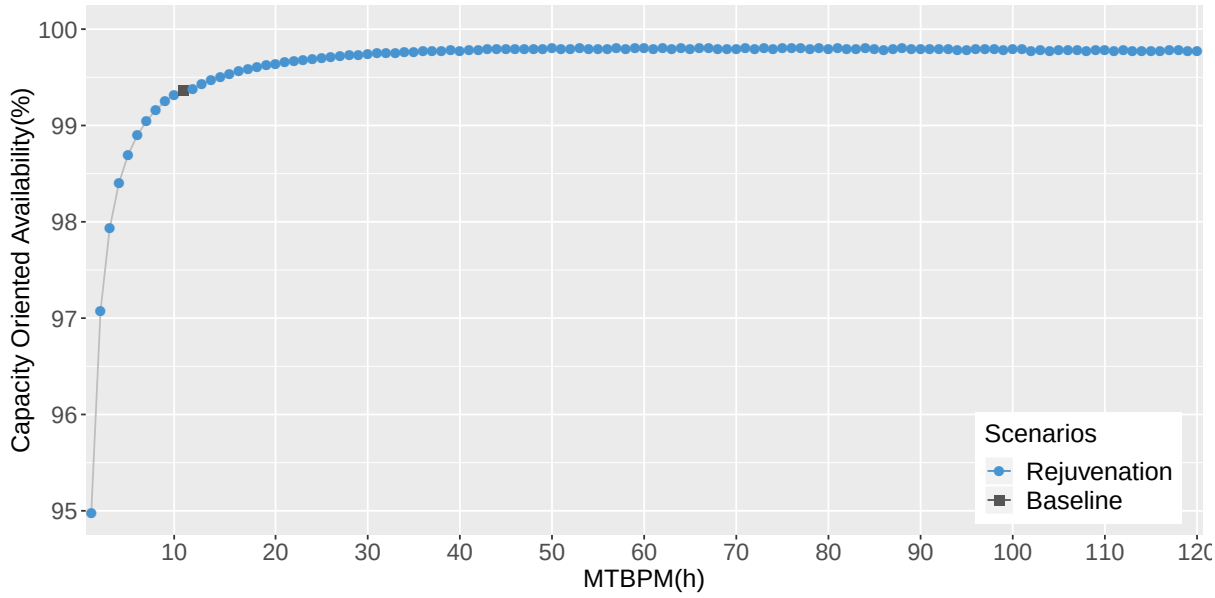


Figure 7.29: COA for All-In-One Configuration (scenarios 3 and 6)

In this case, the baseline approach achieves a COA close to 98.92080977% (see Table 7.27), and the preventive maintenance produces better results than the baseline when the MTBPM is greater than 11 hours. This way, if the service provider prioritizes the high availability, the COA will be closed to 97.9314570%.

The SSA and COA results of scenarios 2 and 5 are depicted in Figs. 7.30 and 7.31.

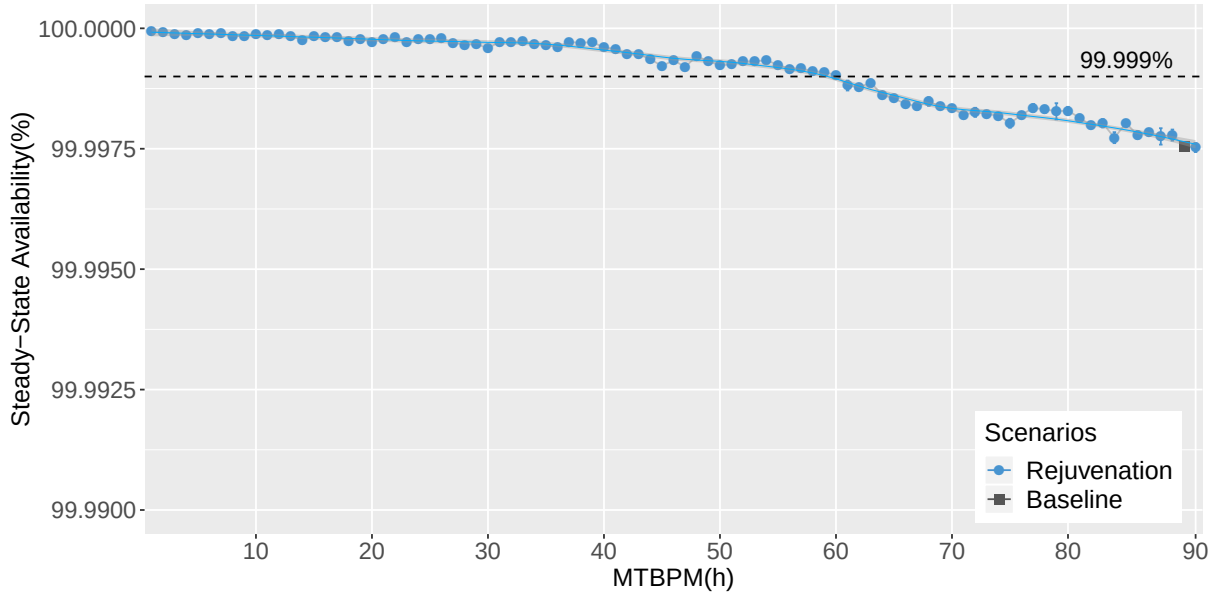


Figure 7.30: SSA for Controller/Neutron Configuration (scenarios 2 and 5)

When deploying the controller and the neutron at the same server, the primary benefit is the great improvement of the maximum MTBPM to achieve high availability, increasing from 3 hours in the scenario 4 to 60 hours in the scenario 5. This represents a good achievement from the provider perspective because as more as one can prolong the next maintenance, fewer resources

