Highly Available Virtual Network Functions and Services Based on Checkpointing/Restore

Giovanni Venâncio,
Elias P. Duarte Jr.

Department of Informatics,
Federal University of Parana (UFPR),
Curitiba, PR, Brazil
E-mail: {giovanni,elias}@inf.ufpr.br

Abstract: Network Function Virtualization (NFV) technology has the potential to have a deep impact on how networks are built and managed. However, in order to achieve its full potential, it is necessary to guarantee the required dependability, and in particular the availability of VNFs (Virtualized Network Functions) and SFCs (Service Function Chains). This work presents NHAM: NFV High Availability Module for the NFV-MANO (NFV Management and Orchestration) reference model. NHAM allows the creation and management of highly-available virtual network services consisting of both stateless and stateful VNFs and SFCs. The architecture provides multiple recovery mechanisms that differ in terms of cost and latency. The solution does not require any modifications of the source code of VNFs/SFCs to make them highly-available. The strategy is based on VNF checkpoint/restore together with SFC buffer management. A prototype was implemented and experimental results are presented showing that carrier grade availability levels can be achieved.

Keywords: Network Function Virtualization, Virtualized Network Functions, High Availability, Fault Tolerance.

Reference to this paper should be made as follows: Venâncio, G. and Duarte Jr., Elias P. (2023) 'Building Highly Available Virtual Network Functions and Services with Checkpointing/Restore', *International Journal of Critical Computer-Based Systems*, Vol. x, No. y, pp.xxx–xxx.

Biographical notes: Giovanni Venâncio received his Ph.D. in Computer Science from Federal University of Paraná (2023) under the supervision of Prof. Elias P. Duarte Jr. Giovanni holds an MSc (2018) in Computer Science and a Computer Science degree (2016) at the same institution. His research interests include Network Function Virtualization, High Availability, and Fault-Tolerant Distributed Systems.

Elias P. Duarte Jr. is a Professor at Federal University of Parana (UFPR), Curitiba, Brazil, where he is the leader of the Computer Networks and Distributed Systems Lab (LaRSis). Prof. Duarte has published more than 180 peer-reviewed papers, and supervised more than 130 graduate/undergradute students. He has served on editorial boards and as chair of several conferences and workshops in his fields of interest. Research interests include Computer Networks and Distributed Systems, their Dependability Management, and Algorithms. He is a member of the Brazilian Computer Society and a Senior Member of the IEEE.

1 Introduction

Virtualization technology represents the most promising solution to the "Internet ossification" issue caused by the unanticipated growth that has taken place since the design of decades-old Internet protocols. With virtualization, the network becomes programmable, facilitating its evolution along multiple directions. Network Function Virtualization (NFV) is one of the essential technologies enabling the replacement of hardware-based middleboxes by software running on off-the-shelf hardware [1]. Virtual Network Functions (VNFs) are used to implement individual network services, which can be combined to form complex Service Function Chains (SFCs) consisting of multiple VNFs connected in a predefined order [2, 3, 4]. Thanks to the availability of NFV technology, network services that were previously accessible only from a limited number of vendors can now be downloaded from Internet marketplaces [5]. The adoption of NFV technology has brought significant benefits in terms of network flexibility and management. To standardize the execution and management of NFV-based services and ensure interoperability of various VNFs, the European Telecommunications Standards Institute (ETSI) has proposed the NFV-MANO architecture [6].

Although network services executed as virtualized software offer several advantages, it is undeniable that they are more susceptible to failures than traditional specialized hardware alternatives [7]. The transition from hardware devices to virtualized platforms brings several challenges regarding dependability [8, 9]. Factors such as the integration complexity of multiple software systems in different layers, the interoperability of hardware and software components provided by different vendors, and the limited experience in operating virtualized network environments are some of the challenges that make it difficult to ensure the dependability of NFV-based networks.

Proprietary hardware-based middleboxes, on the other hand, are generally designed with strict resilience goals, similar to the standards defined by carrier-grade systems. The term “carrier-grade availability” refers to the reliability levels that telecommunication carriers and service providers offer for their network services and infrastructure, such as voice communication, data transmission, and internet connectivity. Ensuring carrier-grade availability (at five nines, 99.999%, which corresponds to less than five minutes of downtime per year) is critical to the widespread adoption of NFV technology. The ETSI has established several resiliency requirements for services running in virtualized environments [10, 7].

Several proposals have been put forward to increase the availability of network functions in virtualized environments [11, 12, 13]. However, these solutions come with limitations, such as the use of particular technologies or the need to modify VNF code. Some proposals do not include all the mechanisms necessary to guarantee end-to-end availability. Challenges are compounded by the fact that most network functions are stateful, requiring detailed function state management. Additionally, none of the existing solutions fully comply with the NFV-MANO reference architecture established by the ETSI [6].

In this work we present a novel high availability architecture for NFV-based services, encompassing both stateful Virtualized Network Functions (VNFs) and Service Function Chains (SFCs). The architecture, known as NHAM (NFV High Availability Module), has been integrated as a module into the NFV-MANO reference architecture, aligning with the specifications put forth by the ETSI. NHAM adopts a virtualization-centric approach, allowing any VNF or SFC instantiated on the NFV platform to seamlessly inherit the high availability and resiliency attributes.

NHAM's operations are twofold: it manages the internal state of VNFs, and performs fault management through a plethora of mechanisms that guarantee high availability. To monitor and control the internal state of VNFs, NHAM leverages checkpoint/restore-based techniques, thereby ensuring that after a VNF failure, its internal state can be recovered, retaining its previous state. NHAM also offers four different resiliency mechanisms that can be used to configure and update VNF replicas. These resiliency mechanisms differ in terms of computational resources and recovery time, enabling different types of VNFs with varying availability requirements to recover from failures.

NHAM's implementation-agnostic design ensures that any NFV-based service can achieve high availability without any modification of the VNF source code. The virtualized nature of VNFs enables checkpoints to be taken by saving the network function instance, providing a generic method to preserve the service state without requiring VNF code alterations.

Moreover, NHAM addresses the availability of stateful SFCs and puts forth a strategy to build resilient SFCs. This strategy combines checkpointing with buffer management, enabling the synchronization of the traffic processed by each VNF with its corresponding checkpoints. As a result, NHAM guarantees end-to-end service recovery that is complete and correct, allowing it to tolerate multiple VNF failures and prevent packet losses and duplications due to failures.

To assess the performance and availability of NFV-based services with NHAM's support, a prototype was implemented, and experiments were conducted. We demonstrate that depending on the strategy and parameters employed, carrier-grade availability can be achieved. This work is is an extended version of the LADC'2022 paper [14].

The remaining sections of this work are organized as follows. In Section 2, an overview of NFV and the NFV-MANO architecture, including SFCs, is presented. Section 3 describes the NHAM architecture, and Section 4 outlines the SFC fault-tolerance strategy. Section 5 presents the implementation and experiments, and Section 6 discusses related work. Finally, Section 7 concludes the paper.

2 Virtualized Network Functions & Services: An Overview

Network Function Virtualization (NFV) has been proposed as a software-based alternative for the implementation of network middleboxes, such as firewalls, Network Address Translation (NAT) devices, Intrusion Detection Systems (IDS), among others. Traditionally, middleboxes have been available as specialized hardware [15], which can be challenging to manage and troubleshoot [16]. These services represent a significant portion of a network's capital expenditures (CAPEX) and operational expenses (OPEX) [17]. NFV technology has been also proposed as the means to deploy general COIN (COmputing In the Network) services within the network [18]. NFV reduces costs, improves flexibility, and simplifies the design, development, and management of network services [1]. There are also other advantages, such as reduced energy and physical space requirements [19].

The European Telecommunications Standards Institute (ETSI) has promoted the development of the NFV-MANO (NFV Management and Orchestration) reference architecture [6]. This architecture enables virtual functions and services from different developers to interoperate seamlessly, and includes modules for VNF control and orchestration, as well as lifecycle and resource management. Additionally, NFV-MANO defines communication interfaces and provides abstractions for the resources necessary

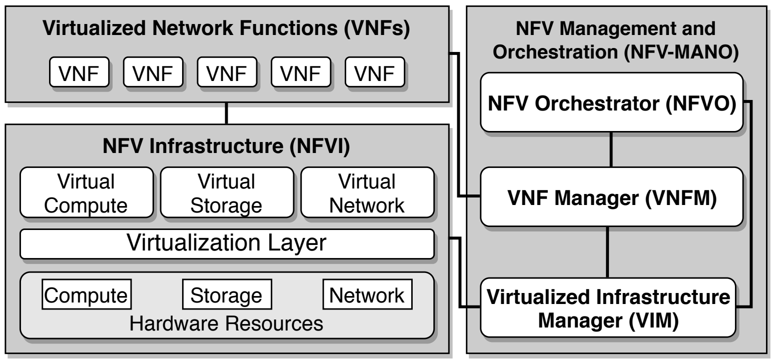

Figure 1 The ETSI NFV-MANO architecture.

to execute VNFs [20]. The NFV-MANO architecture, along with the NFVI (NFV Infrastructure) and the VNFs themselves, are depicted in Figure 1.

