NHAM: An NFV High Availability Architecture for Building Fault-Tolerant Stateful Virtual Functions and Services

Giovanni Venâncio, Elias P. Duarte Jr.
{gvsouza,elias}@inf.ufpr.br
Dept. Informatics - Federal University of Paraná - Curitiba, Brazil

ABSTRACT

Despite the many advantages of Network Function Virtualization (NFV) technology, the dependability of virtual services must be carefully addressed so that NFV can meet the requirements of commercial carriers. In particular, it is essential to provide mechanisms to ensure their correct and continuous operation. In this work we propose NHAM: an NFV High Availability Module designed within the NFV-MANO (NFV Management and Orchestration) reference model. NHAM allows the creation and management of fault-tolerant virtual network services consisting of stateful VNFs (Virtualized Network Functions) and SFCs (Service Function Chains). The proposed architecture provides fault management, including a choice of recovery mechanisms that can be applied depending on the specific needs of each service. The solution is holistic in the sense that it does not require any modifications of the source code of VNFs/SFCs to make them fault-tolerant. The strategy is based on SFC buffer management coupled with VNF checkpoint/restore, providing high availability in a transparent way. A prototype was implemented and experimental results are presented.

KEYWORDS

Network Function Virtualization, Virtualized Network Functions, High Availability, Fault Tolerance

1 INTRODUCTION

Virtualization is perhaps the single most important technology that can lead to a solution to the "Internet ossification" problem: when decades-old Internet protocols were designed it was hard to anticipate the extraordinary growth that has since taken place. With virtualization technologies, the network becomes programmable and can evolve along multiple directions. Network Function Virtualization (NFV) is one of those technologies, which allows the replacement of hardware-based middleboxes by software that runs on off-the-shelf hardware [19]. Network services are implemented with Virtual Network Functions (VNFs), which can be further combined in complex Service Function Chains (SFCs), that consist of multiple VNFs connected in a predefined order [6, 7, 9]. With NFV technology, network services that were previously available from a handful of vendors can now be downloaded from Internet marketplaces [1]. The adoption of NFV technology represents a major impact in terms of improving network flexibility and management. In order to standardize the execution and management of NFV-based services and to allow the interoperability of a broad range of VNFs, the European Telecommunications Standards Institute (ETSI) has proposed the NFV-MANO architecture [22].

Despite its many advantages, it is undeniable that network services executed as virtualized software are more prone to failures than traditional alternatives available as specialized hardware [11]. The shift from hardware devices to virtualized platforms brings several challenges in terms of dependability [17, 26]. The complexity to integrate multiple software systems in different layers, the interoperability of hardware and software components provided by different vendors, and the lack of experience on operating virtualized network environments are some of the factors that make it a challenge to ensure the dependability of NFV-based networks.

On the other hand, middleboxes implemented on specialized, often proprietary hardware are typically designed with strict resilience goals. which corresponds to slightly more that 5 minutes of downtime per year. In order to ensure the adoption of NFV technology, it is essential to guarantee carrier-grade availability (five nines, 99.999%). The ETSI has defined a number of resiliency requirements for services running in virtualized environments [11, 16].

Several proposals have been made to increase the availability of network functions in virtualized environments [8, 14, 15]. However, all those solutions present limitations, such as the use of particular technologies or the requirement to modify VNF code. Some do not incorporate all the mechanisms needed to ensure end-to-end availability. The challenges soar by taking into account that most network functions are *stateful* (*i.e.,* they require fine-grained function state management). Finally, none of the existing solutions is fully compliant with the NFV-MANO reference architecture defined by the ETSI [22].

In this work we propose a high availability architecture for NFV-based services, including both stateful VNFs and SFCs. The architecture, called NHAM (NFV High Availability Module), is integrated as a module to the NFV-MANO reference architecture, being fully compliant with the specifications defined by the ETSI. The strategy adopted by NHAM is defined on the virtualization level, so that any VNF or SFC instantiated on the NFV platform can inherit the high availability and resiliency properties in a transparent way.

NHAM performs fault management, while also managing the internal state of VNFs and providing multiple mechanisms to ensure high availability. A set of operations is proposed to monitor and control the internal state of VNFs with techniques based on checkpoint/restore. In this way, NHAM ensures that after a VNF fails it can be recovered preserving the internal state as it was before the failure occurred. NHAM offers a choice of four different resiliency mechanisms, used to configure and update VNF replicas. The resiliency mechanisms vary in terms of computational resources and recovery time, allowing different types of VNFs with different availability requirements to recover after failures.

NHAM was designed to be agnostic to the implementation: any NFV-based service becomes highly available without the developer having to make any changes to the VNF source code. As VNFs are executed in virtualized environments, checkpoints can be taken

efficiently by saving the network function instance. This is a generic approach that can be used to preserve the service state without requiring VNF code modifications.

Furthermore, NHAM addresses the availability of stateful SFCs, presenting a strategy to build resilient SFCs. The strategy couples checkpointing with buffer management, allowing the synchronization of the traffic processed by each VNF with the corresponding checkpoints. As a consequence, NHAM ensures complete and correct end-to-end service recovery, tolerating multiple VNF failures while also preventing packet losses and duplications due to failures.

A prototype was implemented, and experiments were executed to assess the performance and availability of NFV-based services with the support of NHAM. We show that, depending on the strategy and the parameters, it is possible to reach carrier-grade availability.

The rest of this work is organized as follows. Section 2 presents an overview of NFV and the NFV-MANO architecture, also describing SFCs. Section 3 presents an overview of the NHAM architecture, while the SFC fault-tolerance strategy is described in Section 4. Section 5 presents the implementation and experiments. Section 6 presents related work. Finally, Section 7 concludes the paper.

2 VIRTUAL FUNCTIONS AND SERVICES: AN OVERVIEW

This section presents a brief overview of NFV technology including the NFV-MANO reference model, and the IETF SFC architecture.

2.1 NFV: Network Function Virtualization

Network Function Virtualization (NFV) was proposed as an alternative based in software to implement network middleboxes, such as firewalls, Network Address Translation (NAT) devices, Intrusion Detection Systems (IDS) among others. Middleboxes have been traditionally implemented as specialized hardware [25], and are often hard to manage and troubleshoot [27]. These services represent a significant fraction of the capital expenditures (CAPital EXpenditures - CAPEX) and operating expenses (OPerational EXpenditures - OPEX) of a network [2]. NFV reduces the cost, improves the flexibility, simplify the design, development, and management of network services [19]. There also other advantages, for instance in terms of energy and physical space requirements [10].

The ETSI has coordinated efforts that led to the NFV-MANO (NFV Management and Orchestration) reference architecture [22]. NFV-MANO allows the interoperability of virtual functions and services from different developers, and includes modules for VNF control and orchestration, including lifecycle and resource management. In addition, NFV-MANO defines communication interfaces and provides abstractions for the resources needed to execute VNFs. Figure 1 shows the NFV-MANO architecture, the NFVI (NFV Infrastructure) and the VNFs themselves.

