On the Resilience of the NFV-MANO: An Availability Model of a Cloud-native Architecture

Besmir Tola, Yuming Jiang, and Bjarne E. Helvik
Department of Information Security and Communication Technology
NTNU-Norwegian University of Science and Technology
Trondheim, Norway
Email: besmir.tola@ntnu.no, yuming.jiang@ntnu.no, bjarne@ntnu.no

***Abstract*—With Network Function Virtualization (NFV), the management and orchestration of network services require a new set of functionalities to be added on top of legacy models of operation. Due to the introduction of the virtualization layer and the decoupling of the network functions and their running infrastructure, the operation models need to include new elements like virtual network functions (VNFs) and a new set of relationships between them and the NFV Infrastructure (NFVI). The NFV Management and Orchestration (MANO) framework plays the key role in managing and orchestrating the NFV infrastructure, network services and the associated VNFs. Failures of the MANO hinders the network ability to react to new service requests or events related to the normal lifecycle operation of network services. Thus, it becomes extremely important to ensure a high level of availability for the MANO architecture. The goal of this work is to model, analyze, and evaluate the impact that different failure modes have on the MANO availability. A model based on Stochastic Activity Networks (SANs), derived from current standard-compliant microservice-based implementations, is proposed as a case study. The case study is used to quantitatively evaluate the steady-state availability and identify the most important parameters influencing the system availability for different deployment configurations.**

***Index Terms*—NFV-MANO, OSM, Availability, SAN models, Docker.**

I. Introduction

Network Functions Virtualization (NFV) is expected to bring significant changes in today's network architectures. By decoupling the network function software from the underlying hardware infrastructure, hence, allowing the software to run on commodity hardware, it provides the necessary flexibility to enable agile, cost-efficient, and on-demand service delivery combined with automated management.

The European Telecommunications Standards Institute (ETSI) defines the NFV-Management and Orchestration (NFV-MANO) framework [1], in the following referred to as simply MANO, as a set of three main functional blocks: the NFV Orchestrator (NFVO), the VNF Manager (VNFM), and the Virtualized Infrastructure Manager (VIM). The NFVO orchestrates all the functionality on the service level including operations like on-board, instantiate, scale, or terminate network services. The VNFM is responsible for the lifecycle management (e.g. instantiation, scaling, and healing) of one or more virtual network function (VNF) instances. It receives management (e.g. deploy, scale, and terminate) instructions for VNFs from the NFVO, which it executes through its interface with the VNFs. The third major component, the VIM, manages the physical infrastructure (NFVI) where the VNFs are executed.

Operating-system-level virtualization technologies, commonly referred to as containers (e.g., Docker [2] or LXD [3]) have enabled a shift in the way applications are deployed going from a monolithic to a microservice-based, i.e., cloud-native, architecture. The later empowers the development, deployment, and operation of large and complex applications as a set of independent smaller and lighter components (i.e., microservices) where each component provides a specific service, and communicates through well-defined lightweight mechanisms. This way, service provisioning becomes more flexible, agile, and reliable [4]. Driven by such benefits, several open source MANO projects leverage a micro-service architecture in deploying and operating MANO components through lightweight containers [5]–[7].

Network operators demand that some of their NFV-based services ensure a carrier grade quality of service [8], i.e. highly reliable and trustworthy. However, service outages, induced by various component failures, are inevitable events that service operators need to deal with. To this end, an automated management and orchestration system embracing resiliency aspects is mandatory for conducting correct counter-actions to such events. Failures on the management and orchestration level could jeopardize the functionality of all the network and potentially impact the service delivery by inducing severe outages, which sometimes might be hard to deal with [9], [10]. It is thus of an utmost importance to ensure that a logically centralized control and orchestration system is highly dependable and able to ensure *service continuity* [11]. To this end, ETSI has streamlined several guidelines and requirements of the management and orchestration resiliency capabilities [12].

The objective of this paper is to model and quantitatively analyze MANO steady-state availability when deployed on container-based technologies for various deployment options. To this end, we present an NFV-MANO availability model, derived from current ETSI-compliant architectures, based on Stochastic Activity Networks (SANs) and perform a quantitative availability assessment aiming at finding the factors mostly impacting system availability. In the model, we incorporate various failure modes on both hardware and software level of the MANO framework. A sensitivity analysis helps us identify the most critical components of the system in terms of relative impact on the system steady-state availability. Moreover, we examine several ways of deploying the software stack aiming at providing higher availability, inspired by current MANO implementations adopting cloud-native practices.

The paper structure is organized as follows. Section II presents the related work and highlights the key contributions. The MANO architecture used to provide the basis of the model is illustrated in Section III. The case study availability model is presented in Section IV. In Section V, we show the results of the analysis and conclude the paper by highlighting the most important insights in Section VI.