The NFVI encompasses the virtualized infrastructure where the VNFs are instantiated, managed, and executed. This infrastructure comprises physical storage, network, and computational resources, which are abstracted into virtual resources through a virtualization layer. The virtualization layer is made up of a hypervisor that creates and manages virtualized devices, such as Virtual Machines (VMs) and containers, providing isolation for each VNF to operate independently. In Figure 1, the VNFs symbolize the instances that execute on the NFVI.

NFV-MANO is composed of three main modules. The first module is the NFV Orchestrator (NFVO), which facilitates the composition of VNFs on SFCs (Service Function Chains) [21, 22]. The NFVO is also responsible for managing the SFCs lifecycle and VNFs resources. The second module is the VNF Manager (VNFM), which is responsible for VNF lifecycle management, including VNF instantiation, deletion, configuration, and auto-scaling [23]. To perform its functions, the VNFM utilizes the VNF Descriptor (VNFD), a template that specifies the operational and deployment requirements for each VNF. The third module is the Virtualized Infrastructure Manager (VIM), which controls and manages the computing resources of the NFVI, including the creation, deletion, and reconfiguration of virtual devices.

Regarding VNF availability, the ETSI has defined several resiliency requirements for NFV platforms and environments [24, 25]. Specifically, an NFV platform must support the resiliency of VNFs of different types provided by various vendors. Different levels of resiliency may be defined because different VNFs have different requirements. Additionally, to ensure high availability, an NFV platform must provide a comprehensive fault management system that can detect and help recover from VNF failures. Finally, an NFV platform must guarantee that stateful VNFs retain their internal state in case of failure.

Despite the fact that many NFV platforms are fully compliant with the NFV-MANO architecture, none of them offers the complete set of functionalities required to ensure end-to-end availability for VNFs and SFCs. The aim of the present work is to bridge this gap by proposing a high availability NFV architecture that integrates with the NFV-MANO reference model.

Although VNFs perform specific functions, they can be integrated into complex network services called SFCs. An SFC comprises multiple VNFs connected in a predefined order through which traffic is routed [2, 26, 27]. According to the Internet Engineering Task Force

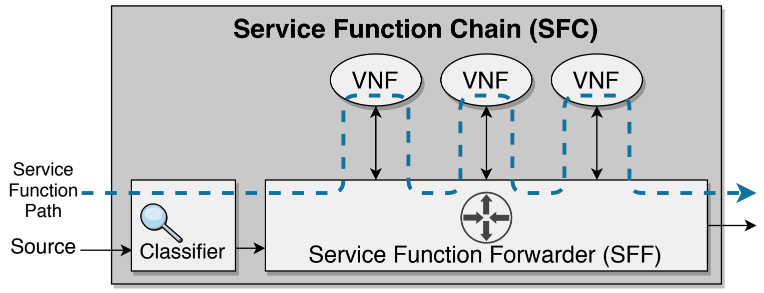
Service Function Chain (SFC)
VNF
VNF
VNF
Service Function Path
Source
Classifier
Service Function Forwarder (SFF)

Figure 2 The IETF architecture for Service Function Chains.

(IETF), the architecture of an SFC (as shown in Figure 2) comprises Classifiers, Service Function Forwarders (SFFs), and the VNFs themselves, which are briefly described below.

When network traffic enters the SFC, it first reaches the Classifier which applies predefined policies to determine the appropriate Service Function Path (SFP) to forward the traffic. These policies may consider several parameters such as source and destination IP addresses, ports, and protocols (e.g., TCP, UDP) among others. Once the Classifier selects the appropriate SFP, the traffic is encapsulated and forwarded to the corresponding SFP. As an SFC can have multiple SFPs, the header of the encapsulated traffic includes an identifier that specifies the selected SFP.

The Service Function Forwarder (SFF) has the task of transmitting packets from the Classifier to one or more network functions in a predetermined sequence. It accomplishes this by utilizing the information included by the Classifier in the SFC header. Once a Virtual Network Function (VNF) has processed incoming traffic, it sends the processed packets back to a SFF. Subsequently, the SFF forwards the traffic to the next VNF in the SFP, and this process repeats until all VNFs have processed the traffic. Finally, upon receiving the traffic from the last VNF in the SFP, the SFF removes the header from the packets and delivers the traffic to its ultimate destination.

To summarize, SFCs are a way of composing multiple VNFs to provide end-to-end network services. They enable the flexible and dynamic chaining of functions, allowing operators to create new services on demand. SFCs also provide an abstraction layer between the service provider and the underlying network infrastructure, making it possible to optimize network traffic by steering it through specific paths. However, ensuring high availability for SFCs can be challenging, especially in complex environments with many VNFs and SFCs. In the next section, we propose a high availability architecture that builds upon the NFV-MANO reference model and provides mechanisms to guarantee service continuity in the presence of component failures or network disruptions.

3 NHAM: A High Availability NFV Architecture

NHAM (NFV High Availability Module) is an architecture designed to ensure high availability for NFV. It provides strategies for building resilient VNFs and SFCs, including failure detection and recovery. NHAM is capable of handling heterogeneous functions and services from different providers. In a highly available SFC, the system continues to operate correctly even after faults occur, such as when one or more VNFs crash. As the recovery time decreases, the availability of the service increases. Detecting and reacting to failures quickly is essential to minimize downtime. However, redundancy alone or simply re-instantiating failed functions is not sufficient to solve the problem [28, 29]. Since most VNFs are stateful, their internal state changes according to the processed packets and the execution flow of the function. Hence, preserving the VNF state after recovery is crucial.

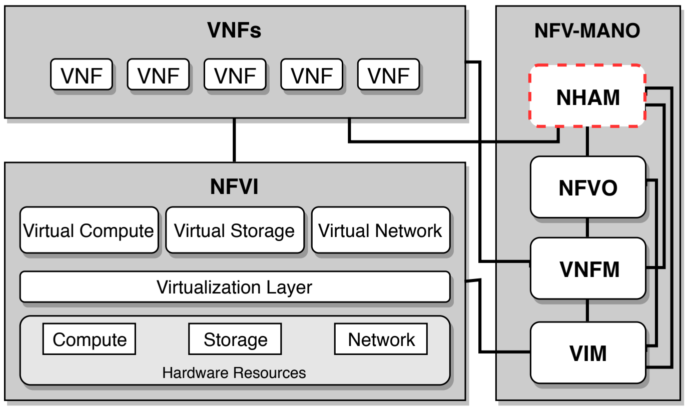

Figure 3 NHAM within the NFV-MANO architecture.

NHAM is a high availability solution for stateful and stateless NFV-based services. NHAM was designed as a module of the NFV-MANO architecture, and communicates with the other modules of the NFV-MANO architecture, as shown in Figure 3. 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; the strategy for resilient SFCs is described in the next section.

3.1 NHAM: The Architecture

NHAM is composed of two main components, which are shown in Figure 4: the Fault Management System (FMS) and the VNF State Manager (VSM). These components are

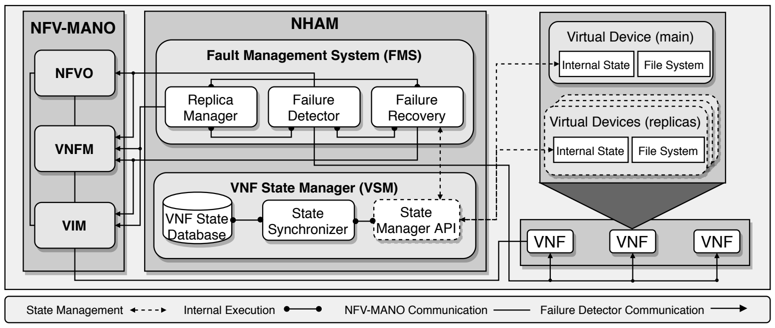

Figure 4 NHAM: Architecture.

described next. The FMS includes functionalities for failure detection and recovery, which are critical for ensuring high availability in the context of NFV. Failure detection involves monitoring VNF instances and identifying crashes [30]. NHAM employs two mechanisms for failure detection. The Failure Detector (FD) module uses a polling strategy, where messages are sent periodically to the VNFs being monitored, and acknowledgements are expected to 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 feature provided by several hypervisors. If a VNF is suspected of having crashed, the FD immediately adds it to a list of suspects and sends a notification message to the VNFM.

Furthermore, the FMS incorporates a Replica Manager that offers a range of resiliency options which are described in Subsection 3.3 to manage faults and recovery of Virtual Network Functions (VNFs). Depending on the specific availability requirements of a given VNF instance, one of four resiliency mechanisms can be selected. Additionally, the FMS assumes the responsibility of VNF recovery. NHAM has interfaces with the VIM, VNFM, and NFVO as detailed in Subsection 3.3.

To preserve the internal state of a VNF after recovery and ensure the correct recovery of stateful VNFs, NHAM employs the VNF State Manager (VSM) component. The VSM includes a State Synchronizer, an API for handling the internal state of VNFs, and a database responsible for storing the VNF states. A detailed description of the VSM is in the next subsection.

