Dependable Virtual Network Services: An Architecture for Fault- and Intrusion-tolerant SFCs

Giovanni Venâncio*, Vinicius Fulber-Garcia*, José Flauzino*, Eduardo A. P. Alchieri†, Elias P. Duarte Jr.*
*Federal University of Paraná, UFPR, Curitiba, Brazil
E-mail: {jwvflauzino, giovanni, vinicius, elias}@inf.ufpr.br
†University of Brasília, UnB, Brasília, Brazil
E-mail: alchieri@unb.br

***Abstract*—A virtual network service can be implemented as a Service Function Chain (SFC) which is a composition of multiple Virtualized Network Functions (VNFs) forming the service topology. The IETF defines a standard architecture for SFCs, that includes traffic classification and forwarding elements. Considering that several network services implement functionalities that are critical for the correct operation of the network, the failure of an SFC can compromise the entire infrastructure, causing monetary losses or security issues. This work proposes the Fault-& Intrusion Tolerant SFC (FIT-SFC): an architecture to support secure and highly available virtual services. While most of previous related work only assumes crash faults, FIT-SFC uses replication strategies to also tolerate intrusions on any component of the SFC architecture, while still being fully compliant with the IETF reference model. A prototype was implemented and experimental results show that FIT-SFC presents acceptable overhead and fast failover, tolerating faults/intrusions on both stateless and stateful services.**

I. Introduction

The Network Functions Virtualization (NFV) paradigm has been changing the way networks are designed and operated [1], [2]. NFV provides an attractive alternative to replacing network functions that were traditionally implemented in specialized hardware (*i.e.,* middleboxes) with virtual software instances running on commodity hardware [1], called a Virtualized Network Function (VNF). VNFs can rely on a variety of virtualization technologies. By replacing hardware for software, the NFV paradigm has a significant impact on the way networks are managed, simplifying and improving the flexibility of the design, operation, and provisioning of network functions [3].

A virtual network service can be built through the composition of multiple VNFs, and is known as a Service Function Chain (SFC) [4]. In an SFC, VNFs are interconnected in a predefined order forming the service topology through which traffic is steered. An SFC can be composed of multiple functions [5] and deployed either within a single domain or across multiple domains [6]. The Internet Engineering Task Force (IETF) has proposed a reference architecture for SFCs [7] that enables the execution of virtualized services in the network, while supporting multiple types of topologies [8]. The SFC architecture proposed by the IETF consists primarily of traffic classification and forwarding elements that facilitate the creation and execution of arbitrary virtual network services. The main components of that architecture are: *(i)* Service Classifier (SC), responsible for receiving, classifying, and forwarding traffic to the appropriate SFC and; *(ii)* Service Function Forwarder (SFF), responsible for actually forwarding the packets between VNFs according to the order defined in the Service Function Path (SFP) – the order of VNFs through which the traffic must traverse.

Although NFV technology offers several advantages over hardware alternatives, it is undeniable that network services based on virtualization are more susceptible to failures, in particular as it relies on multiple layers of heterogeneous software [9], [10], [11]. Considering that several network services implement functionalities that are critical for the correct operation of the network, the failure of an SFC can compromise the entire infrastructure, leading to system downtime and monetary losses [12], [13]. In addition, failures of security services can lead to multiple security issues [14], affecting not only the underlying infrastructure (*i.e.,* the NFV network), but also the user services and applications. The unavoidable conclusion is that ensuring high availability and the security of NFV-based network services is essential to enable their adoption in large-scale production networks.

Despite the fact that virtualized network services have been extensively explored in the literature, several aspects of fault-tolerance have still been marginally addressed. Among the strategies for building fault-tolerant virtual services, some focus on detecting, tolerating, and eventually recovering from VNF crash faults [15], [16], [17], [18]. A crash fault specifies that the system element stops completely and looses all internal state – a crashed element does not perform any local computations nor send messages to other processes.