II. Related Work

Even though the MANO may have a huge impact on the NFV-enabled network service performance [9], [10], a study of its failure dynamics and overall availability analysis is still missing in the literature. Almost all related work focus on network service availability modeling and quantification. They either focus on specific use cases like virtualized-EPC [13] and virtualized-IMS [14], or model and analyze generic network services provided through NFV-based networks [15], [16] without considering the impact of the MANO on the overall service performance. By aggregating non-state space (Reliability Block diagrams) and state-space models (Stochastic Reward Nets) they quantify and give insights on the service availability and propose appropriate redundancy configurations aiming at providing 5-nines availability.

In a more recent study [17], a composed availability model of an NFV service, based on SANs, is proposed. Each VNF, composing the network service, is considered as a load-sharing cluster and the authors propose separate models for various redundancy mechanisms called Availability Modes and investigate the impact that a faulty orchestrator has on the service availability. Differently, in this paper we propose availability models derived from current micro-service based implementations, i.e., containerized, and provide insights on the most critical parameters affecting the availability for different deployment options.

Availability models of containerized systems for different configurations have been proposed in [18]. The authors propose and compare various container deployments and through both analytic and simulation results they investigate k-out-of-N availability and the system sensitivity to failure parameters. In [19], the same authors present a software tool, called ContAv, for the evaluation of containerized systems' availability. Through the use of both non-state and state-space models designed by the authors, the tool assess the system availability for different configurations and allows a system architect to easily parametrize and perform sensitivity analysis.

In [20], even though not related to availability modeling, the authors propose centralized and distributed mechanisms for a providing a reliable and fault-tolerant microservice-based MANO. The mechanisms exploit load balancing and state sharing and include some tunnable parameters which can help an operator optimise the trade-offs between reliability and the associated costs in terms of resource usage. The proposed setup allows the definition of a cost function which can help the operator determine the best configuration among the centralized and distributed mechanisms.

Compared to the related studies our contribution aims at filling the current gap regarding the availability assessment of a critical element of the NFV architecture. We model and assess a hypothetical MANO system inspired by the current trend of adopting cloud-native software development and maintenance embraced by several architectures. In addition, we investigate various containerized deployment options aiming at achieving high availability levels and identify critical failure parameters impacting the MANO availability.

III. Case Study

An ETSI-compliant MANO should adhere to the specifications streamlined by ETSI and include the main functional blocks which should interact with each other through well-defined reference points and provide an end-to-end network service orchestration. In this paper we extrapolate the deployment options of OSM [5], a well-established architecture hosted by ETSI and led by a large community including both operators and research institutions [5]. OSM is closely aligned with NFV specifications and consists in a production-quality and VIM-independent software stack. Seven releases have been distributed up to now. Release 6 (Release 0 has had a relatively short lifetime) is currently the latest release and includes different installation procedures where the MANO components can be deployed as *dockerized* instances [2] into a hypervisor-based virtualized environment, a public hosting infrastructure, or directly into a proprietary commodity hardware. The latter represents the most common way of deploying and running the OSM stack. It represents the most advanced release including among others network service and slicing capabilities, enhanced user interface, and a lighter orchestrator.

The default installation deploys 13 docker containers running in a Docker *swarm mode* with each component having one single replica. Docker *swarm mode* is a native feature of Docker for managing and orchestrating a cluster of Docker engines called *swarm*. It entails several cluster management characteristics like: i) decentralized configuration of cluster nodes at runtime, ii) automatic scaling, iii) automatic cluster state reconciliation, and iv) integrated load balancing. A swarm is a cluster of Docker nodes, running in a *swarm mode*, and they act as managers, who manage the swarm membership and delegate tasks, and workers which run swarm services.

A Docker node can be a manager, worker, or both. A *swarm* may consist in only one node which by default will act as a manger and worker at the same time, but it cannot be only a worker without a manager. We refer to this as the

Manager configuration. In case the cluster is composed of worker and manager nodes, we refer to it as *Manager-Worker* configuration.

One of the key features of a swam is the automatic cluster state reconciliation. This is very important in terms of fault management policies. In case one of the services of the cluster is down, the swarm state changes and the manager immediately respawns the failed container/containers on other available node and the service stack becomes healthy again.

IV. Availability Model

A SAN is a modeling formalism with which detailed performance, dependability, or performability models can be implemented in a comprehensive manner [21]. SANs are stochastic extentions of Petri Nets consisting of four primitives: *places*, *activities*, *input gates*, and *output gates*. Places are graphically represented as circles and contain a certain number of tokens which represent the *marking* of the place. Activities are actions that take a certain amount of time to fire and move tokens from one place to another. Input and output gates define marking changes that occur when an activity completes. Different from output gates, the input gates are also able to control the enabling of activity completion, i.e., firing.

In the following, we illustrate the proposed models representing the different MANO configurations.