The NFVI corresponds to the virtualized infrastructure in which the VNFs are instantiated, managed, and executed. The NFVI includes physical storage, network and computational resources. Physical resources are abstracted into virtual resources through a virtualization layer, which is composed of a hypervisor that creates and manages virtualized devices, *i.e.* Virtual Machines (VM) and

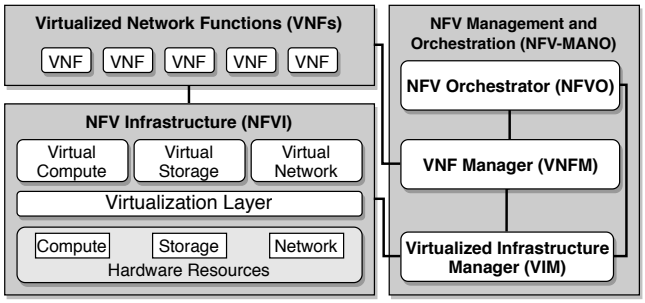

Figure 1: NFV-MANO reference architecture.

containers. This allows isolation so that each VNF can run independently. The VNFs in Figure 1 represent the instances that are executing on the NFVI.

NFV-MANO itself is organized in three main modules. The first is the NFV Orchestrator (NFVO), which allows the composition of VNFs on SFCs (Service Function Chains), also managing their lifecycle and resources [13]. The second module is the VNF Manager (VNFM), which is responsible for VNF lifecycle management, including the instantiation, deletion, configuration, and auto scaling of VNFs [31]. To perform its functionalities, the VNFM makes use of the VNF Descriptor (VNFD), a template responsible for the specification of each VNF in terms of operational and deployment requirements. Finally, the Virtualized Infrastructure Manager (VIM) controls and manages the computing resources of the NFVI. The VIM is primarily responsible for the creation, deletion, and reconfiguration of virtual devices.

In terms of VNF availability, the ETSI has specified several resiliency requirements for NFV environments and platforms [20, 24]. In particular, an NFV platform must support ensure the resiliency of VNFs of different types and provided by different vendors. In addition, it is expected that different levels of resiliency can be defined as different VNFs have different requirements. Furthermore, to ensure high availability, an NFV platform must provide a complete fault management system, being able to detect and recover from VNF failures. Finally, the NFV platform must ensure that stateful VNFs preserve their internal state in case of failure.

While many NFV platforms are fully compliant with the NFV-MANO architecture, no MANO-compliant solution provides the full set of functionalities required to ensure end-to-end availability for VNFs and SFCs. The purpose of the present work is to fill this gap by proposing a high availability NFV architecture integrated to the NFV-MANO reference model.

2.2 Service Function Chains

While VNFs perform specific functions, they can be combined into complex network services, the SFCs. An SFC consists of multiple VNFs connected in a predefined order through which traffic is steered [9, 12]. According to the IETF (Internet Engineering Task Force), the architecture of an SFC (shown in Figure 2) consists of *Classifiers*, *Service Function Forwarders* (SFFs), and the VNFs themselves, briefly described next.

The Classifier receives the network traffic as input and determines using policies to which SFC the traffic should be forwarded.

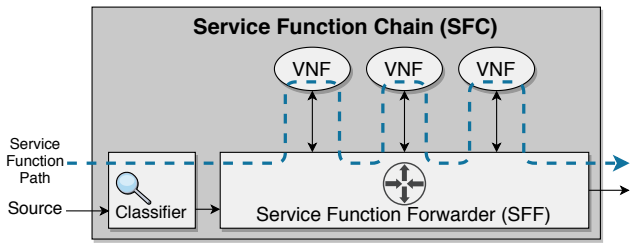

Figure 2: SFC Architecture.

Policies can be based on several fields, such as source and destination IP addresses, ports, protocol (*e.g.,* TCP, UDP), among others. The Classifier also forwards traffic to the required SFCs. The path along which the traffic goes through within an SFC is called Service Function Path (SFP) [9]. Once classified, the traffic is encapsulated so that it can be routed to the correct SFP. As an SFC may consist of multiple SFPs, headers include the identifier of the SFP to be used.

The SFF is responsible for forwarding packets from the Classifier to one or more network functions in a pre-defined order. In order to do this, the SFF uses the information inserted by the Classifier in the SFC header. After a VNF processes the incoming traffic, it forwards the processed packets back to the SFF. Then, the SFF forwards the traffic to the next VNF of the SFP, and this process is repeated until the traffic has been processed by all VNFs. Finally, when receiving the traffic from the last VNF of the SFP, the SFF removes the header from the packets and forwards the traffic to the final destination.

3 NHAM: A HIGH AVAILABILITY NFV ARCHITECTURE

NHAM (NFV High Availability Module) is a high availability architecture for NFV that defines strategies for VNF and SFC resiliency, including failure detection and recovery, covering heterogeneous functions and services from different providers. A highly available SFC continues its correct execution even after faults occur, *i.e.* after one or more VNFs crash. The availability of a service increases as the recovery time reduces. Therefore, it is necessary to detect and react to failures quickly, minimizing downtime. However, the problem cannot be solved by just using redundancy neither by simply re-instantiating failed functions [30, 32]. Most VNFs are *stateful* – the internal state of the network function changes according to the packets processed and the execution flow of the function itself. Thus the VNF state must be preserved after the recovery.

NHAM is a high availability solution for stateful NFV-based services, integrated as a module to the NFV-MANO reference architecture. NHAM includes efficient fault management features. VNFs simply *inherit* high availability properties, with no need for developers to make any changes to the source code in order to make a service highly-available. NHAM assumes the classical crash fault model. A description of the NHAM architecture is presented in the next subsection. The strategy defined by NHAM to ensure high availability of individual VNFs follows.

3.1 NHAM: The Architecture

NHAM consists of two main components, as shown in Figure 3: the Fault Management System (FMS) and the VNF State Manager (VSM), described next. Fault management systems include functionalities for failure detection and recovery. In the context of NFV, failure detection consists of monitoring VNF instances, and identifying crashes [29]. NHAM has two mechanisms for failure detection. The Failure Detector (FD) module employs a polling strategy – messages are sent periodically to the VNFs being monitored; acknowledgements should arrive before a timeout expires. The timeout is computed adaptively. In addition to the polling messages, the FD checks the state of the VNF by directly inspecting the corresponding virtual device – a functionality provided by several hypervisors. Immediately after a VNF is suspected to have crashed, it is added by the FD to a list of suspects and a notification message is sent to the VNFM.

In addition, the FMS also has a Replica Manager, which features a choice of resiliency mechanisms, described in Subsection 3.3. The resiliency mechanisms are used to manage VNF replication. For each VNF instance, one of four different resiliency mechanisms can be specified. The choice is done according to the particular VNF availability requirements. The FMS is also responsible for VNF recovery. In order to recover a failed VNF, NHAM communicates with the VIM, VNFM, and NFVO, also described in Subsection 3.3.

In order to preserve the internal state of a VNF after its recovery – ensuring the correct recovery of stateful VNFs – NHAM employs the VNF State Manager (VSM) component. The VSM consists of a State Synchronizer, an API for handling the internal state of VNFs, and a database responsible for storing the VNF states. The VSM is described in the next subsection.