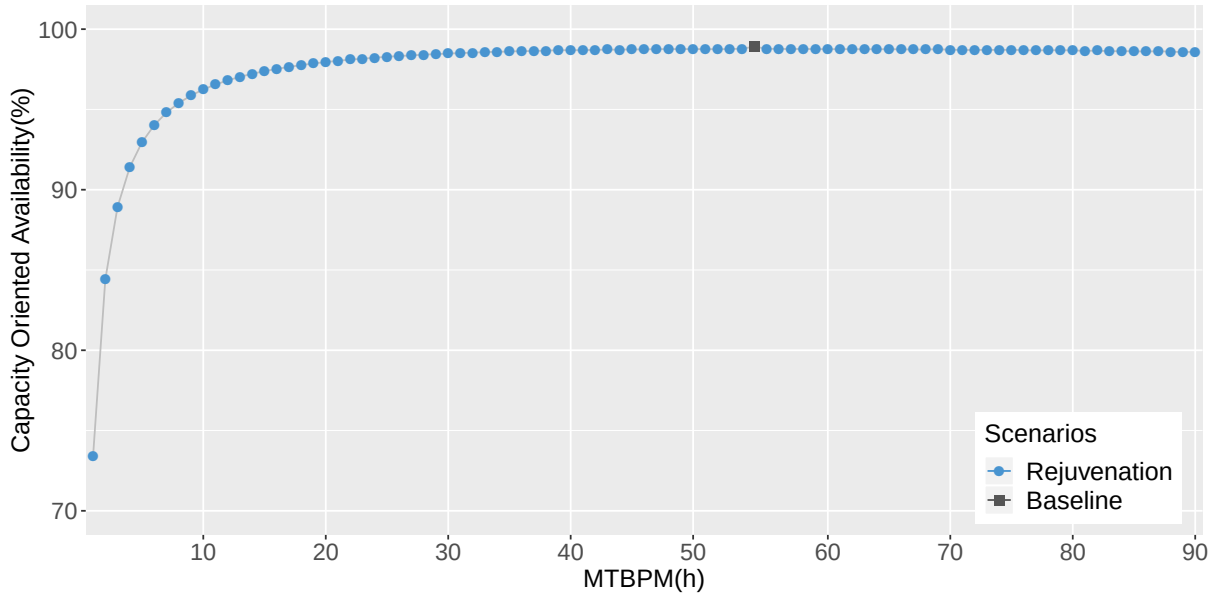


Figure 7.31: COA for Controller/Neutron Configuration (scenarios 2 and 5)

are wasted. On the other hand, when using this configuration ($MTBPM = 60 hours$), the baseline approach obtained better COA (99.04539234%) result than the preventive maintenance approach, and if the provider focus on high availability, the COA will be 98.7542228%.

Finally, Figs. 7.32 and 7.33 depict the results of scenarios 3 and 6. In this case, when six servers are adopted in a 2N redundancy for each openstack deployment mode, the maximum MTBPM to provide high availability was 78 hours. Compared to scenario 5, the improvement in the MTBPM was 18 hours.

Different from the other previous scenarios, any MTBPM in a time interval from 1 to 120 hours produces better COA results than baseline, as depicted in Figure 7.33.

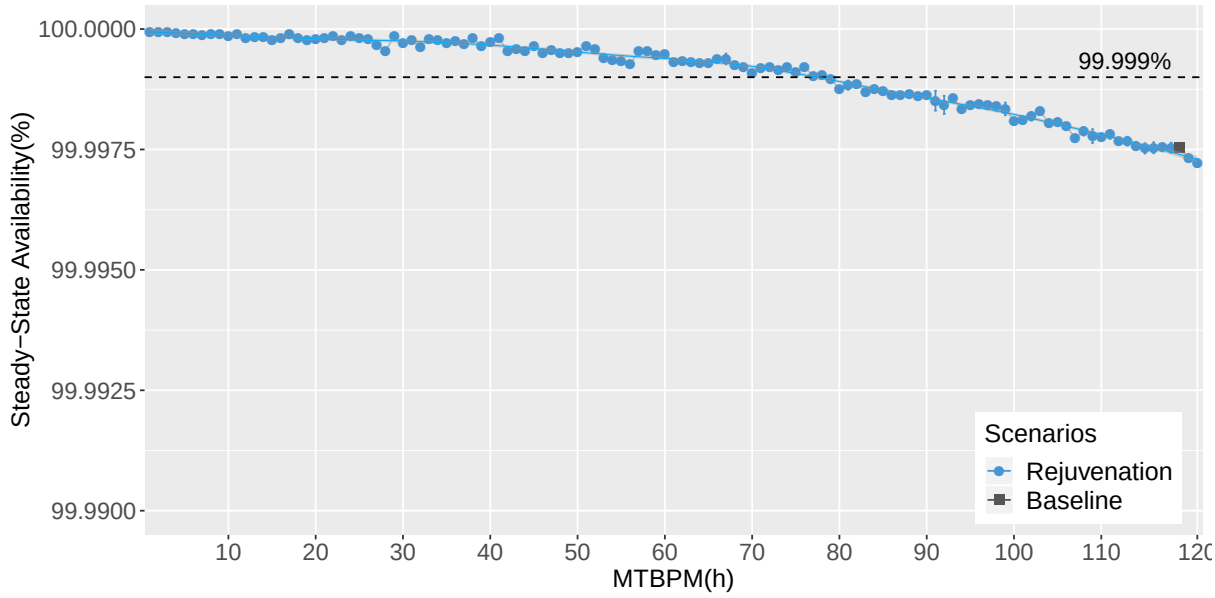


Figure 7.32: SSA for Controller/Neutron/Compute Configuration (Scenarios 3 and 6)