A. Manager Configuration

Past studies classify software faults into two main categories, Bohrbugs and Mangelbugs [22]. Bohrbugs, otherwise called deterministic, are typically easily reproducible since they tend to manifest themselves consistently under the same conditions. They often may lead to a software crash or process hanging and the bugs need to be identified and resolved. Mandelbugs are bugs whose activation and error propagation are complex. As a result, it is quite hard to reproduce and their manifestation is transient in nature. They are usually caused by timing and synchronization issues resulting in race conditions. A retry operation or software restart may resolve the issue. There is a further subtype of Mandelbugs that is related to an aspect know as software aging. Software aging is a well-know issue which characterizes the software failure rate due to phenomena like the increase of software execution period [23]. It has been shown that the increase of process runtime is a common cause to the increase of software failure rate and the system performance may degrade over time. Typical faults in IT systems caused by aging effects include resource leakages, numerical error, or data corruption accumulation. Therefore, the failure might occur as a result of the increase of system uptime. Common methods of recovering from such failures rejuvenation techniques consisting of restart and/or reboots procedures [24].

On the software level, for the scope of our investigation, we differentiate between two types of software failures, non-aging related failures and aging related failures. The former set aims at representing both Bohrbugs and non-aging Mangelbugs where the majority of these failures can be recovered through a manual intervention for software repair and the latter represents the failures due to aging effects where the majority of these failure are recovered by a software restart/reboot.

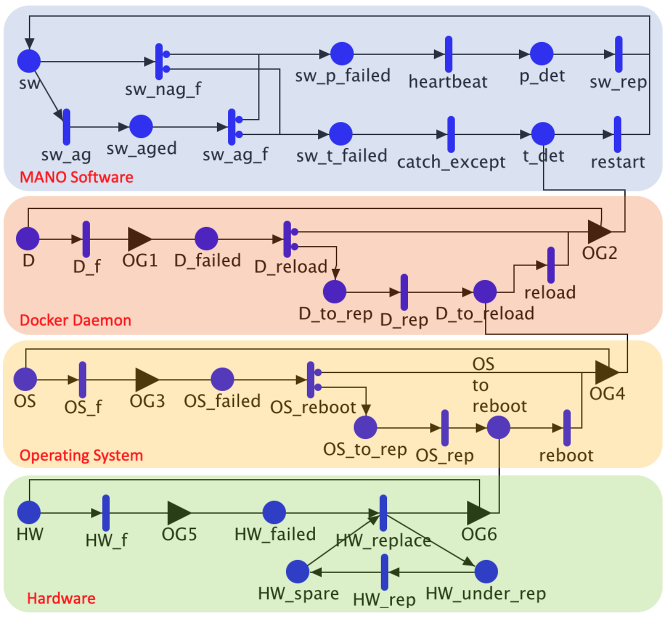

Fig. 1: SAN availability model of the MANO deployed in a Manager configuration.

Fig. 1 illustrates the SAN model of the *Manager* configuration. It consists of the deployment of the MANO containerized software into one physical node which acts as both manger and worker for the service tasks. The model includes the MANO software, Docker daemon, OS, and hardware components and their relative places *sw*, *D*, *OS*, and *HW* are initialized with 1 token each, indicating a fully working system. Similar to previous works (see related work [13]–[16]), it is assumed that all the timed activities follow a negative exponential distribution unless otherwise specified.

The aging effect is represented through a specific timed activity *sw_ag* with rate $\lambda_{\mathrm{sw_{ag}}}$ which defines the time it takes for the software to age, i.e., the average time that the software accumulates errors that might lead to an aging-related failure. The timed activities *sw_ag_f* and *sw_nag_f* represent the aging and non-aging related software failure events with rates $\lambda_{\mathrm{sw-fail_{ag}}}$ and $\lambda_{\mathrm{sw-fail_{nag}}}$, respectively. For both events, we differentiate between two types of software failures based on their recovery process. To this end, we make use of case probabilities associated to the timed activities where C_{nag} defines the probability that a non-aging related failure event is recovered with a software restart. With probability $1-C_{\mathrm{nag}}$ the failure recovery requires a manual intervention for executing a software repair. Similarly, C_{ag} defines the probability that an aging related failure is recovered with a software restart and with $1 - C_{\mathrm{ag}}$ the recovery requires a software repair.