However, none of the existing solutions address failures of the components of the SFC architecture itself. Note that those failures can compromise the operation of the entire set of network services in execution. For instance, a failure of the forwarding component may prevent a packet flow from being forwarded between the VNFs within the SFC. This behavior could be exploited, for instance, to bypass a security function in the SFC or even deny access to the service itself. Therefore, it is critical to tolerate faults not only of VNFs but also of the the SFC architecture components. Furthermore, it is also essential to tolerate other types of faults besides crashes, in particular omission and Byzantine faults, described next. An component that suffers an omission fault only sends and/or

receives a subset of the messages it should. A component that suffers a Byzantine fault, behaves arbitrarily and thus can produce messages that differ from those it should according to its specification. This type of fault is typically used to represent a malicious behavior, *e.g.* due to an intrusion.

In this context, this work proposes Fault-& Intrusion Tolerant SFC (FIT-SFC): an architecture designed to support secure and highly available virtual services. FIT-SFC employs replication strategies to tolerate crash, omission, and Byzantine faults, while being fully compatible with the IETF reference architecture, enabling platform interoperability with different types of virtualized services. FIT-SFC addresses the failures of SFCs that process either stateless or stateful flows. For stateful flows, two types of consistency are employed. In the first type, the FIT-SFC ensures message consistency for each flow it processes. In the second type, multiple replicas must maintain the consistency across the different flows.

A prototype of the FIT-SFC architecture was implemented. The evaluation was performed with three virtualized services, each with different characteristics and requiring the support of different FIT-SFC features: *(i)* a Domain Name System (DNS) service; *(ii)* an authenticated DNS service; and *(iii)* a Load Balancer (LB) service. Experimental results evaluate the overhead of the proposed architecture and its fault- and intrusion tolerance capabilities.

The rest of this paper is organized as follows. Related work is described in Section II. Section III presents the FIT-SFC architecture. Section IV includes the implementation and experimental results. Finally, the conclusion follows in Section V.

II. Related Work

Fault Tolerant Chaining (FTC) [16] is a system that tolerats SFC crash faults. Based on the packet flow itself, FTC extracts information about the VNF state of each VNF. However, FTC does not create replicas of the VNFs – each VNF instance acts as a replica for its successor VNF along the SFC. Due to this strategy FTC is incompatible with the IETF SFC architecture, as it requires direct communication between VNF instances. Also, it is up to the network operator to specify which operations cause state changes, making it an error-prone approach.

Wang and others [15] employ standby VNF instances to provide crash-tolerant virtual services. For stateful VNFs, a mechanism is proposed to capture the state of an active VNF instance and replicate it to its standby replicas. The strategy can be considered computationally intensive, furthermore it is not compatible with the IETF SFC architecture.

Correct, High performance Chains (CHC) [17] is a framework to tolerate SFC faults that decouples VNF processing from its internal state, which is stored in a distributed database. CHC specifies a VNF architecture, and only tolerates crash faults of VNFs compliant with that architecture. In case of failures, a new VNF must be instantiated, and the corresponding state is retrieved from the database. This strategy has a significant impact on throughput and downtime.

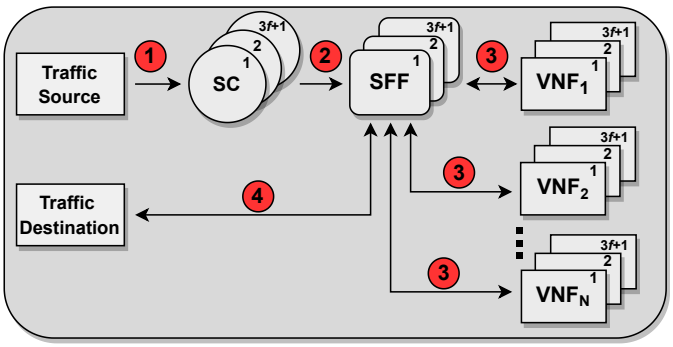

Fig. 1. FIT-SFC execution flow.

REINFORCE [19] is a fault-tolerant NFV framework that tolerates crash faults of VNFs and SFCs. REINFORCE maintains a dedicated replica for each VNF and replicates the internal state using checkpoints, applying techniques to improve service throughput. However, REINFORCE is not compatible with the IETF SFC architecture, making the integration with other systems a challenging task.

NHAM [20] is a high availability architecture for SFCs. The solution employs a checkpoint-based strategy for VNFs together with buffer management to tolerate crash faults of multiple stateless and stateful VNFs of an SFC. Although fully compliant with the ETSI and IETF architecture, NHAM does not tolerate faults of the components of the SFC itself.