NHAM communicates with other NFV-MANO modules, in particular the NFVO, VNFM, and VIM. Recall that those modules perform, among other tasks, those related to the lifecycle of virtualized services, including the instantiation, configuration and termination of VNFs and SFCs. As an example of the NHAM-MANO interaction, during the recovery of a VNF NHAM requests the creation of new VNFs to the VNFM. Another example: NHAM reconfigures SFCs through the NFVO.

3.2 Management of Stateful VNFs

The VSM is the component responsible for the recovery of stateful VNFs. The VSM is based on checkpoint/restore [4]. As VNFs run on virtual devices, which can be either virtual machines or containers, saving the network function instance is a feasible and attractive strategy to capture the VNF state without requiring any modification of the VNF source code. Periodically, checkpoints containing a representation of the system state are captured and saved in non-volatile memory. When a failure is detected, the system can be restored to the most recently saved checkpoint.

The state of a VNF consists of information that can be classified as either external or internal [20]. The external state consists of static information that does not change – or changes infrequently – over time. Firewall/IDS rules and NAT port mapping tables are examples of information stored as external state of a network function. The external state can be easily recovered when the VNF recovers.

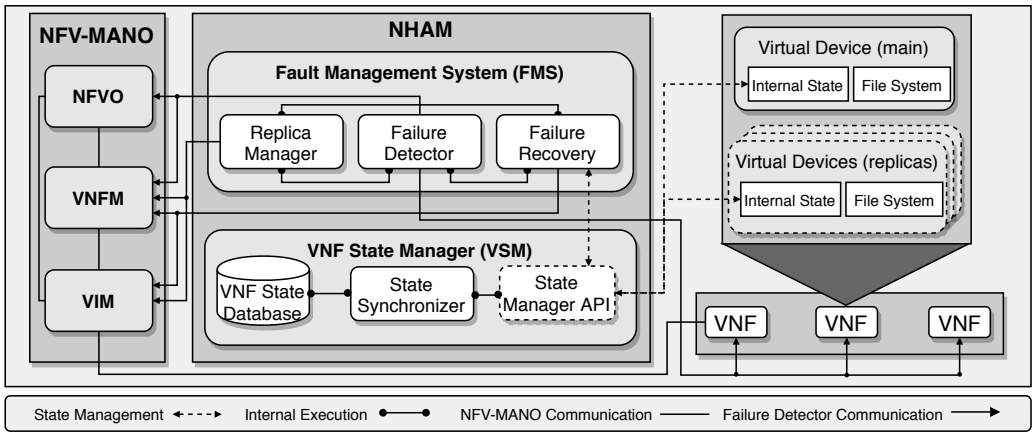

Figure 3: The NHAM Architecture.

The internal state consists of information that is updated as packets are processed and the function follows its execution flow. The memory mapping, TCP connections, and cache contents are examples of internal state information. The main challenge of state management is to preserve and guarantee the consistency of the internal state as VNFs fail and recover.

The main component of the VSM is the State Synchronizer, responsible for capturing internal state information and saving VNF checkpoints. For each VNF, the State Synchronizer employs an agent that periodically collects internal state information. NHAM defines an API for VNF state management, which consists of two main operations, described next.

export_vnf_state. This operation saves a checkpoint of a specific VNF. First, the virtual device has to be momentarily paused, so that the state information for the checkpoint can be obtained. Once the required state information is obtained, the VNF execution is resumed. After it is ready, the checkpoint is sent either to the VNF State Database or directly to a replica, depending on the resiliency mechanism adopted.

import_vnf_state. This operation is employed to restore the state of a VNF with a checkpoint. Two parameters are specified: *(i) vnf* is the VNF instance identifier and; *(ii) checkpoint*, indicates from where the corresponding checkpoint has to be imported. In the execution of the *import_vnf_state* operation, the first step is to momentarily pause the VNF that will be updated with the checkpoint. Then, the checkpoint is imported and the VNF is updated. After the operation completes, the VNF outputs a code indicating that it was successfully updated with the new checkpoint.

Note that NHAM also allows the recovery of stateless VNFs, for which it is not required to save state information.

3.3 Resiliency Mechanisms and Failure Recovery

Choosing a strategy to ensure the high availability of VNFs depends on the resiliency requirements of each specific network function [24]. For example, network functions that handle real-time traffic have more strict requirements than functions that handle best-effort traffic. Therefore, the NFV platform should support different strategies with different properties and costs.

NHAM presents four different resiliency mechanisms that rely on two replication methods, which are defined by an ETSI standard [20]: Active-Standby and Active-Active. In the Active-Standby method, the VNF replica has already been instantiated but is in standby mode. In the Active-Active method, the replica has been instantiated and is active, periodically updating its state.

The resiliency mechanisms vary in terms of cost and recovery time. The selection of one of those mechanisms for a specific VNF depends on the VNF features and requirements. Each of the resiliency mechanisms employs a different recovery procedure. The resiliency mechanisms and the corresponding recovery procedures are described next.

3.3.1 No Redundancy (0R). As the name suggests, the No Redundancy (0R) mechanism does not employ any type of redundancy. In this case, the State Synchronizer exports the VNF checkpoints to the VNF State Database. This approach is the simplest and the cheapest in terms of computational resources required, as it does not use virtual resources to maintain any replicas.

On the other hand, after a failure occurs, the recovery time is longer than that of the other methods. As the 0R mechanism does not employ any replicas, the first step of the recovery process is to instantiate a new VNF, replacing the one that has failed. Next, NHAM updates the internal state of the new VNF. To do so, the State Synchronizer imports the most recent checkpoint of the failed VNF to the newly created VNF. Once the recovery process is complete, a reconfiguration process begins. The first step is to obtain the updated information of the newly instantiated VNF, including its IP address and other identifiers. Then, NHAM sends this updated information to the corresponding NFV-MANO modules.

3.3.2 Primary Replica Active-Standby (1R-AS). The Primary Replica Active-Standby (1R-AS) resiliency mechanism employs the Active-Standby method with a replica that is instantiated but remains in a standby mode. In this mechanism, as in the 0R mechanism, the State Synchronizer exports the VNF checkpoints to the VNF

Table 1: Comparison of the different resiliency mechanisms.

Resiliency Mechanism	Method	#Replicas	Database	Recovery Time	Resource Usage	Reconfiguration
0R	None	0	Yes	Very High	Very Low	Yes
1R-AS	Active-Standby	1	Yes	Moderate	Low	Yes
1R-AA	Active-Active	1	No	Low	High	Yes
MR-AA	Active-Active	M	No	Very Low	Very High	No

State Database. This option is more expensive than 0R because it consumes virtual resources to maintain the replica, but has a shorter recovery time.

The difference is that upon a failure, the replica is already created and the State Synchronizer only needs to import the most recent checkpoint into the replica, thus updating its internal state. Finally, the reconfiguration procedure is executed. The replica becomes the primary VNF and NHAM sends a request to NFV-MANO to update the required information. Then, a new replica is instantiated and left in standby mode.