Once a software failure is experienced, a token is placed in either *sw_p_failed* or *sw_t_failed* defining the recovery process the software will undergo. *heartbeat* and *catch-exception* represents the detection of the failures and are defined with deterministic times μ_h and μ_c. *sw_rep* and *restart* represents the repair (including any eventual reboot or upgrade of software) and restart events of the software with rate $\mu_{\mathrm{sw_{rep}}}$ and $\mu_{\mathrm{sw_{res}}}$. On the docker engine level, i.e., daemon, *D_f* and *D_reload* represents the failure and recovery events of the daemon with rates λ_{D} and $\mu_{\mathrm{D_r}}$, respectively. The recovery entails a daemon reload where with probability C_{D} a daemon reload recovers the failure and with $1 - C_{\mathrm{D}}$ a hard repair is needed. The later is defined through the activity *D_rep* with rate $\mu_{\mathrm{D_{rep}}}$. Once the daemon is repaired, an additional reload is performed to fully recover it. Similarly to the daemon, the operating system level is modeled with the same dynamics having specific failure and repair parameters which we introduce in Section V. On the hardware level, *HW_f* and *HW_replace* represents the failure and recovery events of the hardware with rates λ_{HW} and $\mu_{\mathrm{HW_{rep}}}$, respectively. The place *HW_spare* indicates the spare hardware equipment used to replace the failed hardware and is initialized with 1 token.

Finally, the following output gates define the token marking movements among places: *OG1/OG3/OG5* manage the failure events of the daemon, OS, and hardware levels, respectively. When their related timed activities fire, connected to their incoming arcs, the output gate places 1 token in the respective failed position and sets to zero the upper-level places. This is because a failure of the physical hardware will cause a failure of the OS which in turn impacts the operational state of the daemon and MANO software as well; *OG2/OG4/OG6* places 1 token in their relative working place, i.e., *D/OS/HW*, and the relative upper-level places to which they are connected by outgoing arcs. For example, a recovery from a daemon failure brings the daemon in the up state but requires a restart of the MANO software for a fully working MANO.

B. Manager-Worker Configuration

The *Manager-Worker* swarm configuration consists of two separate nodes forming a cluster where the OSM stack is deployed on the worker node and the Manager node performs the control and scheduling of tasks. Fig. 2 depicts the *Manager-Worker* SAN model. To distinguish the models of the two entities, we make use of a suffix *_M* for all the places and activities regarding the manager part. The system is fully working if there is a token in either of the *sw*, *sw_aged*, or *sw_M* places.

On the worker node, the MANO software component is similar to the *Manager* configuration except for the recovery phase where once a failure is detected, the containers running the software are respawned, through the timed activities *respawn* or *respawn1*, in the manager node. We distinguish two cases: when a software repair is needed, the token is moved from *p_det* of the worker node to *p_det_M* of the manager. In the other case, the token is moved from *t_det* to *sw_M* indicating that a respawn, i.e., container restart, is sufficient to recover the system. However, for both cases, we consider the eventuality of a respawn process that fails. To this end, we consider two case probabilities associated with the timed activities. With probability C_{respawn}, the container respawn

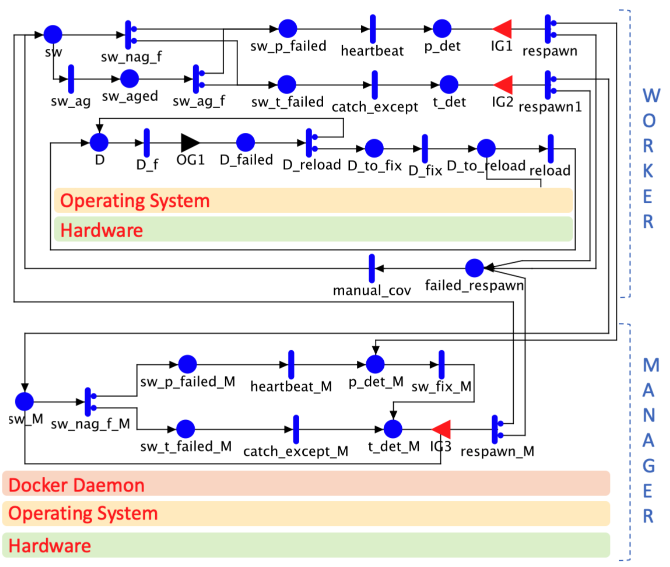

Fig. 2: SAN availability model of the MANO deployed in a Worker-Manager configuration.

is successful and $1 - C_{\mathrm{respawn}}$ it fails. In the latter, there is a need for a manual coverage, represented by *manual_cov*, and the token is placed back in place *sw*. In order for the respawn to instantiate, the hosting manager node needs to be operational and this is controlled by the enabling gates *IG1/IG2* which enable the respawn only if the daemon, OS and hardware of the manager are working, i.e., their respective places *D_M*, *OS_M*, and *HW_M* contain each 1 token. In addition, differently to the *Manager* setup, once the daemon fails, there is just the recovery of the daemon since the MANO software is immediately respawned in the manager node. The rest of the model is similar to the *Manager* configuration, hence we omit further illustrating.

On the manager node, once a token is deposited in *sw_M*, the system is again operational. While the software is running in this node, we assume that it is subject to only non-aging software failures. This is because *swarm mode* best practices suggest that the worker node should be the dedicated node for handling task requests in a 'normal' condition. Therefore, we limit the hosting of the MANO software to the manager node only for the period the worker node is failed. To this end, the input gate *IG3* enabled a respawn of the software containers from the manager node to the worker node once the worker node is up and running again and ready to accommodate the containers. As a result, the manager node will host the containers for a relatively short time compared to the software aging time, hence making the assumption of only non-aging failure events while the software is running on the manager node a reasonable assumption. The rest of the manager components, i.e., daemon, OS, and hardware are similar to the *Manager* configuration.