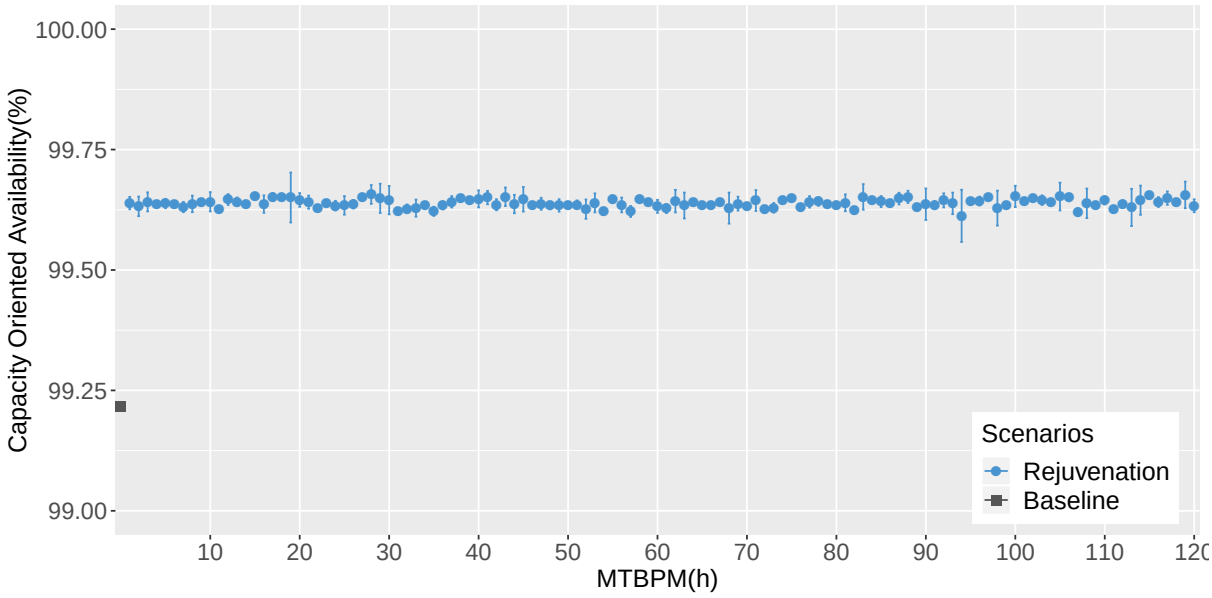


Figure 7.33: COA for Controller/Neutron/Compute Configuration (Scenarios 3 and 6)

The results of this final case study demonstrate the effectiveness in the adoption of VM live migration rejuvenation technique for achieving high availability, i.e., a Steady-State Availability greater than 99.999%.

7.8 Final Remarks

This chapter presented the results obtained by the designed dependability models. The first two case studies showed the benefits, over the studied metrics, in adopting a strategy of eliminating single points of failure in Proxmox server virtualization environment. The third case studied showed how reasonable was our proposed modeling approach in the estimation of availability in Service Function Chains through a comparison with previous literature work. The fourth and fifth case studies showed the improvements in steady-state availability and capacity oriented availability with the joint adoption of redundancy and software rejuvenation based on service function chain live migration. Therefore, through the described studies, it was possible to demonstrate how network administrators and analysts can make decisions in the provision of high available virtual network chains.

8

Conclusion

The changes in the communication networks coming from virtualization enable the replacement of static equipment and interconnections by dynamic virtualized components. The traditional appliances have been replaced by Virtual Network Functions, decoupling the network functions from their underlying hardware. Virtualized networks represent additional obstacles to high availability because it results in more layers of software: the increasing number of software components required to run virtualized systems also increases the number of possible failures. A widely used design principle in fault tolerance is to introduce redundancy to enhance systems availability. Thus, the modeling and analysis of redundant mechanisms in virtualized environments, as cloud computing infrastructures, is relevant to enable high availability and business continuity in concomitance with resources wastage reduction or elimination.

In this work, we analyzed the steady-state availability and capacity oriented availability metrics of Virtual Network Functions using dependability modeling, with the adoption of hierarchical and heterogeneous modeling.

Testbeds are designed and assembled with the primary purpose of understanding the operation of high availability infrastructures. After, they also were used to enable the measurement of parameters used as inputs into the proposed models. We have argued that these measures can be used by specialists with the aim of improving the precision of models related to the focus of this thesis.

We presented 5 case studies in which the models' applicability can be observed. The first case study focuses on the analysis of server virtualization environments to investigate the benefits in the adoption of active/active redundancy in a cluster of video caches. The second analyzes the joint operation of virtualized network function and a traditional appliance. Sensitivity analysis was adopted in these two case studies to state the most impactful parameters regarding the evaluated metrics.

The third case study aims at analyzing how reasonable is our proposed modeling approach regarding Service Function Chains through a comparison with previous literature. The fourth case study analyzes the benefits in tackling the aging phenomenon presented in an HA cloud infrastructure adopting rejuvenation through VM live migration in a 3N redundant SFC. Finally, the fifth case study analyzes the behavior of the metrics of interest facing a reduction for a 2N

redundant environment in the cloud infrastructure.

The following sections describe the main contributions of this thesis, the limitations, and proposes possible extensions as future directions.

8.1 Contributions

As the main contributions of this thesis, we can highlight:

- a hierarchical and heterogeneous stochastic set of models to represent Virtual Network Function and Service Function Chains, including redundancy for high available services provision, as well as the aging phenomenon analysis and the adoption of its countermeasure rejuvenation mechanism based on SFCs live migration;
- a High Available (HA) private cloud computing was assembled using openstack, enabling the measurement of Mean Time To Chain Live Migration (MTTCLM) parameter in Service Function Chains experiments. The main goal was to feed the presented stochastic models also with real measured values;
- the analysis of the proposed VNF and SFC models, improving the previously known results regarding the number of physical servers required to provide an HA VNF chain.

Also, we extend the Tuning Infrastructure for Server Virtualization Experiment Protocol (TISVEP) with messages specifications (reported in Appendix B), as well as with its coding, automating the experiments execution of Service Function Chains live migration.