3.3.3 Primary Replica Active-Active (1R-AA). The Primary Replica Active-Active (1R-AA) resiliency mechanism employs the Active-Active method, in which the VNF has an active replica. Having such a replica means that the State Synchronizer periodically updates the replica with the VNF checkpoints. Therefore, the replica constantly receives updates reflecting the state of the primary VNF. The 1R-AA strategy is more expensive than the 1R-AS, however the recovery time is shorter.

Upon the occurrence of a failure, no particular recovery procedure is required besides a reconfiguration to indicate that the replica is now the primary VNF. Finally, a new replica is instantiated to replace the one that became the primary.

3.3.4 Multiple Replicas Active-Active (MR-AA). The Multiple Replicas Active-Active (MR-AA) resiliency mechanism is a generalization of the 1R-AA mechanism, in which the VNF can be seen as a member of a group of M replicas that are continuously synchronized. The State Synchronizer is responsible for keeping the states of the M replicas consistent. In addition, any of the replicas in this group can be accessed to obtain the service. MR-AA is the most expensive of all mechanisms, as it requires the synchronization of the M replicas, but presents the shortest downtime.

In case a failure, this mechanism does not require any reconfiguration, since a user can simply access any replica in the group. The only reconfiguration required is due to the fact that it is possible to specify the minimum and maximum number of replicas in a group. If the number of correct replicas gets below a minimum threshold, then new replicas are created.

3.4 A Comparison of the Resiliency Mechanisms

A comparison of the different resiliency mechanisms both in terms of resource usage (*e.g.,* memory and CPU utilization) and recovery time is shown in Table 1.

The 0R mechanism presents the lowest cost, and has the longest recovery time. It is ideal for VNFs that execute low priority functions and can tolerate longer failover times. The 1R-AS approach has a recovery time shorter than the 0R, as it does not require the creation of a new replica during the recovery process. This approach fits VNFs that have high disk capacity or are compute-intensive, as the standby replica does not consume computing or networking resources.

The 1R-AA and MR-AA strategies based on the Active-Active method have the shortest recovery times since the state of the replicas is constantly updated. On the other hand, they have the highest resource utilization. They are ideal for critical VNFs that require higher levels of availability.

Of all mechanisms, MR-AA is the most expensive since it maintains a group of synchronized replicas. Thus, NHAM must guarantee that the state is consistent across the multiple replicas. The State Synchronizer is a centralized component responsible for ensuring the whole group is updated correctly. To do this, the State Synchronizer waits for confirmations from each replica that the state has been updated. Only after receiving acknowledgments from all M replicas the State Synchronizer can start the synchronization of the next state. The State Synchronizer requires a replica monitoring strategy to identify failed replicas.

NHAM must also guarantee the consistency of replica states in two other specific situations: *(i)* when a VNF is incorrectly suspected to have failed and; *(ii)* when a VNF fails as the state is being updated. In the first case (false suspicions of failure), NHAM performs the same procedure as if it the replica had actually failed. A reconfiguration step is executed through which a new instance or an existing replica replaces the one that is mistakenly suspected of having failed. Now consider the second case, in which a VNF fails as the State Synchronizer is still updating the replicas. In this case, in order to avoid any inconsistencies, the State Synchronizer stops the process and rolls back all replicas to their previous state using the last checkpoint that was correctly saved. The failed VNF is eliminated, and the correct replicas remain consistent.

4 HIGHLY AVAILABLE SFCS

In this section we describe the strategy employed by NHAM to make SFCs highly available.

4.1 Buffer Management and Assumptions

NHAM adopts a high availability strategy that guarantees the end-to-end recovery of stateful SFCs (*i.e.,* composed of one or more stateful VNFs) after an arbitrary number of VNFs fail along the service chain. In most SFC implementations, each VNF along a SFP is preceded by a buffer, used by the SFF to store packets before they are delivered to the VNF. NHAM employs two buffers for each VNF

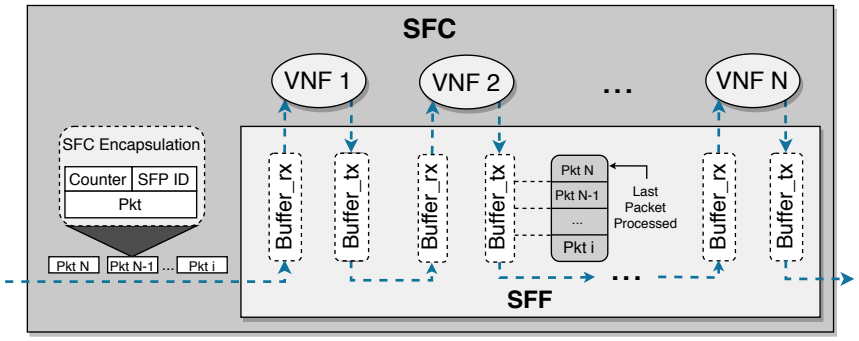

Figure 4: The High Availability Architecture for SFCs.

along the chain, as shown in Figure 4. *buffer_rx* precedes the VNF in the chain and receives the traffic to be delivered to the VNF; thus *buffer_rx* stores packets that have not yet been processed by the VNF. *buffer_tx* follows the VNF in the chain and receives traffic output by the VNF; thus *buffer_tx* stores traffic that has already been processed by the VNF.

The data flow starts as the *buffer_rx* of the first VNF of the chain receives from the SFF the packets which will be processed by the VNF. Later on the SFF forwards the packets from *buffer_rx* to the VNF. After processing the traffic according to the network function it implements, the VNF outputs the packets into *buffer_tx*. The SFF acts again moving packets from *buffer_tx* of a VNF to the *buffer_rx* of the next VNF along the chain. This process is then repeated for all other VNFs along the chain. After the last VNF processes the traffic and places the packets in the final *buffer_tx*, it is the SFF that is responsible for properly delivering this traffic to the destination.

We assume that the buffers, the SFF, and other MANO components do not fail. This is a realistic assumption: in order to support highly available SFCs the environment on which these SFCs run must have itself been designed to be fault-tolerant. VNFs are not shared by multiple SFCs, i.e. each VNF is employed by a single SFC. It is recommended to deploy VNFs and buffers on physically decoupled hardware. Furthermore, the recovery strategy assumes that all VNFs along the chain process traffic in FIFO order. Thus whenever two packets are sent in a certain order to the VNF, if they are not dropped by the VNF, they are output preserving that order.

The Hold/Release strategy is proposed to guarantee the consistent recovery of an SFC after a failure. Hold/Release is based on a combination of VNF checkpointing coupled with buffer management, and is described next.

4.2 The SFC Hold/Release Strategy

The recovery of a stateful SFC consists of two main parts: *(i)* the individual recovery of each VNF that has failed, including state restoration in case of stateful VNFs (described in Section 3.2); and *(ii)* the retransmission of traffic lost due to VNF failures, which is done with the Hold/Release strategy described next *(ii)*.