C. Replicated Configuration

One of the most advantageous *swarm* features is automatic scaling and integrated load balancing. In case MANO utilization gets close to its resource limits, an operator can easily

TABLE I: Availability model parameters.

Intensity	Time	Description [Mean time to]
$\lambda_{\mathrm{sw}_{\mathrm{ag}}}^{-1} = 1$	week	MANO software aging
$\lambda_{\mathrm{sw-fail}_{\mathrm{ag}}}^{-1} = 3$	days	next MANO software failure after aging
$\lambda_{\mathrm{sw-fail}_{\mathrm{nag}}}^{-1} = 2$	month	next MANO non-aging software failure
$\mu_{\mathrm{sw}_{\mathrm{rep}}}^{-1} = 1$	hour	MANO software repair
$\mu_{\mathrm{sw}_{\mathrm{res}}}^{-1} = 30$	seconds	MANO software restart
$\mu_{h}^{-1} = 10$	seconds	heartbeat*
$\mu_{c}^{-10} = 1$	millisecond	catch exception*
$\lambda_{\mathrm{D}}^{-1} = 4$	months	next daemon failure
$\mu_{\mathrm{D}_{\mathrm{rep}}}^{-1} = 1$	hour	daemon repair
$\mu_{\mathrm{D}_{\mathrm{r}}}^{-1} = 15$	seconds	daemon reload
$\lambda_{\mathrm{OS}}^{-1} = 4$	months	next OS failure
$\mu_{\mathrm{OS}_{\mathrm{rep}}}^{-1} = 1$	hour	OS repair
$\mu_{\mathrm{OS}_{\mathrm{r}}}^{-1} = 5$	minutes	OS reboot
$\lambda_{\mathrm{HW}}^{-1} = 6$	months	next hardware failure
$\mu_{\mathrm{HW}_{\mathrm{rep}}}^{-1} = 4$	hours	hardware repair
$\mu_{\mathrm{HW}_{\mathrm{replace}}}^{-1} = 1$	hour	hardware replace
$\mu_{\mathrm{respawn}}^{-1} = 1$	minute	respawn MANO software containers
$C_{\mathrm{nag}} = 0.3$		prob. for non-aging transient failures
$C_{\mathrm{ag}} = 0.7$		prob. for aging transient failures
$C_{\mathrm{D}} = 0.9$		daemon reload coverage factor
$C_{\mathrm{OS}} = 0.9$		OS reboot coverage factor
$C_{\mathrm{respawn}} = 0.9$		respawn coverage factor
$N_{\mathrm{spare}} = 1$		Number of spare hardware

*Deterministic time

spin up additional replicas of the containers and the swarm integrated load balancer will manage the task scheduling without any additional configuration required from the operator. Spinning up additional replicas can bring advantages both in terms of performance and availability. To this end, we consider the case where multiple MANO instances are running in both *Manager* and *Manager-Worker* setups and the system is considered availability if at least one replica is working.

For modeling the replicated configuration, it is sufficient setting the number of tokens in the *sw* place equal to the number of replicas for both *Manager* and *Manager-Worker* configurations. This way, the models resemble a setup where multiple containers, for each of the MANO components, are launched and run in the same OS and physical hardware.

V. Numerical Analysis

In this section we present the sensitivity analysis of the steady-state availability for both configurations and failure impact on the overall unavailability. The presented models are defined in the Möbius software tool [25] and the numerical analysis is performed using discrete-event simulations, integrated in the tool, with 95% confidence interval and 0.05 width of relative confidence interval.

A. *Manager configuration: Sensitivity Analysis*

We performed a sensitivity analysis to determine which of the parameters have the highest impact on the steady-state availability of the Manager configuration. The SAN model parameters are retrieved from previous literature [17]–[19] and are illustrated in Table I. They represent the reference values and given these parameters, the achieved MANO availability is presented in Table II, together with its component availabilities where the latter are derived from individual dynamics, i.e., not influenced by underlying component failures. It can be seen that the software part is the most fragile component. For computing a sensitivity analysis, the parameters regarding failure and recovery events were increased and reduced by one order of magnitude, i.e., $\times 10$ and $\times 10^{-1}$, from their reference values. The availability sensitivity to these parameters, separated into failure and recovery events, is presented in Fig. 3.

TABLE II: Steady-state availability of Manager Configuration.