8.2 Limitations

During the period of this work, some limitations were identified. They are reported below.

- Models growth: even with the division of modeled systems in sub-models, the models tend to grow in the analysis of distinct scenarios, making it difficult to generate results due to processing times;
- HA Cloud management: the cost of assembling the HA cloud computing infrastructure, comprises the training regarding installation, configuration, and operation of openstack cloud, results in a high time-consuming task. The storage and network hardware assembling comprised the acquisition of equipment, their subsequent installation, configuration, and test together with the server hardware, the operating system, and the openstack software. Updates in the infrastructure, even performed automatically, resulted in re-configuration efforts. So, after the deployment of Ocata version of openstack software, no updates were implemented;

- Adopted models and techniques: the user must have basic knowledge regarding the parameterization of the system variables. The users must receive proper training or a system that aids the users should be created.

8.3 Future Work

Although this thesis tackles some issues regarding dependability in the provision of VNFs and SFCs, there are possibilities to improve and extend the current work. The following items summarize some possible improvements.

- Explore parallel live migration techniques: the migration between source and destination server was performed in series. The adoption of parallel live migration is a potential source to reach lower MTTCLM values. The models can be adjusted to cover parallel migration updating the firing semantics of the involved transitions from Single Server to Infinite Server;

- Explore alternative virtualization approaches, such as: para-virtualization (with Xen) and Container-based virtualization (with LXC). It is also a potential source to reduce MTTCLM values, consequently reducing the downtime due to VNFs and SFCs migration times.

- Consider different repair policies. Policies based on component replacement, different number of repair team size, and different average failure times. Cold and warm standby policies are also suitable alternatives that may be evaluated by analytical models. Moreover, by combining cost models, this work could be expanded by considering scenarios that evaluate the violation of a contracted service levels.

- Besides redundancy and live migration software rejuvenation, explore additional approaches for SSA and COA improvements, such as elasticity. It is possible to allocate and release replicas of VNFs and SFCs at periods of traffic increasing, as well as release replicas at periods of traffic decreasing.

- Analyze different sizes of Service Function Chains, aiming to establish an upper boundary in the number of chain services that do not violate the high availability feature (five 9's of availability).

- Automatize the live migration software rejuvenation mechanism in TISVEP protocol, allowing the implementation of the adopted rejuvenation approach also in real HA cloud infrastructures.

References

[1] A. Abdelsalam, F. Clad, C. Filsfils, S. Salsano, G. Siracusano, and L. Veltri. Implementation of virtual network function chaining through segment routing in a linux-based nfv infrastructure. In *2017 IEEE Conference on Network Softwarization (NetSoft)*, pages 1–5, July 2017.

[2] A. Abhari and M. Soraya. Workload generation for youtube. *Multimedia Tools and Applications*, 46(1):91–118, 2010.

[3] L. A. Adamic and B. A. Huberman. Zipf's law and the internet. *Glottometrics*, 3:143–150, 2002.

[4] W. Ahmad, O. Hasan, and S. Tahar. Formal dependability modeling and analysis: A survey. *CoRR*, abs/1606.06877, 2016.

[5] M. Ajmone Marsan, G. Conte, and G. Balbo. A class of generalized stochastic petri nets for the performance evaluation of multiprocessor systems. *ACM Trans. Comput. Syst.*, 2(2):93–122, May 1984.

[6] J. Alonso, R. Matias, E. Vicente, A. Maria, and K. S. Trivedi. A comparative experimental study of software rejuvenation overhead. *Performance Evaluation*, 70(3):231–250, 2013.

[7] J. Alonso and K. S. Trivedi. Software Rejuvenation And Its Application In Distributed Systems, 2015.

[8] I. Amazon Web Services. About aws, 2017.

[9] I. Amazon Web Services. Summary of the amazon s3 service disruption in the northern virginia (us-east-1) region, 2017.

[10] I. Amazon Web Services. What is cloud computing?, 2017.

[11] S. Andriole, M. Conrad, P. Cosi, R. Debons, D. Heimann, N. Mittal, M. Palakal, and K. Trivedi. *Advances in Computers*, volume 31. Academic Press, 1990.

[12] J. Araujo, R. Matos, P. Maciel, R. Matias, and I. Beicker. Experimental evaluation of software aging effects on the eucalyptus cloud computing infrastructure. pages 1–7, 2012.

[13] J. Araujo, R. Matos, P. Maciel, F. Vieira, R. Matias, and K. S. Trivedi. Software rejuvenation in eucalyptus cloud computing infrastructure: A method based on time series forecasting and multiple thresholds. In *2011 IEEE Third International Workshop on Software Aging and Rejuvenation*, pages 38–43, Nov 2011.

[14] J. Arlat, K. Kanoun, and J. C. Laprie. Dependability modeling and evaluation of software fault-tolerant systems. *IEEE Transactions on Computers*, 39(4):504–513, Apr 1990.

[15] S. Ballmer. Steve ballmer: Worldwide partner conference 2013 keynote, 2013.

[16] K. Benz and T. M. Bohnert. Impact of pacemaker failover configuration on mean time to recovery for small cloud clusters. *IEEE International Conference on Cloud Computing, CLOUD*, pages 384–392, 2014.

[17] G. Bolch, S. Greiner, H. de Meer, and K. S. Trivedi. *Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications*. Wiley-Interscience, New York, NY, USA, 1998.

[18] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web caching and zipf-like distributions: evidence and implications. In *INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 1, pages 126–134 vol.1, Mar 1999.

[19] F. Callegati, W. Cerroni, and C. Contoli. Virtual networking performance in openstack platform for network function virtualization. *Journal of Electrical and Computer Engineering*, 2016, 2016.

[20] V. Cardellini, E. Casalicchio, J. C. Estrella, and K. R. Lucas Jaquie Cast Branco. *Performance and Dependability in Service Computing: Concepts, Techniques and Research Directions*. Information Science Reference - Imprint of: IGI Publishing, Hershey, PA, 2011.