As mentioned above, NHAM communicates with other NFV-MANO modules, including the NFVO, VNFM, and VIM, to perform various tasks related to the lifecycle of virtualized services. During the recovery of a VNF, NHAM requests the VNFM to create new VNFs, as an example of NHAM-MANO interaction. Additionally, NHAM can reconfigure SFCs through the NFVO.

3.2 Stateful VNF Management

The VSM component is responsible for the recovery of stateful VNFs and is based on checkpoint/restore [31]. Since VNFs run on virtual devices, which can be either virtual machines or containers, capturing the VNF state without modifying the VNF source code is perfectly feasible and represents a very attractive option. To achieve this, checkpoints containing a representation of the system state are periodically captured and saved in non-volatile memory. In the event of a failure, the system can be restored to the most recent checkpoint, ensuring the correct recovery of the VNF.

The state of a VNF can be classified as either external or internal [25]. The external state includes static information that either does not change or changes infrequently over time, such as firewall/IDS rules and NAT port mapping tables. Recovering the external state is relatively easy once the VNF has recovered.

The internal state of a VNF, on the other hand, includes information that is updated as packets are processed and the function executes. Memory mapping, TCP connections, and cache contents are examples of internal state information. The primary challenge in managing VNF state is to preserve and ensure the consistency of the internal state, especially as VNFs fail and recover.

The VSM component of NHAM is responsible for recovering stateful VNFs after a failure, and it achieves this through the State Synchronizer. The State Synchronizer captures internal state information and saves VNF checkpoints, which are representations of the system state at a particular point in time. To do this, the State Synchronizer employs an agent that periodically collects internal state information from each VNF.

NHAM defines an API for VNF state management, which consists of two main operations: **export_vnf_state** and **import_vnf_state**. These operations are used to save and restore the state of a VNF respectively and are described below.

export_vnf_state: This NHAM operation saves a checkpoint of a specific VNF by momentarily pausing the virtual device to obtain the required state information for the checkpoint. After the information is obtained and the VNF execution is resumed, the checkpoint is sent to either the VNF State Database or directly to a replica, depending on the resiliency mechanism adopted. This operation is necessary to ensure that the internal state of a VNF is captured and can be restored in case of a failure.

import_vnf_state: This operation is used to restore the state of a VNF with a previously saved checkpoint. The operation requires two parameters: *(i) vnf*, which is the VNF instance identifier, and; *(ii) checkpoint*, which indicates from where the corresponding checkpoint has to be imported. To execute the 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. Once the operation is completed, 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. Stateless VNFs do not maintain any internal state that needs to be saved or recovered. These VNFs are designed to be stateless, as they perform a simple forwarding operation based on an incoming packet, without storing any information about the previous packets or connections. Therefore, when a failure occurs, these VNFs can be easily recovered by simply restarting them. Since they do not have any internal state that needs to be saved, the import and export VNF state operations are not needed for stateless VNFs.

3.3 NHAM: Resiliency & Recovery Mechanisms

The choice of resiliency strategy for ensuring high availability of VNFs is dependent on the specific requirements of each network function [24]. For instance, functions handling real-time traffic have more rigorous resiliency requirements compared to those handling best-effort traffic. Thus, the NFV platform should support various strategies that have different properties and costs.

NHAM features four resiliency mechanisms that rely on two different replication methods: Active-Standby and Active-Active, both of which are defined by an ETSI standard [25]. In the Active-Standby method, the VNF replica is already instantiated but is in standby mode, ready to take over in case the primary instance fails. On the other hand, in the Active-Active method, the replica has also been instantiated but is actively running and periodically updating its state, allowing for a more seamless transition in case of a failover. The choice between these two replication methods ultimately depends on the specific resiliency requirements of the network function in question.

The cost and recovery time of the different resiliency mechanisms vary, and the selection of a mechanism for a specific VNF depends on its features and requirements. Each mechanism employs a different recovery procedure. The resiliency mechanisms and their corresponding recovery procedures are described in detail below.

3.3.1 No Redundancy (0R)

The No Redundancy (0R) mechanism does not employ any type of redundancy. Therefore, in the event of a VNF failure, the only way to recover is to import the last checkpoint and restart the VNF execution from there. This implies that the service will suffer a downtime proportional to the time it takes to restore the VNF checkpoint, which can be significant for stateful VNFs with large state sizes. As shown in Figure 5, the State Synchronizer periodically takes (in the figure, label 1) and exports (label 2) checkpoints from the VNF to the VNF *State Database*.

After a failure occurs (3), the first step of the recovery process is to instantiate a new VNF (4), 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 from the VNF *State Database* (5) to the newly created VNF (6). 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.

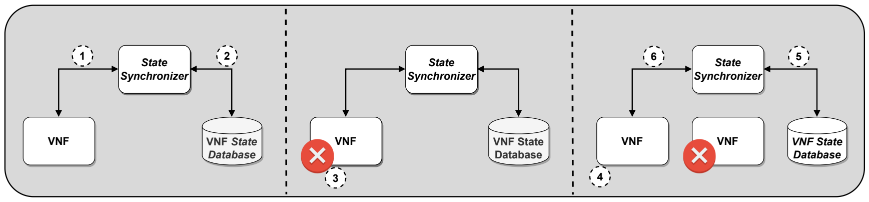

Figure 5 The 0R mechanism.

3.3.2 Primary Replica Active-Standby (1R-AS)

The Primary Replica Active-Standby (1R-AS) resiliency mechanism employs the Active-Standby method (i.e., *warm-standby*) with a replica that is instantiated but remains in a standby mode. As shown in Figure 6, the State Synchronizer exports the VNF checkpoints to the VNF State Database (in the figure, labels 1 and 2), exactly like the 0R mechanism does. However, unlike 0R, 1R-AS uses virtual resources to maintain a replica in standby mode, making it more expensive but with a shorter recovery time. In case of a failure (label 3), the replica is already created and the State Synchronizer imports the most recent checkpoint into the replica to update its internal state (4, 5, and 6). Once the internal state is updated, the replica becomes the primary VNF, and NHAM sends a request to NFV-MANO to update the required information. A new replica is then instantiated (7) and left in standby mode, ready to take over in case of a future failure.

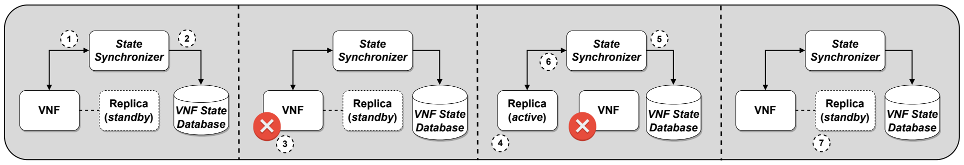

Figure 6 The 1R-AS mechanism.

3.3.3 Primary Replica Active-Active (1R-AA)

The Primary Replica Active-Active (1R-AA) resiliency mechanism is designed to handle the high availability of VNFs that require a lower recovery time than the previous mechanisms. As shown in Figure 7, in the 1R-AA mechanism, each VNF executes as two instances, a primary and a backup. The primary replica processes incoming traffic, while the backup replica remains in standby mode. The backup replica receives updates from the primary through the State Synchronizer (in the figure, labels 1 and 2).

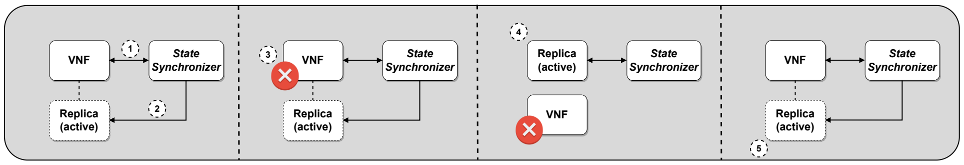

Figure 7 The 1R-AA mechanism.

In the event of a failure (label 3), the 1R-AA mechanism switches to the backup replica, which becomes the primary VNF (4). As the backup replica has received all updates from the primary replica, the failover is immediate. A new backup replica is then created, and the State Synchronizer imports the most recent checkpoint to that replica, updating its internal state (5). Finally, the reconfiguration process is executed.

The 1R-AA mechanism is the most expensive in terms of virtual resource consumption because it requires two replicas for each VNF to remain constantly updated, but provides the fastest recovery time.

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. As shown in Figure 8, the VNF is considered to be a member of a group of $1 + M$ replicas that are continuously synchronized by the State Synchronizer (in the figure, labels 1 and 2). Any of the replicas in the group can be accessed to obtain the service, and the states of the replicas are kept consistent. MR-AA is the most expensive of all mechanisms, as it requires the synchronization of all the M replicas, but it presents the shortest downtime in case of failures. No reconfiguration is required after a failure (label 3), as users can simply access any replica in the group (4). It is possible to specify a minimum and maximum number of replicas in the group. If the number of correct replicas falls below the minimum threshold, new replicas are created (5).