Necklace [21] is an architecture for instantiating robust SFCs. Necklace uses distributed consensus to tolerate crash faults of both processes and communication links. The system is designed with performance concerns. Kong [18] also proposes a solution for ensuring the high availability of SFCs. In this case, the authors use backups for both links and VNF instances of an SFC.

The list of related work confirms that all the existing solutions for building fault-tolerant SFCs assume crash faults. Therefore, to the best of our knowledge, the present work can be considered as the first one to address *(i)* not only VNF faults but also faults of all components of the IETF SFC architecture and *(ii)* to tolerate not only crash failures but also omission and Byzantine failures.

III. Fault- and Intrusion-Tolerant SFCs

This section introduces the FIT-SFC architecture for secure and highly available virtual services. FIT-SFC was designed to extend the IETF SFC reference architecture to tolerate crash, omission, and Byzantine faults under the GST (Global Stabilization Time) partially synchronous timing model. Recall that a component that suffers a Byzantine fault presents arbitrary behavior, thus it usually represents an intrusion. Such a malicious component can make unauthorized modifications on packets traversing the SFC, inject spurious packets into the network, or attempt to leave replicated functions in an inconsistent state.

In order to tolerate faults and intrusions, the FIT-SFC strategy is based on replicating the key components of the

IETF SFC architecture: SC, SFF, and VNFs. It has been proved [22] that $3f+1$ replicas are required to tolerate up to f Byzantine faults. The FIT-SFC execution flow is shown in Figure 1. Despite the replicas, the packets follow the standard flow defined by the IETF SFC architecture. Network traffic is received by the Service Classifier (SC) from the client, which is the traffic source. An SFC also consists of a sequence of VNFs, with an SFF preceding and succeeding each VNF along the chain. Note that, even in a non-replicated SFC, there can be either a single SFF instance or multiple SFF instances.

The next subsections present Byzantine fault-tolerance, the operation of the architecture, and packet flow consistency.

A. Byzantine Fault-Tolerance

As mentioned above, considering that n is the total number of replicas of a given system, and f is the maximum number of replicas that may suffer a Byzantine fault, it is necessary to guarantee that $n \geq 3f+1$. To understand why, consider an element that receives packets from a replicated component. That element must decide which of the packets received from the replicas is correct. As omission faults are also tolerated, it is necessary to guarantee the correct decision after only $n-f$ packets are received, since f packets can be omitted. On the other hand, it is possible that no omission occurred, and among the $n-f$ packets received by the replicas, f were produced by malicious components. Therefore, among the $n-f$ packets, a majority of packets from correct replicas is required to ensure the correct decision. This is only possible if $(n-f)-f > f$, hence $n > 3f$ or $n \geq 3f+1$.

Furthermore, as shown in Figure 2, as packets traverse sequential sets of components, each component can only decide after receiving identical $2f+1$ copies of a packet. The figure actually shows a counter-example, demonstrating that $f+1$ identical copies of a packet are insufficient to guarantee Byzantine fault tolerance. The client is malicious and sends packet #1 to two correct replica and packet #2 to the other two correct replicas. The replicas of the second set of components decide after receiving $2f+1$ copies of the packet, of which a simple majority $f+1$ are identical. The result is that two replicas decide for #1 and two replicas decide for #2. In conclusion, a decision can only be made on a quorum of at least $2f+1$ identical packets.

B. FIT-SFC: Operation

Traffic enters a FIT-SFC from a client. Each client must be modified to incorporate a wrapper to forward packets. The wrapper serves two purposes: *(i)* it sends a copy of each packet to each of the $3f+1$ SC replicas, and *(ii)* it tags each packet sent with a local timestamp that serves as a unique packet identifier. The timestamp is implemented as a local counter for subsequent packets of a given SFC.