[21] F. Carpio and A. Jukan. Improving reliability of service function chains with combined vnf migrations and replications. *arXiv preprint arXiv:1711.08965*, 2017.

[22] R. Chayapathi, S. Hassan, and P. Shah. *Network Functions Virtualization (NFV) with a Touch of SDN*. Pearson Education, 2016.

[23] H. Chirammal, P. Mukhedkar, and A. Vettathu. *Mastering KVM Virtualization*. Packt Publishing, 2016.

[24] L.-D. Chou, C.-W. Tseng, H.-Y. Chou, and Y.-T. Yang. A sfc network management system in sdn. *Mobile and Wireless Technologies 2017*, pages 360–369, 06 2018.

[25] Cisco. Cisco visual networking index: Forecast and methodology, 2015-2020 white paper, 2016.

[26] I. Citrix Systems. Citrix xen server. `https://www.citrix.com/products/xenserver/`, 2017.

[27] J. claude Laprie and B. Randell. Fundamental concepts of computer systems dependability. In *Proc. of the Workshop on Robot Dep. , Seoul, Korea*, pages 21–22, 2001.

[28] CloudStack. Apache cloudstack. `https://cloudstack.apache.org/`, 2017.

[29] I. Costa, J. Araujo, J. Dantas, E. Campos, F. A. Silva, and P. Maciel. Availability evaluation and sensitivity analysis of a mobile backend-as-a-service platform. *Quality and Reliability Engineering International*, 32(7):2191–2205, 2016. qre.1927.

[30] D. Cotroneo, L. De Simone, A. K. Iannillo, A. Lanzaro, and R. Natella. Dependability evaluation and benchmarking of Network Function Virtualization Infrastructures. *1st IEEE Conference on Network Softwarization: Software-Defined Infrastructures for Networks, Clouds, IoT and Services, NETSOFT 2015*, pages 1–9, 2015.

[31] R. d. S. Matos, P. R. M. Maciel, F. Machida, D. S. Kim, and K. S. Trivedi. Sensitivity analysis of server virtualized system availability. *IEEE Transactions on Reliability*, 61(4):994–1006, Dec 2012.

[32] J. Dantas, R. Matos, J. Araujo, and P. Maciel. Eucalyptus-based private clouds: availability modeling and comparison to the cost of a public cloud. *Computing*, 97(11):1121–1140, Nov 2015.

[33] M. Di Mauro, M. Longo, F. Postiglione, and M. Tambasco. *Availability Modeling and Evaluation of a Network Service Deployed via NFV*, pages 31–44. Springer International Publishing, Cham, 2017.

[34] R. Dittner and D. Rule. *The Best Damn Server Virtualization Book Period: Including Vmware, Xen, and Microsoft Virtual Server*. Syngress Publishing, 2007.

[35] P. T. Endo, M. Rodrigues, G. E. Gonçalves, J. Kelner, D. H. Sadok, and C. Curescu. High availability in clouds: Systematic review and research challenges. *J. Cloud Comput.*, 5(1):66:1–66:15, Dec. 2016.

[36] ETSI. Network functions virtualisation (nfv); architectural framework, 2014.

[37] ETSI. Network functions virtualisation (nfv); resiliency requirements, 2015.

[38] E. T. S. I. ETSI. Network functions virtualisation – introductory white paper. Technical report, ETSI, 2012.

[39] I. F5 Networks. The mean time between failures for f5 hardware platforms, July 2016.

[40] D. Farinacci, T. Li, S. Hanks, D. Meyer, and P. Traina. Generic routing encapsulation (gre). RFC 2784, RFC Editor, March 2000. `https://www.ietf.org/rfc/rfc2784.txt`.

[41] N. Feamster, J. Rexford, and E. Zegura. The road to sdn: An intellectual history of programmable networks. *SIGCOMM Comput. Commun. Rev.*, 44(2):87–98, Apr. 2014.

[42] S. Fernandes, E. Tavares, M. Santos, V. Lira, and P. Maciel. Dependability assessment of virtualized networks. In *2012 IEEE International Conference on Communications (ICC)*, pages 2711–2716, June 2012.

[43] L. Foundation. Open vswitch. `http://openvswitch.org/`, 2017.

[44] O. Foundation. Accelerating nfv delivery with openstack. Technical report, 2016.

[45] O. N. Foundation. Openflow specifications. `https://www.opennetworking.org/software-defined-standards/specifications/`, 2017.

[46] H. Fujita, Y. Matsuno, T. Hanawa, M. Sato, S. Kato, and Y. Ishikawa. Ds-bench toolset: Tools for dependability benchmarking with simulation and assurance. In *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2012)*, pages 1–8, June 2012.

[47] R. Fundation. The r project for statistical computing, 2016.

[48] M. Gagnaire, F. Diaz, C. Coti, C. Cerin, K. Shiozaki, Y. Xu, P. Delort, J.-P. Smets, J. L. Lous, S. Lubiarz, and P. Leclerc. Downtime statistics of current cloud solutions. Technical report, International Working Group on Cloud Computing Resiliency, 2013.

[49] R. German. *Performance Analysis of Communication Systems with Non-Markovian Stochastic Petri Nets*. John Wiley & Sons, Inc., New York, NY, USA, 2000.

[50] R. P. Goldberg. *Architectural Principles for Virtual Computer Systems*. PhD thesis, Harvard University - Cambridge, MA US, 1973.

[51] A. Gonzalez, P. Gronsund, K. Mahmood, B. Helvik, P. Heegaard, and G. Nencioni. Service availability in the nfv virtualized evolved packet core. In *2015 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6, Dec 2015.

[52] E. Guedes, L. Silva, and P. Maciel. Performability analysis of i/o bound application on container-based server virtualization cluster. In *Computers and Communication (ISCC), 2014 IEEE Symposium on*, pages 1–7, June 2014.