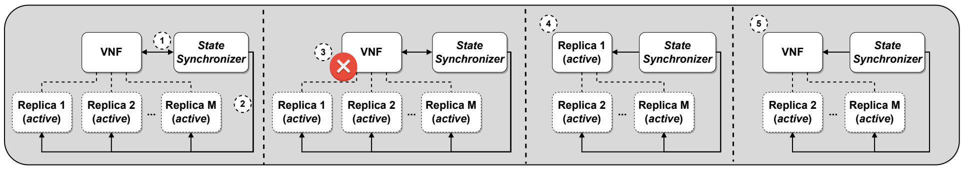

Figure 8 The MR-AA mechanism.

3.4 A Comparison of the Resiliency Mechanisms

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

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-AA mechanism has a shorter recovery time than the 1R-AS mechanism, as its backup replica is kept up-to-date. The MR-AA mechanism provides the shortest downtime, making it ideal for VNFs that require the highest level of resiliency. However, it is also the most expensive mechanism due to the need to synchronize the multiple replicas.

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

Therefore, the choice of resiliency mechanism depends on the specific requirements and priorities of each VNF. It is important to evaluate the trade-offs between cost, recovery time, and resiliency when selecting a mechanism. The 1R-AA and MR-AA strategies are more expensive in terms of resource utilization, but they provide the shortest recovery times, making them suitable for critical VNFs that require higher levels of availability. In addition, the MR-AA mechanism can provide even higher levels of availability, as it ensures that multiple replicas to be continuously synchronized, so that any of the replicas in the group can be accessed to obtain the service.

In addition to the higher cost of maintaining multiple synchronized replicas, the MR-AA mechanism also requires more complex synchronization algorithms and monitoring strategies to ensure the consistency of the states across all replicas. The State Synchronizer plays a crucial role in this mechanism and needs to keep track of all replicas in the group to guarantee the synchronization of the states. A monitoring strategy is needed to detect failures of any of the replicas and to take appropriate actions to replace replicas that have failed with new ones.

NHAM must also ensure the consistency of replica states in two specific situations: *(i)* when a VNF is falsely suspected to have failed, and *(ii)* when a VNF fails while the state is being updated. In the first case, NHAM performs the same recovery procedure as if the replica had actually failed, and a reconfiguration step is executed to replace the replica with a new instance or an existing one. In the second case, to prevent inconsistencies, the State Synchronizer halts the update process and rolls back all replicas to their previous state, using the most recently saved checkpoint. The failed VNF is eliminated, and the remaining replicas remain consistent.

4 Highly Available SFCs

This section presents the NHAM strategy for making SFCs fault-tolerant.

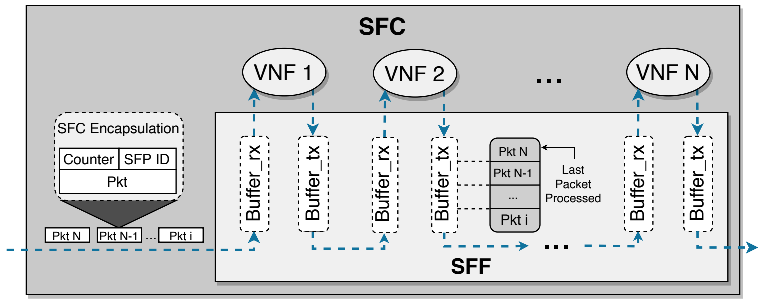

Figure 9 NHAM: high availability for SFCs.

4.1 Traffic Buffering

NHAM employs a high availability strategy that ensures the complete recovery of stateful SFCs, which are made up of one or more stateful VNFs, after any number of VNFs fail along the service chain. In most SFC implementations, a buffer precedes each VNF along the SFP, which is used by the SFF to store packets before delivering them to the VNF. NHAM, on the other hand, uses *two* buffers for each VNF in the chain, as illustrated in Figure 9. The first buffer, called *buffer_rx*, is located before the VNF in the chain and receives the traffic that needs to be delivered to the VNF. This buffer stores packets that have not yet been processed by the VNF. The second buffer, known as *buffer_tx*, is located after the VNF in the chain and receives the traffic output by the VNF. This buffer stores traffic that has already been processed by the VNF.

The data flow begins when the first packets from the SFF are stored in the *buffer_rx* of the first VNF, which will be processed by that VNF. Subsequently, the SFF forwards the packets from *buffer_rx* to the VNF. After processing the traffic, the VNF outputs the resulting packets into *buffer_tx*. The SFF then takes over and moves packets from *buffer_tx* of one VNF to the *buffer_rx* of the next VNF in the chain. This process is repeated for all other VNFs in the chain. When the final VNF processes the traffic and places the packets in the last *buffer_tx*, the SFF is responsible for delivering the traffic to the final destination correctly.

We make the assumption that the buffers, the SFF, and other MANO components do not fail. This is a practical assumption since the environment on which these SFCs operate must have been designed to be fault-tolerant to support highly available SFCs. Additionally, each VNF is assigned to a single SFC and is not shared by multiple SFCs. It is advisable to deploy VNFs and buffers on physically separated hardware. Furthermore, the recovery strategy assumes that all VNFs along the chain process traffic in First-In, First-Out (FIFO) order. Therefore, if two packets are sent to the VNF in a specific sequence and are not dropped by the VNF, they are output in the same order.

The Hold/Release approach is proposed to ensure the reliable recovery of an SFC after a failure. This method employs a blend of VNF checkpointing and buffer management, as detailed below.

4.2 The Hold/Release Strategy

The recovery of a stateful SFC consists of into two key components: *(i)* the recovery of each failed VNF, which involves the restoration of state for stateful VNFs (outlined in Section

3.2); and *(ii)* the retransmission of traffic that was lost as a result of VNF failures. This retransmission is accomplished using the Hold/Release approach, which is explained in detail in the following section *(ii)*.

Prior to storing a packet into the initial *buffer_rx* of the first VNF in the SFC, the SFF encapsulates each packet in order to enable routing along the SFP. Along with the explicit data used to identify the SFP [2], the SFF also incorporates a timestamp in the form of a counter as it encapsulates the packet. This timestamp works as a unique identifier for each sequential packet.

NHAM continuously monitors the VNFs, and as soon as it detects any VNF failure, it promptly alerts the SFF to change the state of the SFC to *recovering*. While the SFC remains in this state, traffic ceases to flow through the VNF until the VNF has fully recovered. In this *recovering* state, a VNF neither accepts packets from *buffer_rx* nor forwards packets to *buffer_tx*.

The Hold/Release strategy retains packets in *buffer_rx* until a checkpoint is taken. This is the "Hold" part of the Hold/Release strategy. In this way NHAM ensures that no packets are lost due to a VNF failures. Once a VNF checkpoint has been taken after it has processed a sequence of packets, we say that the checkpoint *includes* those packets. The SFF can then remove those packets from *buffer_rx*. This is the Release part of the Hold/Release strategy.

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. Conversely, if the VNF does not fail, the SFF waits for the checkpoint to be saved before proceeding. Once the checkpoint is saved, it can be inferred that the last packet in *buffer_tx* has been both processed by the VNF and included in the checkpoint. The SFF then removes from *buffer_rx* the packets up to and including that last packet.

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, thus it avoids sending duplicate packets along the SFC.

The Hold/Release strategy aims to ensure the consistent recovery of an SFC after a VNF failure by combining VNF checkpointing with buffer management. It involves temporarily retaining packets in *buffer_rx* until a VNF checkpoint is taken after those packets are processed. If a VNF fails before the checkpoint is taken, it is rolled back to the previous checkpoint, and all packets it had received from that point must be sent again and processed by the VNF. If the VNF does not fail, the SFF waits until the checkpoint is saved, and then removes all packets up to the last packet that was included in the checkpoint. The SFF also keeps track of the last packet delivered to the next VNF along the chain to avoid sending duplicate packets. Overall, the Hold/Release strategy allows for efficient recovery of an SFC by minimizing packet loss and avoiding duplicate packet delivery.

Consider a scenario where packets i, $i + 1$, and $i + 2$ are transmitted from *buffer_rx* to the VNF. The VNF processes only packet i when a checkpoint is initiated. Upon the completion of the checkpoint, the SFF verifies that packet i is the last packet in *buffer_tx* and thus removes all packets up to and including packet i from *buffer_rx*. Next, the VNF continues, and processes packets $i + 1$ and $i + 2$, which are then forwarded to the next

VNF through *buffer_tx*. The SFF maintains a record of the last packet in *buffer_tx*, which in this case is packet $i + 2$. If the VNF fails, packets $i + 1$ and $i + 2$ must be reprocessed by the VNF after it recovers since they were not included in the most recent checkpoint. Nonetheless, they have already been transmitted to the subsequent VNF in the chain. The SFF keeps track of that, and only forwards new packets from $i + 3$ along the SFC.