The traffic is then received by each of the service classifier replicas (called FIT-Classifier), each of which processes and forwards packets only once to each of the $3f+1$ SFF replicas. As shown in Figure 3, before the SFF forwards packets to the VNFs of the corresponding SFP, it must take into account that

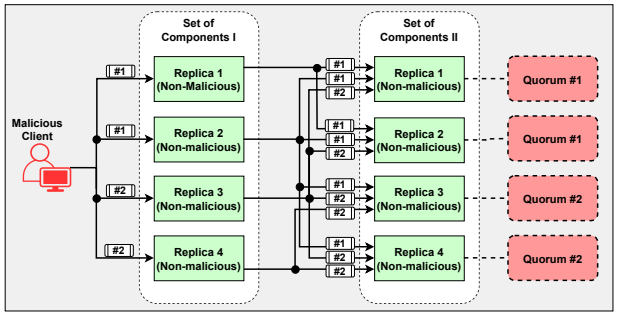

Fig. 2. Scenario with a malicious client.

there may be up to f faulty SC instances, or that the traffic source (client) is malicious. Recall that different versions of packets may be received due to Byzantine faults. Therefore, each SFF replica must perform a vote to select the correct packet to forward to the replicated VNF.

To enable this functionality, FIT-SFC adds a voter to each of the SFF replicas. Once the Voter receives $2f+1$ identical copies of a given packet, the SFF is able to process that packet and forward it to the next destination (*i.e.,* the VNFs). However, if there are no $2f+1$ matching packets, traffic forwarding and processing must be interrupted. In case a decision is reached, each SFF replica sends its copy of the packet to each of the VNF replicas. Similar to the SFFs, each VNF replica also has a Voter element that waits for $2f+1$ identical copies of a given packet before actually processing the packet. Similarly, if the VNF's Voter receives less than this number of identical copies, the flow must be interrupted.

After each VNF replica completes its execution, the Forwarder subcomponent sends a copy of each packet back to each SFF replica. At this point, each SFF replica performs a new vote to obtain a quorum of $2f+1$ packets before forwarding the correct version of the packet to the next set of VNF replicas according to the SFP. This process is repeated until the traffic has been processed by the last replicated VNF of the SFC. Finally, each SFF replica sends a copy of the packet to the final destination. Again, the packet is only sent to the final destination, if the voter confirms that there are $2f+1$ identical packets.

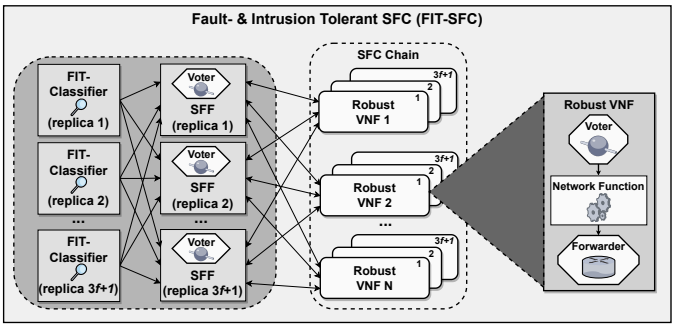

Fig. 3. FIT-SFC architecture.

C. Ensuring the Consistency of Replicated VNFs

The consistency of replicated VNFs must be guaranteed across the multiple replicas. There are different types of network functions, and the traffic they process may or may not modify their internal state. If the internal state of a VNF is decoupled from its processing, the VNF is stateless. On the other hand, if the internal state of the network function changes based on the packets it processes, we refer to it as stateful. In order to ensure service consistency, each replicated VNF executes a different protocol based on the expected type of consistency. In particular, this work classifies network services into three different types: *(i)* stateless, *(ii)* per-flow stateful, and *(iii)* globally stateful. Each service type is described below.

For stateless services, since they do not maintain internal state, it is sufficient to process incoming packets in the order they are received. Since the VNFs do not maintain state, the consistency of the service is not compromised if only some of the VNF replicas execute a particular packet.

On the other hand, per-flow stateful services maintain a state for each flow that passes through the service, which requires packets to be partially ordered. Therefore, packets of the same flow must be processed in the same order by all VNF replicas. In the FIT-SFC architecture, packets of a given flow are ordered by a local timestamp added by the client wrapper, which is used to define the processing order for the flow.