Before the SFF places each packet in the first *buffer_rx* of the first VNF of the SFC, each packet is encapsulated so that it can be routed along the SFP. Besides carrying explicit information used to identify the SFP [9], as the SFF encapsulates a packet it also adds a timestamp, which is implemented as a counter and works as a sequential packet identifier.

NHAM continuously monitors the VNFs, and upon identifying the failure of any VNF, NHAM immediately sends a message to the SFF to change the state of the SFC to *recovering*. When a SFC is in this state, traffic does not progress through the VNF until it has fully recovered. Thus a VNF in the *recovering* state neither receives packets from *buffer_rx* nor sends packets to *buffer_tx*.

In the Hold/Release strategy, after the packets in *buffer_rx* are delivered to the VNF, they are not removed from the buffer, *i.e.* they are temporarily retained. The VNF then receives and processes the packets, and stores the resulting traffic on *buffer_tx*. The idea is to keep the traffic in *buffer_rx* until a VNF checkpoint is taken after it has processed that traffic. If a checkpoint is taken after a given packet has been processed we say that the checkpoint *includes* that packet.

In case the VNF fails before the checkpoint is taken, it is rolled back to the previous checkpoint, and all packets it had received from that point (which are still in *buffer_rx*) must be sent again and processed by the VNF. In case the VNF does not fail, the SFF waits until the checkpoint is saved. At this point, it is possible to conclude that the last packet in *buffer_tx* has been both processed by the VNF and included in the checkpoint. Then, the SFF removes all packets up to that last packet from *buffer_rx*.

Consider as an example that all packets up to packet i have been processed by a VNF when a checkpoint starts. Consider that packet $i+1$ had also been sent from *buffer_rx* to the VNF, but was not included in the checkpoint. As the checkpoint completes, the SFF confirms that the last packet that was already in *buffer_tx* is packet i and can conclude that this packet was included in the checkpoint. Now all packets up to i can be removed from *buffer_rx*. Note that packet $i+1$ cannot be removed: if it is necessary to rollback, packet $i+1$ must be reprocessed by the VNF. NHAM also keeps track of the last packet delivered to the next VNF along the chain, which can be used to avoid sending duplicate packets along the SFC.

So the strategy can be summarized as follows: *(i)* packets in *buffer_rx* have not yet been processed by the VNF (although they may have been delivered to the VNF); *(ii)* the state of the VNF is updated as it processes each packet; and *(iii)* packets processed by the VNF are output to *buffer_tx*. The SFF keeps track of both the last packet included in the last checkpoint (described above) and of the last packet delivered to the next VNF along the chain, this is the last

packet stored in *buffer_tx*. This allows avoiding sending duplicate packets to the next VNF after a recovering VNF reprocesses traffic to rebuild its state.

Consider for example another situation in which packets i, $i+1$, and $i+2$ have been sent from *buffer_rx* to the VNF. However, the VNF had only processed up to packet i when a checkpoint started. As the checkpoint completes, the SFF confirms that the last packet in *buffer_tx* is packet i, and can then remove all packets up to that one from *buffer_rx*. Consider however that after the checkpoint completes the VNF continues processing and outputs packets $i+1$ and $i+2$ to *buffer_tx* which are forwarded to the next VNF. The SFF keeps track of the last packet in *buffer_tx*, in this case packet $i+2$. Now the VNF fails. As packets $i+1$ and $i+2$ were not included in the last checkpoint, they must be reprocessed after the VNF recovers. However, they have been already forwarded to the next VNF along the chain. After the recovery, the SFF avoids duplicates by forwarding only packets from $i+3$ along the SFC.

The recovery of the VNF proceeds according to its respective resiliency mechanism, as described in Section 3.3. After the VNF is up again with its state restored according with the last checkpoint saved, the next step is the retransmission of all the traffic in *buffer_rx*, including packets the VNF had received since the checkpoint was saved.

The Hold/Release strategy ensures that the SFC recovers regardless of the number of VNFs that have failed. Multiple VNFs may even fail at the same time, for example due to a power failure. Note that as the *buffer_rx* of a given VNF is only cleaned after a VNF checkpoint is saved and the corresponding processed packet is in *buffer_tx*, the traffic processed between the checkpoints of each VNF is not lost and the consistency of the SFC as a whole is guaranteed.

5 IMPLEMENTATION AND EXPERIMENTAL EVALUATION

NHAM was implemented as a prototype within an NFV platform that is compliant with the NFV-MANO reference model. The prototype was implemented in Python and is based on Docker containers [18]. A REST API was implemented for VNF state management. CRIU (Checkpoint/Restore in Userspace) [3] was employed to take VNF checkpoints. The checkpoints consist of the minimum information required to restore a failed VNF, including the network function itself and a few related resources, such as memory maps and the process tree. The VNFs employed in the experiments are packet forwarders.

Experiments were executed on an Intel Core i7 processor at 2.50GHz with 8 cores, 16 GB RAM, a 1Gbps Ethernet NIC, and Linux Ubuntu 20.04. Each VNF consists of an Ubuntu server with 256 MB RAM and 1 CPU. The MR-AA mechanism uses 3 replicas by default. NHAM does not require any kernel patches in order to work. The first set of experiments evaluates and compares the impact of the four different VNF resiliency mechanisms during SFC recovery time as the number of VNFs of the SFC grows. The second set of experiments evaluates resource usage of each recovery strategy, both in terms of memory and CPU utilization. The third set of experiments measures the impact of NHAM on the throughput. Finally, the last experiment evaluates the availability of NHAM-supported NFV-based services. Each experiment was repeated 10 times and the results are averages presented with a confidence interval of 95%.

5.1 Failure Recovery Time

As a short downtime is particularly important to improve the availability of virtual services, the first set of experiments measures the total recovery time, from failure detection until recovery completes. The failures were injected using scripts that remove all connections from the VNFs. Therefore, monitoring messages are not received, which triggers failure suspicions. The first experiment, shown in Figure 5, measures the average recovery time after a single VNF of the SFC fails. This experiment compares the four different resiliency mechanisms, and the number of VNFs in the SFC increases from 1 to 8.

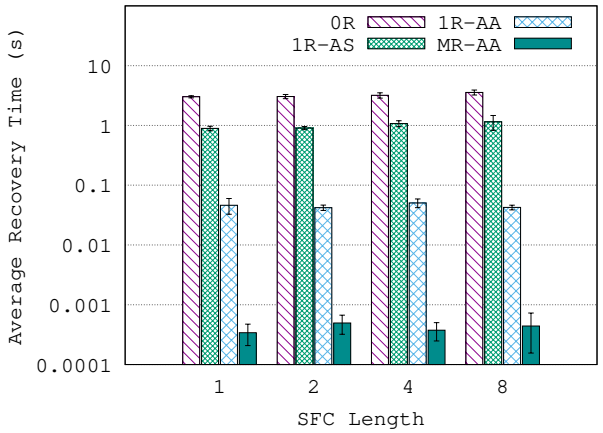

Figure 5: Average recovery time after a single VNF fails.