The VNF recovery process follows the adopted resiliency mechanism, as discussed in Section 3.3. Once the VNF is operational again, with its state restored based on the last checkpoint stored, 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 guarantees SFC recovery regardless of the number of VNFs that have failed, and works correctly even if multiple VNFs fail simultaneously, such as due to a power outage. Notably, the *buffer_rx* of a particular VNF is only cleared after a VNF checkpoint is saved, *and* the processed packet is placed in *buffer_tx*. Thus the traffic handled between each VNF checkpoint is not forfeited, and the integrity of the entire SFC is assured.

5 Implementation and Experimental Evaluation

NHAM was implemented as a prototype on an NFV platform compliant with the NFV-MANO reference model. The prototype was developed in Python, utilizing Docker containers [32]. For VNF state management, a REST API was created. VNF checkpoints were taken using CRIU (Checkpoint/Restore in Userspace) [33], containing the essential information to restore a non-operational VNF, such as the network function itself and some associated resources, like memory maps and the process tree. The VNFs employed in the experiments were packet forwarders.

One of the major advantages of NHAM is that it offers multiple alternatives that a user can choose to turn its VNFs fault-tolerant. A VNFD (VNF Descriptor) is employed to specify the desired strategy. Next we present an example VNFD. This VNFD specifies that NFV-MANO should instantiate the VNF on an Ubuntu container, employing 4 CPUs and 512 MB of RAM. The MR-AA resiliency mechanism is chosen, with three VNF active replicas (besides the primary). The replicas are kept updated according to the state of the primary. The checkpoint interval is defined to be of 250ms. The packets processed by the VNFs have 1024 bytes, while the captured state of each VNF is in average 512 KBytes (actually this may vary according to the specific VNF being processed).

```
topology_template:
  node_templates:
    capabilities:
      nfv_compute:
        properties:
          mem_size: 512 MB
          num_cpus: 4
    properties:
      type: container
      image: ubuntu
    resiliency:
      num_backups: 3
      cooldown: 250 ms
      vnf_level:
        type: MR-AA
```

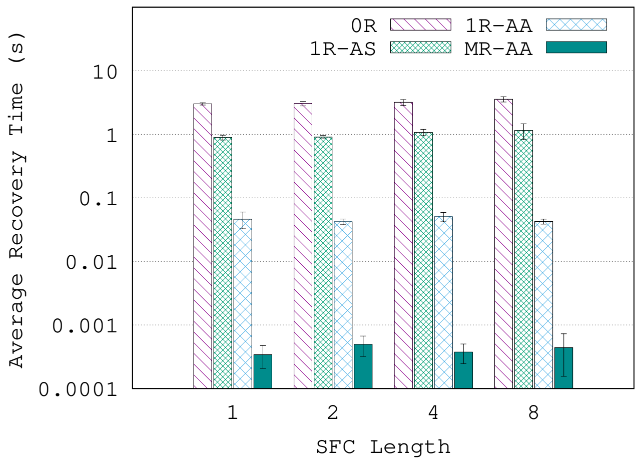

Figure 10 Average time a single VNF takes to recover.

The experiments were executed on an Intel Core i7 processor with 8 cores, 16 GB RAM, a 1 Gbps Ethernet NIC, and Linux Ubuntu 20.04. Each VNF comprises an Ubuntu server with 256 MB RAM and 1 CPU, while the MR-AA mechanism defaults to 3 replicas. NHAM does not need any kernel patches to operate. The first experiment set examines and compares the impact of the four distinct VNF resiliency mechanisms on SFC recovery time as the SFC's VNF count increases. The second experiment set evaluates the resource utilization of each recovery strategy, in terms of memory and CPU consumption. The third experiment set measures the impact of NHAM on throughput. Finally, the last experiment assesses the availability of NFV-based services supported by NHAM. Each experiment was repeated ten times, and the outcomes are averages presented with a 95% confidence interval.

5.1 Failure Recovery Time

The goal of the first set of experiments is to measure the time it takes for the SFC to recover from a failure. The downtime during a failure is particularly critical to improve the availability of virtual services. The failures were introduced by scripts that disconnect all connections from the VNFs, thereby triggering failure suspicions. The experiment measures the time it takes from the detection of a failure until the recovery process completes.

The first experiment in this set compares the four different resiliency mechanisms and measures the average recovery time after a single VNF in the SFC fails. The number of VNFs in the SFC is increased from 1 to 8, and the results are shown in Figure 10.

The results of the first set of experiments confirm the hypothesis that the 0R mechanism presents the longest recovery time, with up to 3.6 s of downtime for an SFC with 8 VNFs. This is due to the time it takes to instantiate a new VNF, which takes approximately 2.4 s on average, and the time needed for the VSM to restore the most recent checkpoint. On the other hand, the 1R-AS mechanism shows better results compared to the 0R mechanism, and this can be explained by the fact that a replica has already been instantiated and is in standby mode. As NHAM uses the Active-Standby method, only importing a checkpoint into the replica is necessary, resulting in a total recovery time of 1.14 s for an SFC with 8 VNFs.

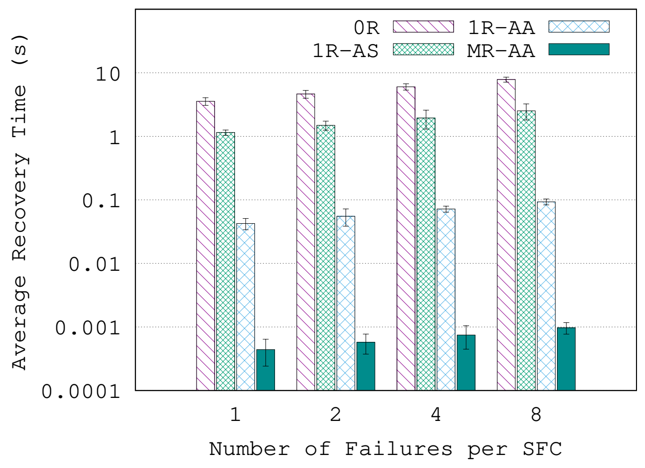

Figure 11 Time to recover multiple failures per SFC.

On the other hand, the 1R-AA and MR-AA mechanisms achieved the best results, with recovery times of 0.05 s and 0.0002 s, respectively. The results show that *(i)* NHAM maintains similar levels of performance even when the SFC length increases and; *(ii)* the recovery time remained unchanged, regardless of the SFC length, for all strategies.

In the experiment shown in Figure 11, the impact on recovery time due to multiple failures occurring simultaneously is evaluated. SFCs with 8 VNFs were used, and the number of failures per SFC ranged from 1 (single VNF failure) to 8 (failure of the entire SFC).

The impact of recovering multiple VNFs in parallel was evaluated in the next experiment. NHAM is designed to recover multiple VNFs simultaneously, as described in Section 4. The experiment measured the impact of increasing the number of VNFs that fail at the same time on the recovery time. For the 0R resiliency mechanism, the recovery time increased from 3.56 s for a single failure to 7.8 s for 8 failures, which is an increase of 2.1 times. On the other hand, for the 1R-AS mechanism, the difference in recovery time between a single failure and the failure of the entire SFC was smaller, increasing from 1.14 s to 2.52 s.

It is noteworthy that for the 1R-AA and MR-AA mechanisms, the impact of increasing the number of failures on the recovery time is minimal, since the time to recover from a single failure is already very low compared to the other strategies. For instance, the recovery time for 1R-AA varies from 0.004 s for one failure to 0.009 s for eight failures, while for MR-AA, it varies from 0.0004 s to 0.0009 s. Therefore, it can be concluded that all recovery mechanisms are scalable concerning the number of VNF failures.

5.2 Overhead

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 12 and 13 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%

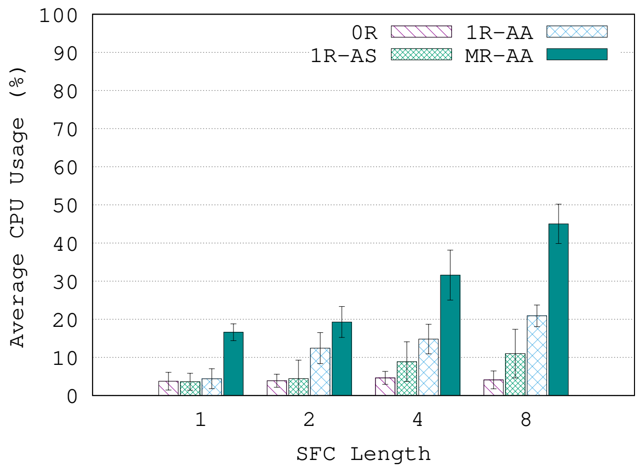

Figure 12 Overhead: CPU consumption.

(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.

In contrast, the mechanisms based on the Active-Active method, 1R-AA and MR-AA, present higher resource utilization due to the constant synchronization of the internal state of their replicas. The 1R-AA mechanism presents a CPU utilization of 21% and memory usage of 38% to synchronize up to 8 VNFs, while the MR-AA mechanism has similar memory usage but higher CPU usage, reaching up to 45% for a SFC with 8 VNFs. It is worth noting that the memory usage of the approaches based on the Active-Active method is significantly higher than the others, as they perform their operations in memory, avoiding non-volatile memory I/O overheads.