	MANO	MANO Sw	Daemon	OS	Hardware
Availability	0.99751	0.99787	0.99964	0.99967	0.99975

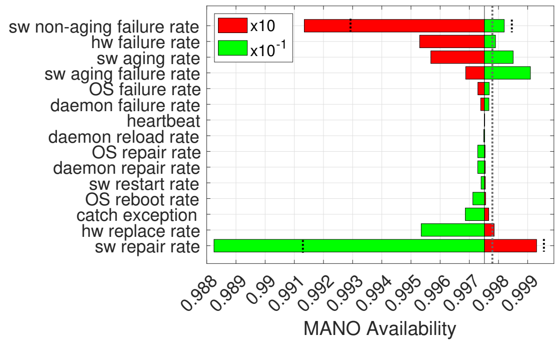

Fig. 3: Sensitivity analysis for the Manager configuration.

From a failure events perspective, the most important parameter is software non-aging failure rate followed by hardware, software aging, and software aging failure rates. Among these, it is the latter that brings the highest improvement on the steady-state availability when there is an order of magnitude reduction of the relative intensities. On the contrary, for the same level of parameter reduction, software repair rate has the highest impact by reducing the system availability from 0.997 to almost 0.988 followed by the hardware replace rate. At the same time, the highest improvement is achieved for a software repair rate increase reaching 0.9993.

Beside the failure and repair parameters, the probability factors that define the types of software failures which may have an important influence on the overall availability. The choice of the reference values is driven by common assumptions that non-aging failures, i.e., deterministic failures, often lead to system crashes and debugging processes can improve software robustness by identifying and resolving the bug. Hence, choosing a $C_{\mathrm{NAG}} = 0.3$ means that the majority of such failures require a software repair. The opposite is valid for aging-related failures which more often may lead to *transient* failures that can be resolved by simply restarting the software. To this end, we explore a range of these factors and their impact on the availability.

Fig. 4 presents the MANO availability for the different combinations. We notice that for the same level of reduction, the aging factor achieves a much higher availability improvement compared to the non-aging factor. This indicates that the system can benefit more from the transient nature of aging related bugs than those of deterministic failures, otherwise called Bohrbugs.

1) Failure Impact: It is expected that on average around 52 failures per year will contribute to a total duration of

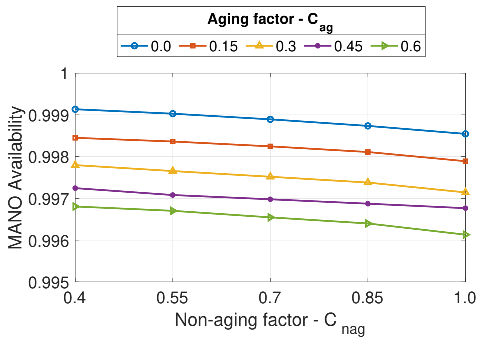

Fig. 4: Impact of aging and non-aging factors on the availability.

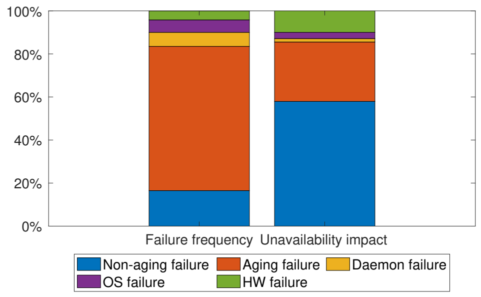

Fig. 5: Failure frequency and relative impact on the unavailability.

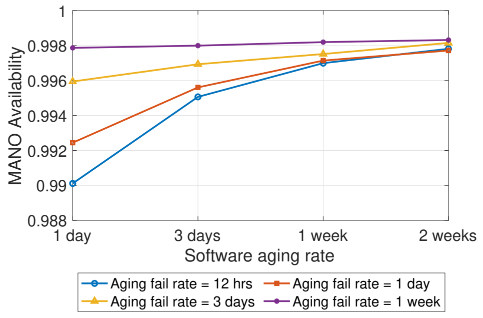

Fig. 6: Impact of software aging and its failure rate on the availability.

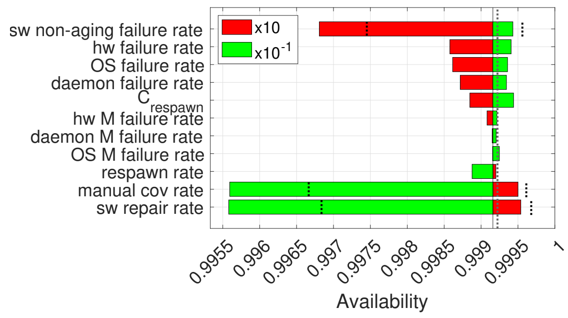

Fig. 7: Sensitivity analysis for the Manager-Worker configuration.