As predicted, the 0R mechanism presents the longest recovery time, reaching up to 3.6s of downtime for an SFC with 8 VNFs. This is the time it takes to instantiate a new VNF – which takes approximately 2.4s on average – and have the VSM restore the most recent checkpoint. The 1R-AS mechanism presents better results, and this can be explained by the fact that a replica has already been instantiated and is in standby mode. By using the Active-Standby method, NHAM only needs to import a checkpoint into the replica, and the total recovery time was of 1.14s for a SFC with 8 VNFs.

Mechanisms based on the Active-Active method present even better results. In particular, the 1R-AA mechanism reached an average of up to 0.05s of the total recovery time, as the active replicas already have the up-to-date state. Finally, as expected, the MR-AA mechanism achieved the best results, as it maintains a group of synchronized replicas. The MR-AA recovery time is only 0.0002s, which corresponds to the time to update the replica group. It is important to note that, for all strategies, the recovery time remained unchanged, regardless of the SFC length.

The next experiment evaluates the impact on the recovery time when multiple failures occur at the same time, for example, due to a link failure or a power outage. The experiment shown in Figure 6 employs SFCs with 8 VNFs and the number of failures per SFC varies from 1 (single VNF failure) to 8 (failure of the whole SFC).

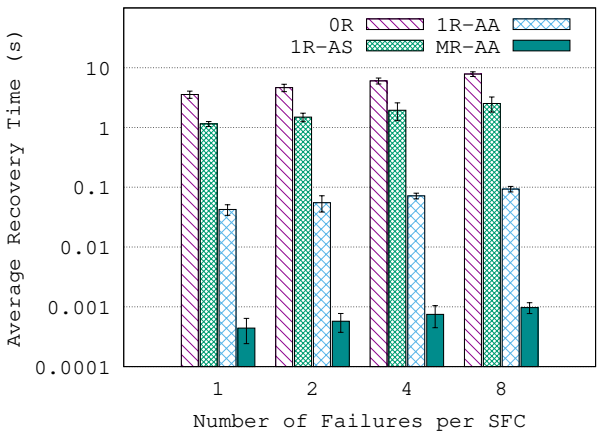

Figure 6: Average recovery time of multiple failures per SFC.

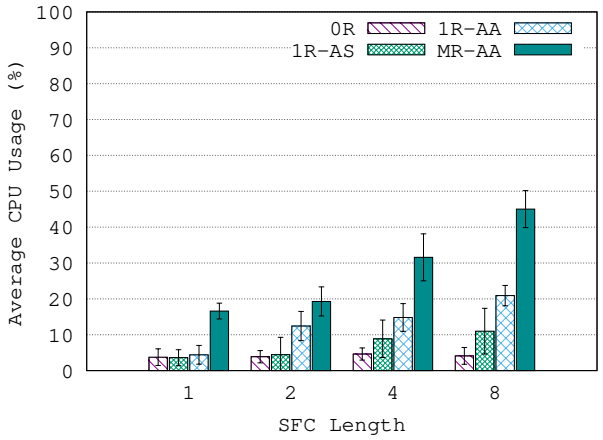

Figure 7: CPU usage.

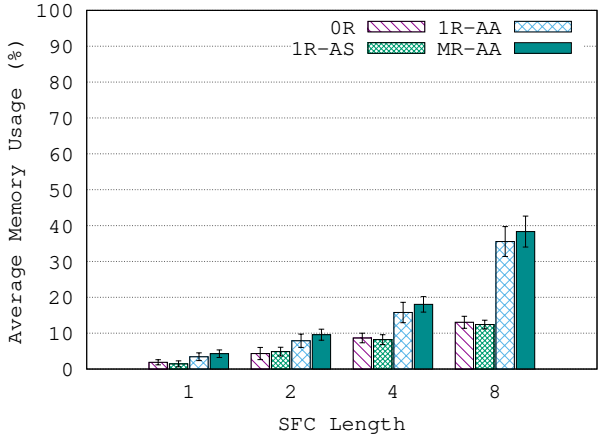

Figure 8: Memory usage.

As described in Section 4, NHAM detects and recovers multiple failed VNFs in parallel. We measured the impact of increasing the number of VNFs that fail simultaneously on the recovery time. For the 0R resiliency mechanism, the recovery time increases from 3.56s (one failure) to 7.8s (8 failures) – thus an increase of 2.1 times. For the 1R-AS mechanism, the difference between recovering a single failure and recovering the entire SFC is smaller, increasing from 1.14s to 2.52s.

For the 1R-AA and MR-AA mechanisms the difference is even smaller, since the time to recover from a single failure is significantly lower than that of the other strategies. For the 1R-AA strategy, the recovery time varies from 0.004s (1 failure) to 0.009s (8 failures), while for the MR-AA the recovery time varies from 0.0004s to 0.0009s. From these results, it is possible to conclude that all the recovery mechanisms are scalable with respect to the number of VNF failures.

5.2 Resource Usage

In next experiment we investigated the cost of the resiliency mechanisms in terms of memory and the CPU utilization, including the cost to monitor, recover, and synchronize the internal state of VNFs. Figures 7 and 8 show the results for memory and CPU utilization for each of the resiliency mechanisms as the length of the SFC length varies from 1 to 8 VNFs. The 0R mechanism presents a longer recovery time in exchange for lower cost. The 0R mechanism scales well as the number of VNFs grow: its CPU utilization remains roughly constant. For memory usage, the increase is proportional to the number of VNFs, ranging from 1.88% (1 VNF) to 13.04% (8 VNFs). On the other hand, although the 1R-AS mechanism has a shorter recovery time than 0R, it maintains the same performance levels as 0R, both in terms of CPU and memory.

As expected, mechanisms based on the Active-Active method present higher resource utilization, as they are constantly synchronizing the internal state of their replicas. The 1R-AA mechanism presents CPU utilization of 21% and memory usage of 38% to synchronize up to 8 VNFs. As for the MR-AA mechanism, despite having similar memory usage than the 1R-AA, its CPU usage is higher: up to 45% for a SFC with 8 VNFs. Note that memory usage of approaches based on the Active-Active method is significantly higher than the others. This is due to the fact that 1R-AA and MR-AA perform their operations in memory, avoiding non-volatile memory I/O overheads.

5.3 SFC Throughput

NHAM employs the Hold/Release strategy to guarantee the high availability of virtualized services. The next experiment measures the impact of that strategy on the throughput by evaluating the impact in two different scenarios: the first scenario is without failures while in the second failures occur every 30s. The SFCs used in these experiments consisted of 4 VNFs.

In the scenario without failures (Figure 9), each resiliency mechanism is compared to the baseline, which corresponds to a bare SFC that is not running NHAM. The 0R and 1R-AS mechanisms presented similar throughput, which was expected as both take checkpoints in the same way. These mechanisms decrease the throughput by approximately 11.5%; this is due to the fact that checkpoints are

obtained, compressed, and saved in non-volatile memory, increasing the amount of time that the VNF is stopped.