[53] E. A. C. Guedes. Performability Analysis of Web Cache Server Clusters Applied to Server Virtualization. Master's thesis, Federal University of Pernambuco, Brazil, 2015. [Online]. Available: http://www.modcs.org/wp-content/uploads/thesis/Dissertation-Erico.pdf.

[54] J. Halpern and C. Pignataro. Service function chaining (sfc) architecture. RFC 7665, RFC Editor, October 2015.

[55] D. M. Hamby. A review of techniques for parameter sensitivity analysis of environmental models. *Environmental Monitoring and Assessment*, 32(2):135–154, 1994.

[56] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee. Network function virtualization: Challenges and opportunities for innovations. *IEEE Communications Magazine*, 53(2):90–97, 2015.

[57] D. Heimann, N. Mittal, and K. Trivedi. Dependability modeling for computer systems. pages 120 – 128, 03 1991.

[58] S. Herker, X. An, W. Kiess, S. Beker, and A. Kirstaedter. Data-center architecture impacts on virtualized network functions service chain embedding with high availability requirements. In *2015 IEEE Globecom Workshops (GC Wkshps)*, pages 1–7, Dec 2015.

[59] Y. Huang, C. Kintala, N. Kolettis, and N. Fulton. Software rejuvenation: analysis, module and applications. *Twenty-Fifth International Symposium on Fault-Tolerant Computing. Digest of Papers*, pages 381–390, 1995.

[60] D. Inc. Get started, part 1: Orientation and setup. https://docs.docker.com/get-started/, 2019.

[61] V. Inc. Vmware vsphere. `https://www.vmware.com/products/vsphere.html`, 2017.

[62] R. Jain and S. Paul. Network virtualization and software defined networking for cloud computing: a survey. *IEEE Communications Magazine*, 51(11):24–31, November 2013.

[63] W. Keesee. A method of determining a confidence interval for availability. 1965.

[64] D. S. Kim, F. Machida, and K. Trivedi. Availability modeling and analysis of a virtualized system. In *Dependable Computing, 2009. PRDC '09. 15th IEEE Pacific Rim International Symposium on*, pages 365–371, Nov 2009.

[65] H. Kim and N. Feamster. Improving network management with software defined networking. *IEEE Communications Magazine*, 51(2):114–119, February 2013.

[66] D. Kusnetzky. *Virtualization: A Manager's Guide*. O'Reilly Media, 2011.

[67] KVM. Kvm - kernel-based virtual machine. `http://www.linux-kvm.org/page/Main_Page`, 2017.

[68] C. Labs. Pacemaker cluster resource manager. `https://www.clusterlabs.org/pacemaker.html`, 2017.

[69] J.-C. Laprie. Dependability Computing And fault tolerance : concepts and terminology. III:2–11, 1996.

[70] Y. Liu and K. S. Trivedi. A general framework for network survivability quantification. pages 369–378, 2004.

[71] C. J. Lu and W. Q. Meeker. Using degradation measures to estimate a time-to-failure distribution. *Technometrics*, 1993.

[72] W. Luis, C. Jonatas, and A. Marques. Data Plane Programmability Beyond OpenFlow : Opportunities and Challenges for Network and Service Operations and Management. *Journal of Network and Systems Management*, 2017.

[73] LXC. Lxc - linux containers. `https://linuxcontainers.org/`, 2017.

[74] F. Machida, D. S. Kim, and K. S. Trivedi. Modeling and Analysis of Software Rejuvenation in Server Virtualized System. pages 5–10, 2011.

[75] P. R. M. Maciel, R. Matias, J. Dong, and S. Kim. *Dependability Modeling*. Number 3. 2012.

[76] M. Mahalingam, D. Dutt, K. Duda, P. Agarwal, L. Kreeger, T. Sridhar, M. Bursell, and C. Wright. Virtual extensible local area network (vxlan): A framework for overlaying virtualized layer 2 networks over layer 3 networks. RFC 7348, RFC Editor, August 2014. `https://www.rfc-editor.org/rfc/rfc7348.txt`.

[77] M. Malhotra and K. S. Trivedi. Dependability modeling using petri-nets. *IEEE Transactions on Reliability*, 44(3):428–440, Sep 1995.

[78] H. Masutani, Y. Nakajima, T. Kinoshita, T. Hibi, H. Takahashi, K. Obana, K. Shimano, and M. Fukui. Requirements and design of flexible nfv network infrastructure node leveraging sdn/openflow. In *Optical Network Design and Modeling, 2014 International Conference on*, pages 258–263, May 2014.

[79] R. Matos, J. Araujo, P. Maciel, F. Vieira, R. Matias, and K. S. Trivedi. Software rejuvenation in eucalyptus cloud computing infrastructure: A method based on time series forecasting and multiple thresholds. *Proceedings - 2011 3rd International Workshop on Software Aging and Rejuvenation, WoSAR 2011*, 7(3):38–43, 2011.

[80] R. Matos, J. Dantas, J. Araujo, K. S. Trivedi, and P. Maciel. Redundant eucalyptus private clouds: Availability modeling and sensitivity analysis. *Journal of Grid Computing*, 15(1):1–22, 2017.

[81] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. Openflow: Enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74, Mar. 2008.

[82] P. M. Mell and T. Grance. Sp 800-145. the nist definition of cloud computing. Technical report, Gaithersburg, MD, United States, 2011.

[83] C. Melo, J. Araujo, V. Alves, and P. Maciel. Investigation of software aging effects on the OpenStack cloud computing platform. *Journal of Software*, 12(2):125–138, 2017.

[84] C. Melo, R. Matos, J. Dantas, and P. Maciel. Capacity-oriented availability model for resources estimation on private cloud infrastructure. In *2017 IEEE 22nd Pacific Rim International Symposium on Dependable Computing (PRDC)*, pages 255–260, Jan 2017.