21.7 hours of unavailability. The contribution of different failure types, in terms of their frequency and the impact on the system downtime, is presented in Fig. 5. It can be observed that software failures are the predominant events, accounting for almost 84% of all the failures. In particular, we notice that despite software aging failures represent almost 60% of the overall failures, they lead to only 27.5% of the MANO downtime. On the other hand, non-aging software related failures consists of the 16% of the total failure but contribute to almost 58% of the total downtime. In addition, hardware failures represent around 4% of all the failures and they contribute to more than 10% of the system downtime.

2) Software aging impact: In the sensitivity analysis we noticed that aging failure rate may have a considerable impact on the availability of the MANO. However, software aging rate and aging failure rate are very unpredictable parameters since they depend on several factor that may be out of developer's control such as software utilization rate, i.e., system load, operating infrastructure and software implementation. We explore a wide range of software aging parameters, varying the aging rate between 1 day and 2 weeks and the aging failure rate between 12 hours and 1 week. The MANO availability for different combinations of the parameters is presented in Fig. 6. It can be seen that the impact of the aging failure rates depends greatly on the rate of aging. When the time it takes the software to age is short, i.e, lower than 3 days, the aging failures have a much higher impact on the availability.

B. Manager-Worker Configuration: Sensitivity Analysis

The deployment of the MANO stack in a *Manager-Worker* configuration entails an automatic respawn of the containers in the manager node in case the MANO experiences a failure. This setup is suitable for recovering system outages due to external components like the daemon engine, OS, or hardware failures of the hosting node, i.e., worker node. However, such procedure is triggered only if the manager is capable, i.e., fully working, of hosting the containers running the software components. Therefore, the respawn procedure is constrained by the operational state of the manager node. Fig. 7 depicts the sensitivity analysis of some of the critical parameters in this setup. First, we notice that applying this configuration, i.e, by joining another node into the swarm, the MANO availability is increased from 0.99751 of the *Manager* case to 0.99916 for the reference parameters (refer to Table I). Second, we observe that software repair and manual coverage rate, i.e., the mean time to manually recover a failed respawn procedure, may have a tremendous impact in deteriorating the system availability where for the latter despite the successful respawn factor is set to 0.9. Moreover, we evidence that the external failures on the worker node (hardware, OS, and daemon) have a much higher impact than those of the manager node parameters. This is due to the policy that a respawn of the containers from the manger to worker is done as soon as the worker node is ready to re-host the containers, i.e, it is recovered from failures which triggered a respawn to the manager in the first place.

C. Replicated Configuration

Fig. 8 illustrates the gains in terms of system availability due to the introduction of additional replicas. For both cases, there is a relatively restricted gain which is due to the constraints posed by external components availabilities since all the replicas are still running on the same physical node. However, engaging multiple replicas brings additional benefits in terms of the reduction of the impact of critical parameters as highlighted by the dotted lines in both Fig. 3

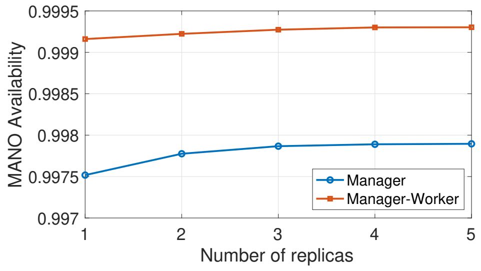

Fig. 8: Availability for different container replicas.

and Fig. 7, representing the cases with 4 replicas for the most impactful parameters in the *Manager* and *Manager-Worker* configurations. We notice that for both configurations, the largest reduction is achieved on software repair rate followed by the non-aging failure rates when the related reference parameters are degraded by one order of magnitude. In addition, a significant reduction is achieved for the impact the manual coverage rate has on the *Manager-Worker* setup.

One solution to the limited gain when multiple replicas are applied can be Docker Universal Control Plane (UPC) for enterprises which envisions a complex architecture that maximally leverages docker swarm scalability for achieving high availability [26]. It consists of a docker swarm with multiple manager and worker nodes instantiated into separated physical nodes. This solution promises much higher availability levels than those anticipated in our study and could represent a viable solution for network operators deploying and managing a cloud-native MANO. We leave the investigation of this solution for future work.

VI. Conclusion

In this paper, we present an availability model for a cloud-native NFV-MANO architecture from which we analyze and quantify its steady-state availability. We have included the most typical failure modes and evaluated their impact through sensitivity analysis for different containerized deployments. The proposed model, based on Stochastic Activity Networks (SANs), captures both failure and recovery dynamics involving containerized applications and the effects of software aging. The investigation has shown that adopting containerized technologies with standard deployments having both single and multiple replicas deployed into a single physical node is not sufficient for achieving “5-nines” availability. The sensitivity analysis also revealed that non-aging-related software failures and software repair stand out as key important failure and repair parameters, respectively. When clustering mechanisms such as Docker swarm mode with separated worker and manager nodes are adopted, we observed that the MANO availability is further increased and the above parameters become less critical when multiple MANO container replicas are engaged. Software aging may have a considerable impact on the availability and we showed the relationship between aging effects and failures related to it. As future work we will consider modeling and analysis of more complex swarm deployment options similar to those designed for Docker Enterprise solutions.