It is also important to highlight that the cost of the resiliency mechanisms can be adjusted according to the service provider's needs. For example, in scenarios where resource utilization is a critical factor, the 0R mechanism can be the best option, whereas, in scenarios where fast recovery is a priority, the MR-AA mechanism can be the most suitable.

5.3 Throughput

The next experiment evaluates the impact of NHAM's Hold/Release strategy on the throughput in two different scenarios using SFCs with four VNFs. In the first scenario, no failures occur, and the performance of each resiliency mechanism is compared to a baseline SFC that is not running NHAM. In the second scenario, failures occur every 30 seconds, and the impact of the Hold/Release strategy is evaluated.

In the absence of failures (Figure 14), the 0R and 1R-AS mechanisms showed similar throughput, as expected, as both mechanisms take checkpoints in the same way. These mechanisms reduced the throughput by approximately 11.5%, owing to the time taken to obtain, compress, and save checkpoints in non-volatile memory, which increases the downtime of the VNF.

The 1R-AA mechanism shows a decrease in throughput of only 4.7%. This mechanism operates in memory, as the internal state is transferred to an active replica, which results in a

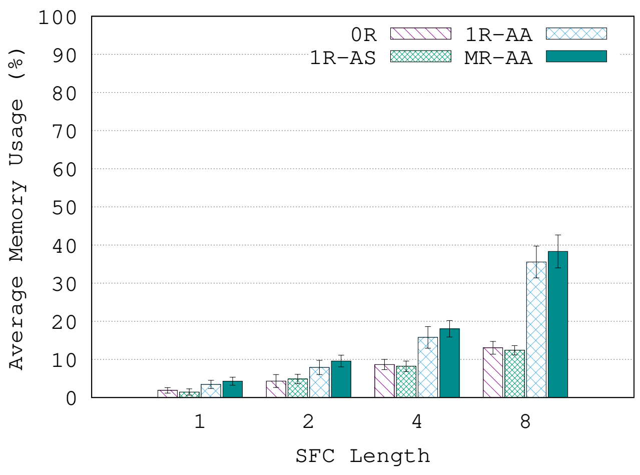

Figure 13 Overhead: memory consumption.

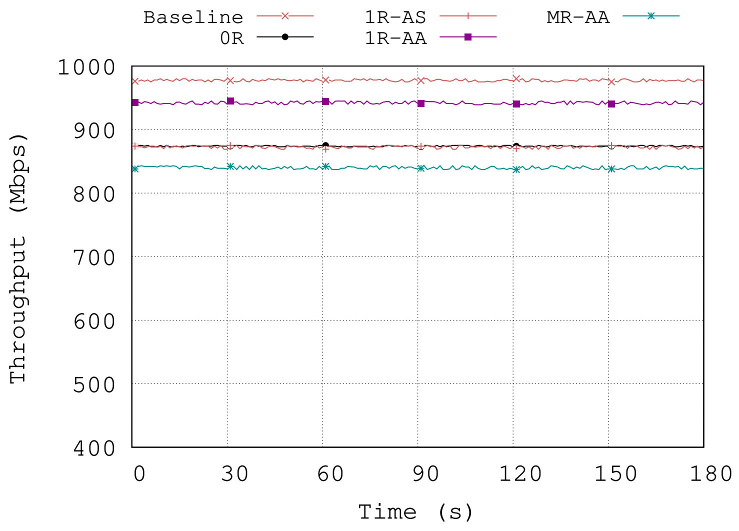

Figure 14 Throughput of fault-free SFCs.

significant improvement in throughput, as discussed in the previous section. In contrast, the MR-AA mechanism exhibits the greatest degradation of throughput. Although it has a very low recovery time, the throughput decreases by 14.1%. The MR-AA mechanism also runs in memory, but the processing required to ensure the consistency of the group of replicas for each VNF has a noticeable impact on throughput.

In the scenario in which VNF failures are injected every 30s, shown in Figure 15, the reduction in throughput is more significant for the 0R and 1R-AS mechanisms compared to the Active-Active-based methods. This is because both mechanisms (0R and 1R-AS) have longer recovery times. It is worth noting that even in this scenario, the throughput remains constant for the 1R-AA and MR-AA mechanisms.

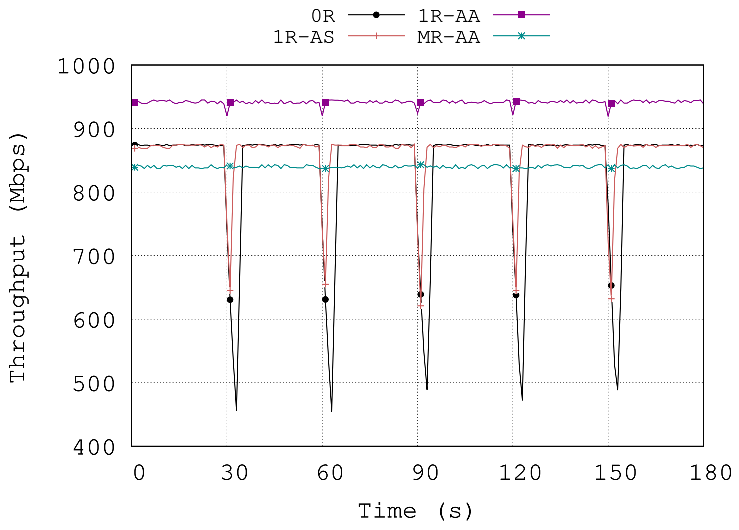

Figure 15 Throughput of SFCs with failures.

5.4 Availability

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

Table 2 SFC availability with a varying MTBF.

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

As anticipated, the 0R mechanism shows the lowest figures in terms of availability. However, it can still be useful for network functions that can tolerate longer recovery times. Even for tests with a higher MTBF (for example, one failure every 30 minutes), the 0R mechanism achieves only 99.3% availability. Similarly, the 1R-AS mechanism also fails to reach the levels of availability necessary to ensure carrier-grade availability of VNFs, although it performs better than 0R.

In the experiment, the 1R-AA mechanism presented superior performance in comparison with 0R and 1R-AS, even in the scenario with a higher probability of failures.

With an MTBF of 1 minute, VNFs using this mechanism achieved a 99.9% availability (three nines), while an MTBF of 10 minutes resulted in 99.99% availability (four nines). On the other hand, the MR-AA mechanism delivered the best results, with 99.999% availability (five nines) achieved in all cases.

The experiments presented in this section demonstrate that the NHAM framework is capable of providing high availability for VNFs, with the Active-Active mechanisms achieving the best results. The 1R-AA mechanism achieved an availability of 99.9% (three nines) even in the most failure-prone scenario, while the MR-AA mechanism reached an availability of 99.999% (five nines) in all cases. In contrast, the 0R and 1R-AS mechanisms had longer recovery times and lower availability rates, making them less suitable for carrier-grade NFV deployments. However, it is important to note that the availability of the cloud platform used to deploy the VNFs and SFCs also affects the overall availability of the system. In general, cloud platforms which are used to deploy NFV environments reach up to approximately 99.9% (three nines) of availability [7]. Despite this constraint, the NHAM framework provides a promising solution for improving the reliability and availability of NFV-based services.

6 Related Work

The REINFORCE framework [12] aims to enhance the resilience of Virtual Network Functions (VNFs) and Service Function Chains (SFCs) by replicating the network functions' states. Unlike the NHAM approach, which offers various resiliency mechanisms, REINFORCE adopts a single Active-Standby method. Moreover, the VNF developers are responsible for identifying the stateful VNF operations in REINFORCE. It is worth noting that the REINFORCE framework is not compliant with the NFV-MANO standard.

The FTC (Fault Tolerant Chain) approach [11] enhances the resiliency of Service Function Chains (SFCs) without relying on checkpointing or packet replay. Instead, FTC embeds VNF state information in packets that traverse the chain. Each VNF acts as a replica for its predecessor, eliminating the need for dedicated replicas. When a VNF fails, it is re-instantiated, and its state is retrieved from the succeeding VNF in the chain. It is important to note that FTC does not comply with the IETF SFC reference architecture, as it assumes that each VNF sends traffic directly to the next one. Additionally, it is not compliant with NFV-MANO. Exactly like in the REINFORCE framework, the VNF developer is responsible for indicating which operations cause state changes by modifying the VNF source code.

Remus [34] is a system designed to provide high availability for virtual machines, rather than NFV. It periodically saves checkpoints from one virtual machine onto a backup virtual machine. Therefore, in the event of a failure, the backup virtual machine can take over seamlessly. Remus also synchronizes checkpoints through buffering, where packets are temporarily stored in a buffer until the synchronization of a new state is complete. This approach is similar to NHAM's buffer management and checkpointing mechanism, but the contexts in which they are used differ.

The authors of a proposal centered on buffers, named Pico Replication (PR) [35], introduce a framework for enhancing the availability of middleboxes. Instead of preserving the internal state of the middleboxes, PR takes checkpoints on individual data flows, while the middlebox carries on processing other flows. Several adaptations are necessary to guarantee the high availability of middleboxes with PR, which involves modifications to both the kernel and SDN controller.