[85] M. Melo, J. Araujo, R. Matos, J. Menezes, and P. Maciel. Comparative analysis of migration-based rejuvenation schedules on cloud availability. *Proceedings - 2013 IEEE International Conference on Systems, Man, and Cybernetics, SMC 2013*, pages 4110–4115, 2013.

[86] Microsoft. Windows hyper-v server. `https://docs.microsoft.com/en-us/windows-server/virtualization/hyper-v/hyper-v-server-2016`, 2017.

[87] D. C. Montgomery and G. C. Runger. *Applied Statistics and Probability for Engineers*. John Wiley and Sons, 2003.

[88] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, Apr 1989.

[89] N. Naksinehaboon, N. Taerat, C. Leangsuksun, C. F. Chandler, and S. L. Scott. Benefits of software rejuvenation on hpc systems. In *International Symposium on Parallel and Distributed Processing with Applications*, pages 499–506, Sep. 2010.

[90] S. Nandwani. Managing kubernetes on openstack at scale. `https://www.youtube.com/watch?v=3OuyLlPrpy0`, May 2017.

[91] T. A. Nguyen, D. S. Kim, and J. S. Park. A Comprehensive Availability Modeling and Analysis of a Virtualized Servers System Using Stochastic Reward Nets. *Scientific World Journal*, 2014, 2014.

[92] T. A. Nguyen, D. Min, and E. Choi. A comprehensive evaluation of availability and operational cost for a virtualized server system using stochastic reward nets. *The Journal of Supercomputing*, Aug 2017.

[93] T. A. Nguyen, D. Min, and J. S. Park. A comprehensive sensitivity analysis of a data center network with server virtualization for business continuity. *Mathematical Problems in Engineering*, 2015.

[94] F. M. D. Nitto, J. Friesse, and S. Dake. Corosync cluster engine. http://corosync.github.io/corosync/, 2017.

[95] A. S. Oliveira. SIMF: Um Framework de Injeção e Monitoração de Falhas de Nuvens Computacionais Utilizando SPN, 2017. Dissertação de Mestrado, UFPE (Universidade Federal de Pernambuco, Recife, Brazil.

[96] ONF. Software-defined networking: The new norm for networks. Technical report, Open Networking Foundation, April 2012.

[97] OpenNebula. About opennebula. https://opennebula.org/about, 2017.

[98] OpenStack. What is openstack? https://www.openstack.org/software/, 2017.

[99] OpenVZ. Openvz linux containers. http://openvz.org, 2017.

[100] Oracle. Oracle virtual box. https://www.virtualbox.org, 2017.

[101] L. Perkov, N. Pavkovic, and J. Petrovic. High-availability using open source software. In *2011 Proceedings of the 34th International Convention MIPRO*, pages 167–170, May 2011.

[102] W. Polygraph. Web polygraph: a benchmarking tool for caching proxies and other web intermediaries., 2016.

[103] Qemu. Qemu - quick emulator. http://www.qemu.org, 2017.

[104] P. Quinn and T. Nadeau. Problem statement for service function chaining. RFC 7498, RFC Editor, April 2015. http://www.rfc-editor.org/rfc/rfc7498.txt.

[105] J. Riley, J. Noss, W. Dillingham, J. Cuff, and I. M. Llorente. A high-availability cloud for research computing. *Computer*, 50(6):92–95, 2017.

[106] A. Rugina, K. Kanoun, and M. Kaâniche. An architecture-based dependability modeling framework using AADL. *CoRR*, abs/0704.0865, 2007.

[107] R. Russell. Netfilter: firewalling, nat, and packet mangling for linux, 2018. http://netfilter.org/.

[108] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar. Flowvisor: A network virtualization layer. *OpenFlow Switch Consortium, Tech. Rep*, 1:132, 2009.

[109] B. Silva, R. Matos, G. Callou, J. Figueiredo, D. Oliveira, J. Ferreira, J. Dantas, A. L. Junior, V. Alves, and P. Maciel. Mercury : An Integrated Environment for Performance and Dependability Evaluation of General Systems.

[110] P. S. Solutions. Proxmox virtual environment. https://www.proxmox.com/en/proxmox-ve, 2017.

[111] C. Strachey. Time sharing in large, fast computers. In *IFIP Congress*, pages 336–341, 1959.

[112] D. Sun, G. Chang, Q. Guo, C. Wang, and X. Wang. A dependability model to enhance security of cloud environment using system-level virtualization techniques. In *2010 First International Conference on Pervasive Computing, Signal Processing and Applications*, pages 305–310, Sep. 2010.

[113] A. T. Tai, L. Alkalai, and S. N. Chau. On-board preventive maintenance: A design-oriented analytic study for long-life applications. *Performance Evaluation*, 35(3):215–232, 1999.

[114] W. Tarreau. Ha proxy tcp/http load balancer. http://www.haproxy.org/, 2017.

[115] Telcordia. Generic requirements for operations systems platform reliability. Technical report, Telcordia Technologies, June 1994.

[116] M. Torquato and I. M. U. P. Maciel. Models for availability and power consumption evaluation of a private cloud with VMM rejuvenation enabled by VM Live Migration. *Journal of Supercomputing*, 74(9):4817–4841, 2018.

[117] K. S. Trivedi. *Probability and Statistics with Reliability, Queuing and Computer Science Applications*. John Wiley and Sons Ltd., Chichester, UK, 2nd edition edition, 2002.

[118] K. S. Trivedi. What is sharpe? https://sharpe.pratt.duke.edu/, 2019.

[119] K. S. Trivedi and M. Malhotra. *Reliability and Performability Techniques and Tools: A Survey*, pages 27–48. Springer Berlin Heidelberg, Berlin, Heidelberg, 1993.

[120] A. A. Ucla, A. Avizienis, J. claude Laprie, and B. Randell. Fundamental concepts of dependability, 2001.