Finally, globally stateful network functions receive multiple flows from different sources, and their state is defined based on the processing of all packets of all flows received. A classic example of a globally stateful service is a load balancer. This type of service receives flows from multiple sources and distributes the load across multiple servers. When replicating this type of service, it is critical to ensure that all replicas process and send all packets in the same order to the same set of servers. In this case, each replica must process every packet from all flows in the same order, *i.e.* it requires total ordering of the packets of these flows. In this case, it is necessary to execute consensus to ensure consistency.

The FIT-SFC architecture implements the Practical Byzantine Fault Tolerance (PBFT) consensus algorithm [23]. Consensus is executed only for globally stateful services, ensuring total order among flows and guaranteeing that all VNF replicas execute the same sequence of packets. PBFT is implemented alongside the VNF Voter component, and the consensus execution occurs prior to the voting step, ensuring that packets are delivered to each Voter in the same order.

IV. Implementation and Experiments

A prototype of the FIT-SFC architecture was implemented in Python 3[1]. The implementation includes all the operational components of the IETF SFC architecture, and tolerates up to f faults of any component.

The SCs of the FIT-SFC architecture keep information about the SFF replicas, in addition to the traditional route and flow configuration information. Similarly, the SFFs also keeps the identifier of each SC and VNF replica of the instantiated services. This is necessary because the SFF communicates with the SC replica and VNF replicas of the instantiated services. Finally, the VNFs also communicate with SFF replicas, and keep information on how to reach them.

To verify the correct operation of the FIT-SFC architecture and to evaluate the cost for tolerating crashes and intrusions, three virtualized network services were implemented: a regular DNS service, an authenticated DNS service, and a Load Balancing (LB) service. These services are briefly described below.

The DNS SFC service consists of a single DNS VNF. Since the traffic processed by this VNF does not change its internal state, this service is classified as stateless. The authenticated DNS service has the same functionality as the previous service, but there is an additional authentication step. In this case, the SFC consists of one VNF that only performs authentication and another that performs the traditional DNS functionality. This service is classified as per-flow stateful because it must store user authentication information while the flow is active. Finally, the LB service implements a round-robin policy to distribute traffic. The SFC consists of a single VNF, and the service is globally stateful between flows, since each packet increments the packet counter at each instance of the load balancer. Consensus must be executed to ensure consistency across multiple replicas of this service.

The experiments were conducted on a virtual infrastructure created using the NIEP emulator [24] and executed on a machine based on an Intel Core I7 processor @ 2.5GHz, 16GB DDR4 RAM, and Ubuntu 16.04. The next subsections present an evaluation of the FIT-SFC architecture in terms of fault-tolerance (including crashes and intrusions) and performance (including overhead).

A. FIT-SFC: Fault-Tolerance

The purpose of first experiment is to demonstrate the ability of FIT-SFC to tolerate both crash and Byzantine faults. In this sense, we evaluate the latency of a virtualized network service employing the proposed architecture to tolerate one fault of each type of component (*i.e.*, $f = 1$). Thus, four replicas of each component of the FIT-SFC (SC, SFF, and VNF) are required – since $3f + 1 = 3 \cdot 1 + 1 = 4$ replicas, in this case.

We consider two cases. In the first case, components crash, thus stop responding. The second case employs man-in-the-middle attacks to modify the contents of packets exchanged between components, representing intrusions. For each case, four scenarios were considered: (A) all components are correct; (B) a replica of the SC component suffers a crash or an intrusion; (C) both a replica of the SC component and a replica of the SFF component crash or is attacked; and (D) one replica each of the SC, SFF, and VNF components suffers a crash or an intrusion. The network service employed in all scenarios of this subsection is the simplest one, consisting of a Domain Name System (DNS), so that each VNF replica corresponds to a replica of the same DNS server. Intrusion scenarios consist

[1]Source code and artifacts to reproduce the experiments are available at https://github.com/joseflauzino/FIT-SFC_NFV-SDN-2024

of a DNS spoofing attack, that modifies the response to the DNS query redirecting the client to a different IP than the one that actually corresponds to the requested domain name.