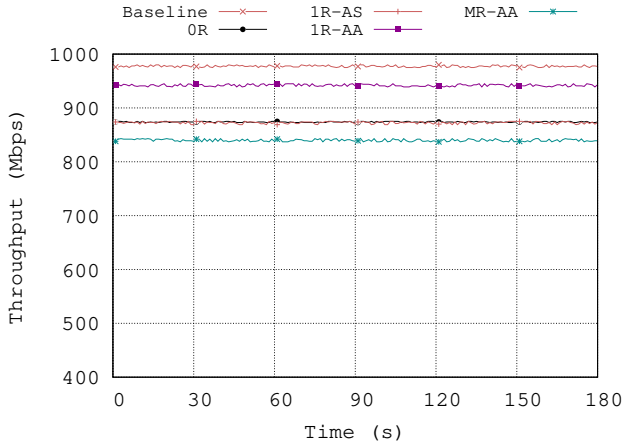

Figure 9: SFC throughput during a failure-free scenario.

For the 1R-AA mechanism, the throughput decreases by 4.7%. As seen in the previous section, this mechanism works in memory as the internal state is transferred to an active replica – this significantly improves the throughput. On the other hand, the MR-AA mechanism is the mechanism that presents the greatest degradation of the throughput. Despite having a very low recovery time, the throughput decreases by 14.1%. Like the 1R-AA, the MR-AA mechanism also runs in memory, however, all the processing done to guarantee that the group of replicas of each VNF remains consistent does have an impact on the throughput.

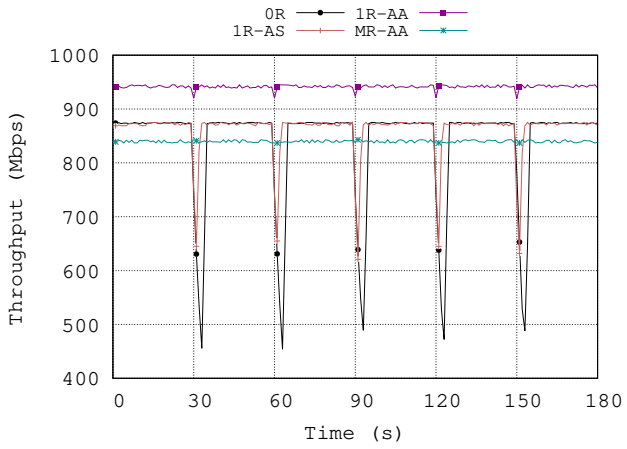

Figure 10: SFC throughput with failures.

In the scenario with VNF failures, shown in Figure 10, failures are injected every 30s. It is possible to observe that the reduction of the throughput is higher for the 0R and 1R-AS mechanisms than for the Active-Active based methods, which is a consequence of their longer recovery times. Note that even in this scenario, the throughput rate remains constant for the 1R-AA and MR-AA mechanisms.

5.4 Evaluating VNF Availability

In this experiment the availability of NFV-based services using NHAM was computed under a varying MTBF (*Mean Time Between Failures*). Table 2 presents the results for each resiliency mechanism. Each experiment lasted for 3 hours and the MTBF indicates the frequency at which failures were injected. In this experiment, SFCs with 8 VNFs were employed.

Table 2: SFC availability as the MTBF varies.

MTBF	Availability (%)			
	0R	1R-AS	1R-AA	MR-AA
60 (1 min)	98.057	98.240	99.916	99.999
300 (5 min)	99.605	99.643	99.983	99.999
600 (10 min)	99.802	99.821	99.991	99.999
900 (15 min)	99.868	99.880	99.994	99.999
1200 (20 min)	99.901	99.910	99.995	99.999
1500 (25 min)	99.920	99.928	99.996	99.999
1800 (30 min)	99.934	99.940	99.997	99.999

The 0R mechanism, as expected, presents the lowest figures, although it can be used for network functions that can tolerate longer recovery times. Even for tests with a higher MTBF (*e.g.* one failure every 30 minutes), 0R reached only 99.3% of availability. Likewise, the 1R-AS mechanism also does not reach the availability levels necessary to ensure carrier-grade availability of VNFs, despite achieving better results than 0R.

The 1R-AA mechanism performs better even in the most failure-prone scenario: with an MTBF of 60, VNFs achieved 99.9% (three nines) of availability and 99.99% (four nines) with an MTBF from 600. Naturally, the MR-AA mechanism presents the best results, reaching an availability of 99.999% (five nines) in all cases.

In general, cloud platforms which are used to deploy NFV environments reach up to approximately 99.9% (three nines) of availability [11]. The results shown in this section allows the conclusion that NHAM can ensure the high availability of VNFs with Active-Active based mechanisms, but this is further constrained by the availability of the cloud platform on which the VNF/SFC is running.

6 RELATED WORK

REINFORCE [15] is a framework to support resiliency for VNFs and SFCs based on replicating the state of network functions. While NHAM provides different resiliency mechanisms, REINFORCE employs a single Active-Standby method for resiliency. In addition, REINFORCE requires the VNF developers themselves to specify which VNF operations are stateful. Furthermore, REINFORCE it is not compliant with NFV-MANO.

FTC (Fault Tolerant Chain) [8] is a solution to improve the resiliency of SFCs that is neither based on checkpointing nor packet replay. FTC piggybacks VNF state information in packets that are propagated through the chain. Each VNF acts as a replica for its predecessor, avoiding the need to use dedicated replicas. After a VNF fails, it is re-instantiated and its state is retrieved from the successor on the chain. FTC is neither compliant with the IETF SFC reference architecture (*e.g.,* assuming that each VNF sends traffic directly to the next one) nor NFV-MANO.

In yet another proposal based on buffers [23], the authors propose Pico Replication (PR), a high availability framework for middleboxes. Instead of capturing the internal state of the middleboxes, PR performs checkpoints directly on specific data flows, while the middlebox continues to process other flows. PR requires several modifications to ensure the high availability of middleboxes, including changes to the kernel and SDN controller.

Another strategy proposed in [14] consists of decoupling the internal state of network functions from their processing. The internal state is saved to a distributed database. In this way, fault tolerance is achieved for stateful network functions, since in case of a failure the new instance can retrieve the updated state from the database, with the corresponding overhead.

An approach based on rollback-recovery is proposed in [28]. The FTMB (Fault-Tolerant Middlebox) system saves the state of middleboxes using two mechanisms: ordered logging and parallel release. Ordered logging is used to save the information needed to reproduce system entries in the event of a failure. Parallel release is an algorithm that complements ordered logging so that the correct reproduction of entries is ensured, given the dependencies between packets. Although this solution reduces the overhead as the system fails, the proposed strategy requires modifications to the VNF source code, which represents a limitation.

In addition to the works described above, most NFV and cloud platforms, such as OpenStack [21] and OSM [5], support some kind of fault tolerance. However, these solutions cannot guarantee the availability of stateful VNFs, since they do not have mechanisms to preserve the internal state of the virtual devices.