Decoupling the internal state of network functions from their processing is another proposed strategy for enhancing fault tolerance of stateful network functions, as described in [36] and [13]. This strategy involves saving the internal state to a distributed database. If a failure occurs, a new instance can retrieve the updated state from the database, which does introduce an overhead. The authors of both works claim that the solutions they propose adds a small latency per processed packet, as replicas are not pre-instantiated. However, if a VNF fails, a new instance must be created and its state updated, which inevitably impacts the overall recovery time. Furthermore, implementing both approaches requires extensive modifications to VNFs themselves.

In [37], a rollback-recovery approach is introduced, which proposes the FTMB (Fault-Tolerant Middlebox) system for preserving the state of middleboxes through "ordered logging" and "parallel release", described next. The ordered logging mechanism saves the necessary data to reproduce system entries in case of a failure, while parallel release is an algorithm that guarantees the correct reproduction of entries, considering the dependencies between packets. Although this solution presents low overhead when the system fails, implementing this approach requires modifications to the VNF source code, which could be considered a drawback.

The authors of [38] suggest a control plane architecture that reallocates traffic flows from failed to operational VNF instances, while maintaining the synchronization of VNF internal states. This control plane, known as OpenNF, handles the state and minimizes data loss by transferring flows through the controller. Moreover, OpenNF proposes a VNF state management API that is comparable to the one suggested for NHAM. However, the OpenNF API demands adjustments to the VNF source code and comes with performance concerns.

In [39] the authors propose the FTvNF framework for VNF fault tolerance. FTvNF tracks VNF states, and aims at reducing the state tracking costs. FTvNF relies on two instances of the protected VNF, called master and slave. In case of a failure, traffic is handled by the slave machine while the master is recovering. Packets arriving at a service chain first go through an sequencer that generates a unique identifier for each packet. The sequencer sends packets through the master VNFs. All the packets are stored in a reliable centralized logger that is assumed to be fault-tolerant, and remain there until FTvNF determines that all packets have been fully processed. After a master fails, the packets are handled by the corresponding slave. The major difference to NHAM is that FTvNF relies on a centralized fault-tolerant component, and presents a single recovery strategy.

A large number of existing fault-tolerant NFV solutions have a focus on VNF/SFC deployment. Nearly all adopt the most usual approach to enhance VNF fault tolerance: the deployment of backup VNFs as stand-by instances [40] or even standby SFCs [41]. Basically all those works explore the problem from an optimization point of view. Usually the problem is formulated and shown to be NP-hard, after that an heuristic is proposed, recent works have a focus on AI techniques [42]. Some of the solutions focus on performance besides availability, such as [43]. Virtually all those works present an evaluation of their proposed strategies using simulation, and treat VNFs and SFCs as abstractions with little relation to actual reference models.

Besides the aforementioned solutions, many NFV and cloud platforms, such as OpenStack [44] and OSM [45], provide some degree of fault tolerance. Nevertheless, these solutions are unable to ensure the uptime of stateful VNFs because they lack mechanisms to retain the virtual devices' internal states.

Table 3 shows a comparison of the main solutions for NFV reliability, according to the following characteristics and properties: (*i*) virtualization, which types of virtualization techniques are supported; (*ii*) NFV, whether the solution is NFV-MANO compatible; (*iii*) SFC, whether the solution supports fault-tolerant Service Function Chains or not; (*iv*) SFC IETF, in case the solution does support SFCs, whether it is compliant with the IETF SFC architecture; (*v*) whether function code modifications are required or not; (*vi*) redundancy methods supported e; (*vii*) the strategy adopted.

Table 3 A comparison of the main solutions for NFV reliability.

Solution	Virtualization	NFV	SFC	SFC IETF	Code Modifications	Redundancy Method	Strategy
NHAM	VM; Container	✓	✓	✓	✗	Active-Active; Active-Standby	Checkpoint/Restore
REINFORCE [12]	Container	✗	✓	✗	✓	Active-Standby	Checkpoint/Restore
FTC [11]	Click	✗	✓	✗	✓	Active-Active	Piggybacking
Remus [34]	VM	✗	✗	✗	✗	Active-Standby	Checkpoint/Restore
HA container [46]	Container	✗	✗	✗	✗	None	Checkpoint/Restore
PR [35]	VM	✗	✗	✗	✓	Active-Standby	Data flow checkpoint/restore
FreeFlow [47]	VM	✗	✗	✗	✓	Active-Standby	Internal state decoupling
CHC [13]	Container	✓	✓	✗	✓	None	Internal state decoupling
StatelessNF [36]	Container	✗	✗	✗	✓	None	Internal state decoupling
FTvNF [39]	VM	✓	✓	✗	✓	Active-Active	Logger; Packet replay
FTMB [37]	VM; Container	✓	✗	✗	✓	Active-Standby	Logger; Packet replay
PLOVER [48]	VM	✗	✗	✗	✗	Active-Active	SMR
OpenNF [38]	VM; Container	✓	✗	✗	✓	Active-Standby	Internal state decoupling
S6 [49]	VM; Container	✓	✗	✗	✗	Autoscaling	Distributed Shared Object

In comparison with NHAM, other solutions only provide partial support for ensuring high availability of stateful VNFs. In particular, three main drawbacks can be identified in those approaches. The first is the lack of support for multiple resiliency mechanisms. The second disadvantage is that several solutions require modifications to the VNFs' source code, which limits both the solution and the types of VNFs that can operate on the platform. Furthermore, having to modify code is also error-prone, which does have an impact on the reliability of functions. Finally, none of those solutions are fully compliant with the NFV-MANO reference architecture – *i.e.,* they do not execute within an NFV-MANO system. For example, it is often necessary to manually execute VNF lifecycle operations (*e.g.,* create a new VNF in case of a failure). This drawback raises interoperability concerns, making integration with other NFV systems a challenging task.

7 Conclusion

In this paper we proposed a strategy to build highly available stateful VNFs and SFCs based on the NFV-MANO reference model. NHAM does not require modifications to the source code of a VNF to make it fault-tolerant: NHAM is based on checkpoint/restore and offers four resiliency mechanisms that can be chosen based on the requirements of the different types of VNFs. Additionally, NHAM employs buffer management to allow the recovery of stateful SFCs. Even after multiple VNFs fail simultaneously, NHAM ensures complete and correct end-to-end service recovery. The proposed architecture was implemented as a prototype, and experimental results were conducted to evaluate its performance and availability. The results show that NHAM is an effective solution to improve the robustness of virtualized services, and it can achieve carrier-grade availability. Future work includes investigating strategies to improve fault prevention and prediction in the context of NFV.

8 Acknowledgments

This work was partially supported by the Brazilian Research Council (CNPq - Conselho Nacional de Desenvolvimento Científico e Tecnológico) grant 308959/2020-5; FAPESP/MCTIC/CGI grant 2021/06923-0; and the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.

References

[1] R. Mijumbi, J. Serrat, J-L. Gorricho, N. Bouten, F. De Turck, and R. Boutaba. Network function virtualization: State-of-the-art and research challenges. *IEEE Communications Surveys & Tutorials*, 18(1):236–262, 2016.

[2] J. Halpern and C. Pignataro. Service Function Chaining (SFC) Architecture. RFC 7665, IETF, October 2015.

[3] V. Garcia, L. Marcuzzo, G. Venâncio, L. Bondan, J. Nobre, A. Schaeffer-Filho, C. dos Santos, L. Z. Granville, and E. Duarte. An nsh-enabled architecture for virtualized network function platforms. In *International Conference on Advanced Information Networking and Applications*, pages 376–387. Springer, 2019.

[4] V. Fulber-Garcia, E. Duarte Jr, A. Huff, and C. dos Santos. Network service topology: Formalization, taxonomy and the custom specification model. *Computer Networks*, 178:107337, 2020.

[5] L. Bondan, M. Franco, L. Marcuzzo, G. Venancio, R. Santos, R. Pfitscher, E. Scheid, B. Stiller, F. De Turck, and E. P. Duarte. Fende: Marketplace-based distribution, execution, and life cycle management of vnfs. *IEEE Communications Magazine*, 57(1):13–19, 2019.

[6] J. Quittek, P. Bauskar, T. BenMeriem, A. Bennett, M. Besson, and et al. Network Functions Virtualisation (NFV); Management and Orchestration. GS NFV-MAN 001 V1.1.1. Technical report, ETSI, 2014.

[7] B. Han, V. Gopalakrishnan, G. Kathirvel, and A. Shaikh. On the resiliency of virtual network functions. *IEEE Communications Magazine*, 55(7):152–157, 2017.

[8] S. Sharma, A. Engelmann, A. Jukan, and A. Gumaste. Vnf availability and sfc sizing model for service provider networks. *IEEE Access*, 8:119768–119784, 2020.

[9] J. Li, W. Liang, M. Huang, and X. Jia. Reliability-aware network service provisioning in mobile edge-cloud networks. *IEEE Transactions on Parallel and Distributed Systems*, 31(7):1545–1558, 2020.