Figure 4 shows the average RTT (Round Trip Time) measured from the execution of 10K DNS queries, each carrying 512 bytes between a client and the DNS service. The confidence interval is of 95%. The results indicate that, as the number of crashed components increases (darker bars), the RTT decreases. The RTT includes the time for all FIT-SFC components to process packets. The RTT decreases from 17.9 ms in scenario A to 15.9 ms in scenario D – a reduction of 11.17%. This occurs because when a replica stops responding (*i.e.*, crashes), the network overhead decreases and fewer packets are received by each of the other replicas (*i.e.*, fewer messages are exchanged among the components).

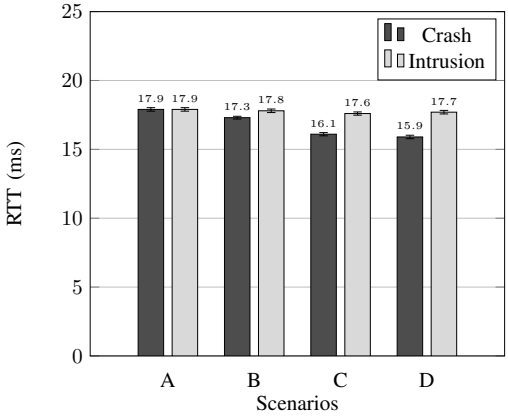

Fig. 4. RTT measured in scenarios with crash faults and intrusions.

On the other hand, in the case of intrusions, the RTT remains at about 17 ms, as the flow of packets from these replicas is not altered. Finally, it is important to stress that all faults were properly tolerated as expected in every scenario.

B. FIT-SFC: Performance

In this subsection, we evaluate the cost of using the FIT-SFC architecture in terms of the overhead it imposes in comparison with an SFC that does not tolerate faults. In particular, we examine the latency, i.e. the time required by the different components of the architecture to process packets of the different types of services. Furthermore, in every experiment we measured the proportion of time spent in each step of the execution flow: *(i)* SC: time spent between the client and the SCs; *(ii)* SFF: time spent between the SCs and the SFFs; *(iii)* VNF: time spent between the SFFs and the VNFs.

The experiment consists of two scenarios. In the first scenario we executed each service with an SFC in which no component is replicated, *i.e.* it does not tolerate crashes/intrusions ($f = 0$) and there is only one instance of each component. In the second scenario FIT-SFC was executed with four replicas of each component, *i.e.* tolerates one crash/intrusion for each component of the architecture ($f = 1$).

The first experiment evaluates the processing time for the DNS service (same of Subsection IV-A). The size of each request is 512 bytes. As shown in Figure 5, in the non-replicated scenario, a DNS request takes on average 79.08% less time to complete than in the scenario with four replicas. Of the total time, approximately 65% was spent on the SC, since it classifies the traffic and encapsulates each packet with the NSH (Network Service Header). The SFF and VNF each take about 17.5% of the total processing time. Those components take less time, because the SFF simply forwards packets and the VNF only handles local requests.

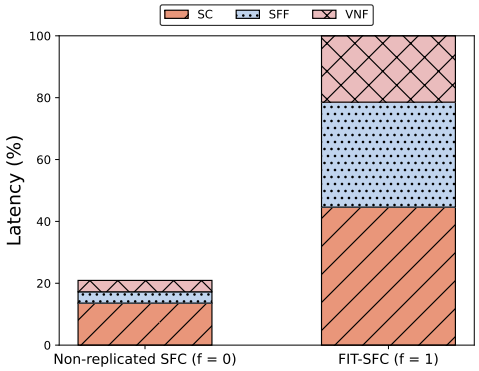

Fig. 5. DNS service latency: SFC vs. FIT-SFC.

Now consider the FIT-SFC scenario, which tolerates $f = 1$ fault/intrusion. In addition to taking about 4.7 times longer than the non-replicated SFC, in this scenario, most of the time was spent by the SC (44.61%), followed by the SFF and the VNF (33.94% and 21.45%, respectively). This is due to the additional message exchanges and comparisons that the Voter elements perform to ensure that at least $2f+1$ matching packets have been received.

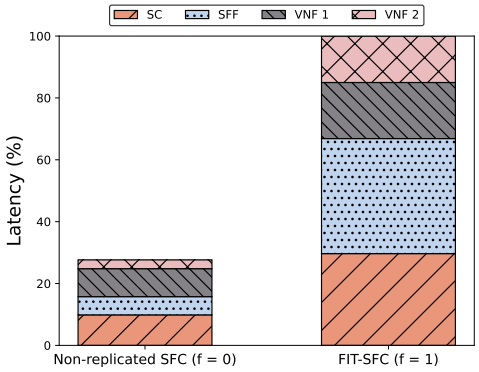

Fig. 6. Authenticated DNS service latency: SFC vs. FIT-SFC.