Acknowledgment

This research was funded by the joint EU FP7 Marie Curie Actions Cleansky Project, Contract No. 607584.

References

[1] G. N. ETSI, “Network Functions Virtualisation (NFV): Architectural Framework,” *ETSI GS NFV*, vol. 2, no. 2, p. V1, 2013.
[2] Docker Website. Accessed: 2019-10-30. [Online]. Available: ”https://www.docker.com/”
[3] LXD. Accessed: 2019-10-30. [Online]. Available: ”https://linuxcontainers.org/”
[4] N. Dragoni *et al.*, “Microservices: yesterday, today, and tomorrow,” in *Present and ulterior software engineering*. Springer, 2017, pp. 195–216.
[5] OSM website. Accessed: 2019-10-30. [Online]. Available: ”https://osm.etsi.org”
[6] OpenBaton website. [Online]. Available: ”https://openbaton.github.io”
[7] SONATA website. Accessed: 2019-10-30. [Online]. Available: ”https://www.sonata-nfv.eu/”
[8] ETSI, “Reliability; Report on Models and Features for E2E Reliability,” ETSI, Tech. Rep. GS REL 003 v1.1.2, 2016-07.
[9] A. J. Gonzalez *et al.*, “Dependability of the NFV orchestrator: State of the art and research challenges,” *IEEE Communications Surveys & Tutorials*, vol. 20, no. 4, pp. 3307 – 3329, 2018.
[10] G. Nencioni *et al.*, “Orchestration and control in software-defined 5G networks: Research challenges,” *Wireless Communications and Mobile Computing*, vol. 2018, 2018.
[11] I. N. ETSI, “ETSI GS NFV-REL 001 v1. 1.1: Network Functions Virtualisation (NFV); Resiliency Requirements,” 2015.
[12] ——, “ETSI GR NFV-REL 007 v1.1.2: Network Function Virtualisation (NFV); Reliability; Report on the resilience of NFV-MANO critical capabilities,” 2017.
[13] A. Gonzalez *et al.*, “Service availability in the NFV virtualized evolved packet core,” in *Global Communications Conference (GLOBECOM), 2015 IEEE*. IEEE, 2015, pp. 1–6.
[14] M. Di Mauro *et al.*, “IP multimedia subsystem in an NFV environment: availability evaluation and sensitivity analysis,” in *2018 IEEE NFV-SDN*. IEEE, 2018, pp. 1–6.
[15] ——, “Service function chaining deployed in an NFV environment: An availability modeling,” in *IEEE CSCN*. IEEE, 2017, pp. 42–47.
[16] ——, “Availability modeling and evaluation of a network service deployed via NFV,” in *International Tyrrhenian Workshop on Digital Communication*. Springer, 2017, pp. 31–44.
[17] B. Tola, G. Nencioni, B. E. Helvik, and Y. Jiang, “Modeling and evaluating NFV-enabled network services under different availability modes,” in *IEEE DRCN*. IEEE, March 2019.
[18] S. Sebastio, R. Ghosh, and T. Mukherjee, “An availability analysis approach for deployment configurations of containers,” *IEEE Transactions on Services Computing*, 2018.
[19] S. Sebastio, R. Ghosh, A. Gupta, and T. Mukherjee, “ContAv: A Tool to Assess Availability of Container-Based Systems,” in *IEEE SOCA*. IEEE, 2019, pp. 25–32.
[20] T. Soenen *et al.*, “Optimising microservice-based reliable NFV management and orchestration architectures,” in *IEEE RNDM*, Sep. 2017, pp. 1–7.
[21] W. H. Sanders and J. F. Meyer, “Stochastic activity networks: Formal definitions and concepts,” in *School organized by the European Educational Forum*. Springer, 2000, pp. 315–343.
[22] M. Grottke and K. S. Trivedi, “Software faults, software aging and software rejuvenation,” *The Journal of Reliability Engineering Association of Japan*, vol. 27, no. 7, pp. 425–438, 2005.
[23] M. Grottke, R. Matias, and K. S. Trivedi, “The fundamentals of software aging,” in *2008 IEEE ISSRE Wksp*, Nov 2008, pp. 1–6.
[24] K. S. Trivedi *et al.*, “Recovery from failures due to Mandelbugs in IT systems,” *Proceedings of IEEE PRDC*, no. December, pp. 224–233, 2011.
[25] Möbius: Model-based environment for validation of system reliability, availability, security and performance. Accessed: 2019-10-30. [Online]. Available: ”https://www.mobius.illinois.edu”
[26] Docker Reference Architecture: Docker Enterprise Best Practices and Design Considerations. Accessed: 2019-10-30. [Online]. Available: ”https://success.docker.com/article/docker-enterprise-best-practices\#highavailabilityindockerenterprise”