[121] J. F. Vilas, J. P. Arias, and A. F. Vilas. *High Availability with Clusters of Web Services*, pages 644–653. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[122] VMWare. Vmware virtualization software for desktops, servers and virtual machines for public and private cloud solutions. http://www.vmware.com, 2017.

[123] W. von Hagen. *Professional Xen Virtualization*. Wrox Press Ltd., 2008.

[124] S. v. Vugt. *Pro Linux High Availability Clustering*. Apress, Berkely, CA, USA, 1st edition, 2014.

[125] B. Wei, C. Lin, and X. Kong. Dependability modeling and analysis for the virtual clusters. In *Proceedings of 2011 International Conference on Computer Science and Network Technology*, volume 4, pages 2316–2320, Dec 2011.

[126] T. S. J. H. William Gropp, Ewing Lusk. *Beowulf cluster computing with Linux*. Scientific and Engineering Computation. The MIT Press, 2 edition, 2003.

[127] M. Williams. Linux ethernet bonding driver howto, April 2011.

[128] Wolfram Research Inc. *Mathematica 10.0*, 2015.

[129] Xen. Xen hypervisor. `http://xen.org`, 2017.

[130] W. Xie, Y. Hong, and K. Trivedi. Analysis of a two-level software rejuvenation policy. *Reliability Engineering and System Safety*, 87(1):13 – 22, 2005.

[131] S. Yoon, T. Na, H. Yoon, and H.-Y. Ryu. An implementation of network computing testbed system on open source virtualized cloud. In *2017 IEEE International Conference on Consumer Electronics (ICCE)*, pages 156–157, Jan 2017.

[132] C. Zhang, L. Fourie, P. Carver, V. Choudhary, R. Fei, X. Wang, R. P. S. Wong, I. Cardoso, A. Motoki, S. Vasudevan, A. Migliaccio, and K. Mestery. Service function chaining extension for openstack networking. `https://docs.openstack.org/networking-sfc/latest/index.html`, 2017.

[133] H. Ziade, R. Ayoubi, and R. Velazco. A survey on fault injection techniques. *International Arab Journal of Information Technology*, Vol. 1, No. 2, July:171–186, 2004.

Appendix

A

HA OpenStack Implementation

Table A.1 presents the status of HA deployment of adopted OpenStack components regarding installation, the necessity of RA customization, and whether the component is already managed by Pacemaker.

Table A.1: OpenStack Components Status

	customized RA	**HA by Pacemaker**
apache		
keystone	NO	YES
horizon	NO	YES
glance		
glance-api	NO	YES
glance-registry	NO	YES
nova		
nova-api	YES	YES
nova-conductor	NO	YES
nova-scheduler	NO	YES
nova-novncproxy	NO	YES
nova-consoleauth	NO	YES
nova-compute	NO	YES
neutron		
neutron-server	YES	YES
neutron-agent-dhcp	NO	YES
neutron-agent-l3	NO	YES

Table 2 lists all additional OpenStack required softwares for deployment of HA Solution.

Table A.2: OpenStack: Additional Required Softwares

	installed	**available RA**	**HA Solution**
MariaDB	YES	NO	Galera
Galera	YES	YES	Pacemaker
HAProxy	YES	YES	Pacemaker
RabbitMQ	YES	YES	native
memcached	YES	NO	none

B

Specification of TISVEP Extension Messages

The Tuning Infrastructure for Server Virtualization Experiment Protocol (TISVEP) [53] messages, that extend the original protocol, are presented below. Each transmitted message has the MSG_ID parameter in the first field. Each message is easily identified by a bold Message Code and name.

312: createChain: the processing of this message results in the execution of openstack SFC API commands that create the service function chain.
Example:

```
MSG_ID=312:SRC_IP=192.168.99.10:DST_IP=192.168.99.16
```

In the above example, the VM source IP 192.168.99.10, in which the Web-Polygraph client-side will be executed, and the VM destination IP 192.168.99.16, in which the Web-Polygraph server-side will be executed, are stated. They are the source and the destination of the service function chain traffic. The result is the creation of the Open vSwitch rules that will forward the traffic throughout the chain. The message is transmitted to the controller cluster IP. Any node belonging to the cluster is able to process the chain creation.

313: deleteChain: the processing of this message results in the execution of openstack SFC API commands that delete the service function chain. This results in the deletion of the Open vSwitch rules that were forwarding the traffic throughout the chain.
Example:

```
MSG_ID=313
```

In the above example, the message identification is transmitted for the controller cluster IP. Any node belonging to the cluster is able to process the chain deletion.

314: detectMigration: the processing of this message is performed by the compute node

receiving the chain migration. The VMs migration is completed when their processes are created in the receiving computer node. We monitor the Process Identification (PID) created by the hypervisor for each VM. When all the PIDs were created, the TISVEP informs the experimenter node, and the chain creation can continue.
Example:

```
MSG_ID=314
```

In the above example, the message identification is transmitted for the compute node receiving the chain migration and the VMs' PIDs monitoring can start.

315: fwRulesInjection: insert missing rules in linux netfilter, that should be inserted by openstack SFC API does.
Example:

```
MSG_ID=315:SRC_IP=192.168.99.10
```

In the above example, the compute node receiving the message will add rules to the linux netfilter firewall, allowing the traffic from Web-Polygraph server-side machine (192.168.99.10) to be forward in the chain.

316: migrateChain: the processing of this message results in the initiation of SFC migration process. The message is transmitted to the controller cluster IP. Any node belonging to this cluster is able to process the chain creation.
Example:

```
MSG_ID=316:DST_CMP_HN=192.168.99.62
```

In the above example, the 316 message ID and the 192.168.99.62 (the IP address of ha-compute02 openstack node) was passed to the controller cluster IP. Any node belonging to the controller cluster is able to process the chain migration.