[10] C. Lac, R. Adams, and et al. Network Function Virtualisation (NFV); Reliability; Report on the resilience of NFV-MANO critical capabilities. GR NFV-REL 007 V1.1.1. Technical report, ETSI, 2017.

[11] M. Ghaznavi, E. Jalalpour, B. Wong, R. Boutaba, and A. Mashtizadeh. Fault tolerant service function chaining. In *SIGCOMM*, pages 198–210, Online, 2020. ACM.

[12] S. G. Kulkarni, G. Liu, KK Ramakrishnan, M. Arumaithurai, T. Wood, and X. Fu. Reinforce: Achieving efficient failure resiliency for network function virtualization based services. In *The 14th International Conference on Emerging Networking EXperiments and Technologies*, pages 41–53, 2018.

[13] J. Khalid and A. Akella. Correctness and performance for stateful chained network functions. In *The 16th NSDI*, pages 501–516, Boston, 2019. USENIX Association.

[14] G. Venâncio and E. P. Duarte. Nham: An nfv high availability architecture for building fault-tolerant stateful virtual functions and services. In *Proceedings of the 11th Latin-American Symposium on Dependable Computing (LADC)*, pages 35–44, 2022.

[15] V. Sekar, N. Egi, S. Ratnasamy, M. K. Reiter, and G. Shi. Design and implementation of a consolidated middlebox architecture. In *NSDI)*, pages 323–336, San Jose, 2012. USENIX.

[16] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, and V. Sekar. Making middleboxes someone else's problem: Network processing as a cloud service. *ACM SIGCOMM Computer Communication Review*, 42(4):13–24, 2012.

[17] D. Cotroneo, L. De Simone, A. K. Iannillo, A. Lanzaro, R. Natella, J. Fan, and W. Ping. Network function virtualization: Challenges and directions for reliability assurance. In *2014 IEEE international symposium on software reliability engineering workshops*, pages 37–42. IEEE, 2014.

[18] G. Venâncio, R. C Turchetti, and E. P. Duarte. Nfv-coin: Unleashing the power of in-network computing with virtualization technologies. *Journal of Internet Services and Applications*, 13(1):46–53, 2022.

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

[20] T. Tavares, L. Marcuzzo, V. Fulber-Garcia, G. Venâncio, M. Franco, L. Bondan, F. De Turck, L. Granville, E. P. Duarte, and C. Santos. Niep: Nfv infrastructure emulation platform. In *2018 IEEE 32nd International Conference on Advanced Information Networking and Applications (AINA)*, pages 173–180. IEEE, 2018.

[21] A. Huff, G. Venâncio, V. Garcia, and E. P. Duarte. 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)*, pages 19–24. IEEE, 2020.

[22] V. Fulber-Garcia, A. Huff, L. Marcuzzo, M. Luizelli, A. Schaeffer-Filho, L. Granville, C. dos Santos, and E. P Duarte. Customizable deployment of nfv services. *Journal of Network and Systems Management*, 29(3):36, 2021.

[23] G. Venâncio, V. Garcia, L. Marcuzzo, T. Tavares, M. Franco, L. Bondan, A. Schaeffer-Filho, C. R. Santos, L. Granville, and E. P. Duarte. Beyond vnfm: Filling the gaps of the etsi vnf manager to fully support vnf life cycle operations. *International Journal of Network Management*, 31(5):e2068, 2021.

[24] M. Schöller, N. Khan, and et al. Network Function Virtualisation (NFV); Resiliency Requirements. GS NFV-REL 001 V1.1.1. Technical report, ETSI, 2015.

[25] H. Nakamura, R. Adams, and et al. 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, 2016.

[26] A. Huff, G. Venâncio, L. Marcuzzo, V. Garcia, C. R. Santos, and Elias P Duarte. A holistic approach to define service chains using click-on-osv on different nfv platforms. In *2018 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6. IEEE, 2018.

[27] V. Garcia, G. Venâncio, E. P. Duarte, T. Tavares, L. Marcuzzo, C. R. Santos, M. Franco, L. Bondan, L. Granville, and A. Schaeffer-Filho. On the design and development of emulation platforms for nfv-based infrastructures. *International Journal of Grid and Utility Computing*, 11(2):230–242, 2020.

[28] G. Venâncio, R Turchetti, E. Camargo, and E. P. Duarte. Vnf-consensus: A virtual network function for maintaining a consistent distributed software-defined network control plane. *International Journal of Network Management*, 31(3):e2124, 2021.

[29] G. Venâncio, R. Turchetti, and E. P. Duarte. Nfv-rbcast: Enabling the network to offer reliable and ordered broadcast services. In *2019 9th Latin-American Symposium on Dependable Computing (LADC)*, pages 1–10. IEEE, 2019.

[30] R. C Turchetti and E. P. Duarte. Implementation of a failure detector based on network function virtualization. In *2015 IEEE International Conference on Dependable Systems and Networks Workshops*, pages 19–25. IEEE, 2015.

[31] E. N. Elnozahy, L. Alvisi, Y. M. Wang, and D. B. Johnson. A survey of rollback-recovery protocols in message-passing systems. *ACM Computing Surveys (CSUR)*, 34(3):375–408, 2002.

[32] D. Merkel. Docker: lightweight linux containers for consistent development and deployment. *Linux Journal*, 2014(239):2, 2014.

[33] CRIU. Checkpoint/Restore In Userspace, https://criu.org/, 2023.

[34] B. Cully, G. Lefebvre, D. Meyer, M. Feeley, N. Hutchinson, and A. Warfield. Remus: High availability via asynchronous virtual machine replication. In *NSDI*, pages 161–174, San Francisco, 2008. USENIX Association.

[35] S. Rajagopalan, D. Williams, and H. Jamjoom. Pico replication: A high availability framework for middleboxes. In *Proceedings of the 4th annual Symposium on Cloud Computing*, pages 1–15, New York, 2013. ACM.

[36] M. Kablan, A. Alsudais, E. Keller, and F. Le. Stateless network functions: Breaking the tight coupling of state and processing. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 97–112, Boston, 2017. USENIX Association.

[37] J. Sherry, P. X. Gao, S. Basu, A. Panda, A. Krishnamurthy, C. Maciocco, M. Manesh, J. Martins, S. Ratnasamy, and L. Rizzo. Rollback-recovery for middleboxes. In *ACM SIGCOMM Computer Communication Review*, pages 227–240, London, 2015. ACM.

[38] A. Gember-Jacobson, R. Viswanathan, C. Prakash, R. Grandl, J. Khalid, S. Das, and A. Akella. Opennf: Enabling innovation in network function control. In *ACM SIGCOMM Computer Communication Review*, pages 163–174, Chicago, 2014. ACM.

[39] Y. Harchol, D. Hay, and T. Orenstein. Ftvnf: Fault tolerant virtual network functions. In *Proceedings of the 2018 Symposium on Architectures for Networking and Communications Systems*, pages 141–147, 2018.

[40] B. Yang, Z. Xu, W. K. Chai, W. Liang, D. Tuncer, A. Galis, and G. Pavlou. Algorithms for fault-tolerant placement of stateful virtualized network functions. In *2018 IEEE International Conference on Communications (ICC)*, pages 1–7. IEEE, 2018.

[41] L. Wang, W. Mao, J. Zhao, and Y. Xu. Ddqp: A double deep q-learning approach to online fault-tolerant sfc placement. *IEEE Transactions on Network and Service Management*, 18(1):118–132, 2021.

[42] W. Mao, L. Wang, J. Zhao, and Y. Xu. Online fault-tolerant vnf chain placement: A deep reinforcement learning approach. In *2020 IFIP Networking Conference*, pages 163–171. IEEE, 2020.

[43] H. Hawilo, M. Jammal, and A. Shami. Network function virtualization-aware orchestrator for service function chaining placement in the cloud. *IEEE Journal on Selected Areas in Communications*, 37(3):643–655, 2019.

[44] OpenStack. OpenStack - open source software for creating private and public clouds, https://www.openstack.org/, 2023.

[45] ETSI. Open Source MANO, https://osm.etsi.org/, 2023.

[46] W. Li, A. Kanso, and A. Gherbi. Leveraging linux containers to achieve high availability for cloud services. In *2015 IEEE International Conference on Cloud Engineering*, pages 76–83. IEEE, 2015.

[47] S. Rajagopalan, D. Williams, H. Jamjoom, and A. Warfield. Split/merge: System support for elastic execution in virtual middleboxes. In *Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, pages 227–240, 2013.

[48] C. Wang, X. Chen, W. Jia, B. Li, H. Qiu, S. Zhao, and H. Cui. PLOVER: Fast, multi-core scalable virtual machine fault-tolerance. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI'18)*, pages 483–489, 2018.

[49] S. Woo, J. Sherry, S. Han, S. Moon, S. Ratnasamy, and S. Shenker. Elastic scaling of stateful network functions. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI'18)*, pages 299–312, 2018.