7 CONCLUSION

In this work we propose NHAM: an NFV high availability architecture which is defined as a module of NFV-MANO reference model. NHAM ensures the high availability of stateful VNFs and SFCs, without requiring source code modifications. Four different resiliency mechanisms are defined, which can be chosen depending on the features and requirements of the different types of VNFs. NHAM employs a state manager that couples checkpoint/restore with buffer management to allow the recovery of stateful SFCs even after multiple VNFs fail simultaneously, ensuring complete and correct end-to-end service recovery. A prototype was implemented and experimental results were executed to evaluate the performance and availability of NHAM-based VNFs and SFCs. Results show that NHAM is an effective solution to improve the robustness of virtualized services that can reach carrier-grade availability. Future work includes the investigation of strategies to improve fault prevention and prediction in the context of NFV.

ACKNOWLEDGMENTS

This work was partially supported by the Brazilian Research Council (CNPq) grant 308959/2020-5, and CAPES Code 001.

REFERENCES

[1] L. Bondan et al. 2019. FENDE: Marketplace-based distribution, execution, and life cycle management of VNFs. *IEEE Communications Magazine* 57, 1 (2019), 13–19.
[2] D. Cotroneo et al. 2014. Network function virtualization: Challenges and directions for reliability assurance. In *2014 IEEE ISSRE Workshops*. IEEE, 37–42.
[3] CRIU. 2022. Checkpoint/Restore In Userspace. https://criu.org/
[4] E. N. Elnozahy, L. Alvisi, Y. M. Wang, and D. B. Johnson. 2002. A survey of rollback-recovery protocols in message-passing systems. *ACM Computing Surveys (CSUR)* 34, 3 (2002), 375–408.
[5] ETSI. 2022. Open Source MANO. https://osm.etsi.org/
[6] V. Fulber-Garcia et al. 2020. Network service topology: Formalization, taxonomy and the CUSTOM specification model. *Computer Networks* 178 (2020), 107337.
[7] V. Garcia et al. 2019. An nsh-enabled architecture for virtualized network function platforms. In *International Conference on Advanced Information Networking and Applications*. Springer, 376–387.
[8] M. Ghaznavi, E. Jalalpour, B. Wong, R. Boutaba, and A. Mashtizadeh. 2020. Fault tolerant service function chaining. In *SIGCOMM*. ACM, Online, 198–210.
[9] J. Halpern and C. Pignataro. 2015. *Service Function Chaining (SFC) Architecture*. RFC 7665. IETF.
[10] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee. 2015. Network function virtualization: Challenges and opportunities for innovations. *IEEE Communications Magazine* 53, 2 (2015), 90–97.
[11] B. Han, V. Gopalakrishnan, G. Kathirvel, and A. Shaikh. 2017. On the resiliency of virtual network functions. *IEEE Communications Magazine* 55, 7 (2017), 152–157.
[12] A. Huff et al. 2018. A holistic approach to define service chains using Click-on-OSv on different NFV platforms. In *2018 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 1–6.
[13] A. Huff et al. 2020. Building multi-domain service function chains based on multiple NFV orchestrators. In *2020 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*. IEEE, 19–24.
[14] J. Khalid and A. Akella. 2019. Correctness and performance for stateful chained network functions. In *The 16th NSDI*. USENIX Association, Boston, 501–516.
[15] S. G. Kulkarni et al. 2018. Reinforce: Achieving efficient failure resiliency for network function virtualization based services. In *Proceedings of the 14th International Conference on emerging Networking EXperiments and Technologies*. ACM, Heraklion, 41–53.
[16] C. Lac, R. Adams, and et al. 2017. *Network Function Virtualisation (NFV); Reliability; Report on the resilience of NFV-MANO critical capabilities. GR NFV-REL 007 V1.1.1.* Technical Report. ETSI.
[17] J. Li, W. Liang, M. Huang, and X. Jia. 2020. Reliability-aware network service provisioning in mobile edge-cloud networks. *IEEE Transactions on Parallel and Distributed Systems* 31, 7 (2020), 1545–1558.
[18] D. Merkel. 2014. Docker: lightweight linux containers for consistent development and deployment. *Linux Journal* 2014, 239 (2014), 2.
[19] R. Mijumbi, J. Serrat, J-L. Gorricho, N. Bouten, F. De Turck, and R. Boutaba. 2016. Network function virtualization: State-of-the-art and research challenges. *IEEE Communications Surveys & Tutorials* 18, 1 (2016), 236–262.
[20] H. Nakamura, R. Adams, and et al. 2016. *Network Functions Virtualisation (NFV); Reliability; Report on Models and Features for End-to-End Reliability. GS NFV-REL 003 V1.1.1.* Technical Report. ETSI.
[21] OpenStack. 2022. OpenStack - Open source software for creating private and public clouds. https://www.openstack.org/
[22] J. Quittek, P. Bauskar, T. BenMeriem, A. Bennett, M. Besson, and et al. 2014. *Network Functions Virtualisation (NFV); Management and Orchestration. GS NFV-MAN 001 V1.1.1.* Technical Report. ETSI.
[23] S. Rajagopalan, D. Williams, and H. Jamjoom. 2013. Pico replication: A high availability framework for middleboxes. In *Proceedings of the 4th annual Symposium on Cloud Computing*. ACM, New York, 1–15.
[24] M. Schöller, N. Khan, and et al. 2015. *Network Function Virtualisation (NFV); Resiliency Requirements. GS NFV-REL 001 V1.1.1.* Technical Report. ETSI.
[25] V. Sekar, N. Egi, S. Ratnasamy, M. K. Reiter, and G. Shi. 2012. Design and implementation of a consolidated middlebox architecture. In *NSDI)*. USENIX, San Jose, 323–336.
[26] S. Sharma, A. Engelmann, A. Jukan, and A. Gumaste. 2020. VNF availability and SFC sizing model for service provider networks. *IEEE Access* 8 (2020), 119768–119784.
[27] J. Sherry et al. 2012. Making middleboxes someone else's problem: Network processing as a cloud service. *ACM SIGCOMM Computer Communication Review* 42, 4 (2012), 13–24.
[28] J. Sherry, P. X. Gao, S. Basu, A. Panda, A. Krishnamurthy, C. Maciocco, M. Manesh, J. Martins, S. Ratnasamy, L. Rizzo, et al. 2015. Rollback-recovery for middleboxes. In *ACM SIGCOMM Computer Communication Review*. ACM, London, 227–240.
[29] R. C Turchetti and E. P. Duarte. 2015. Implementation of a failure detector based on network function virtualization. In *2015 IEEE International Conference on Dependable Systems and Networks Workshops*. IEEE, 19–25.
[30] G. Venâncio et al. 2019. Nfv-rbcast: Enabling the network to offer reliable and ordered broadcast services. In *2019 9th Latin-American Symposium on Dependable Computing (LADC)*. IEEE, 1–10.
[31] G. Venâncio et al. 2021. Beyond VNFM: Filling the gaps of the ETSI VNF manager to fully support VNF life cycle operations. *International Journal of Network Management* 31, 5 (2021), e2068.
[32] G. Venâncio et al. 2021. VNF-Consensus: A virtual network function for maintaining a consistent distributed software-defined network control plane. *International Journal of Network Management* 31, 3 (2021), e2124.