The second experiment in this subsection measures the latency for the authenticated DNS service. The size of each request is 512 bytes plus the authentication information. The difference from the service in the previous experiment is that the authenticated DNS consists of an SFC with two VNFs, instead of one. First of all, the results show that the total time for a DNS request in the non-replicated scenario increases by 69.41% when compared to the same scenario in the previous experiment. This is caused by the required additional steps for

the authentication process, including the extra communication between the two VNFs.

However, for the non-replicated scenario, Figure 6 indicates that a DNS request with authentication takes about 72.37% less time than in the FIT-SFC scenario with four replicas – a reduction of 6.71% compared to the regular DNS service (Figure 5). Figure 6 also shows that most of the time in the authenticated DNS service with four replicas was spent by the SFF (37.19%). The SC component consumed 29.67% of the time, while VNF 1 (the authenticator) and VNF 2 (DNS server) spent 18.13% and 15.01%, respectively.

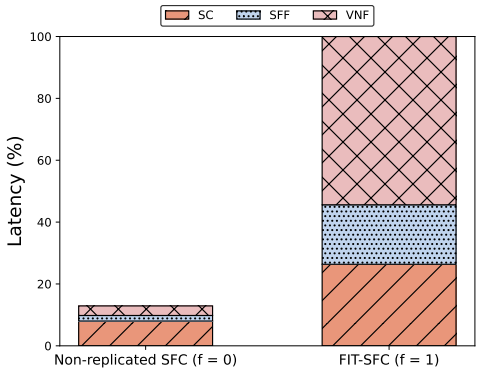

Fig. 7. Load balancing service latency: SFC vs. FIT-SFC.

Finally, the third experiment evaluates the latency of the LB service, as shown in Figure 7. For this experiment, the size of each request is 1024 bytes. As shown in Figure 7 the difference between the total time spent by the SFC without replicas and FIT-SFC with four replicas is of 87.1% - the highest among the experiments. The time spent by the VNF reached 54.40%, also the highest. Both can be explained by the overhead imposed by the consensus mechanism that is executed in order to ensure consistency across the multiple flows across the multiple replicas of the LB.

V. Conclusion

This work proposed FIT-SFC, a replication-based architecture that provides fault and intrusion tolerance for virtualized network services. FIT-SFC allows the internal components of the IETF SFC architecture to fail due to crashes, omissions, and intrusions. To tolerate up to f faults, each component is configured with $3f + 1$ replicas. Naturally, to make a virtual service fault and intrusion tolerant with the FIT-SFC approach comes at cost. Components are replicated, which also increases of the number of redundant packets traversing the network and that must be received and processed by all functions and components. A prototype was implemented, and experimental results confirm that, despite the increased cost imposed by the solution, services do become secure and highly available without requiring extensive modifications to their source code. Future work includes defining strategies to optimize overall processing time, such as batch processing of traffic, as well as conducting large-scale experiments, including multi-domain environments and tolerating more than one fault.

References

[1] R. Mijumbi *et al.*, "Network function virtualization: State-of-the-art and research challenges," *IEEE Communications surveys & tutorials*, vol. 18, no. 1, pp. 236–262, 2015.

[2] M. Chiosi *et al.*, "Network functions virtualisation: An introduction, benefits, enablers, challenges and call for action," in *SDN and OpenFlow World Congress*, 2012, pp. 22–24.

[3] B. Han *et al.*, "Network function virtualization: Challenges and opportunities for innovations," *IEEE Communications Magazine*, vol. 53, no. 2, pp. 90–97, 2015.

[4] P. Quinn *et al.*, "Problem Statement for Service Function Chaining - RFC 7498," Internet Engineering Task Force, Tech. Rep., 2015.

[5] V. Fulber-Garcia *et al.*, "Cusco: a customizable solution for nfv composition," in *International Conference on Advanced Information Networking and Applications*, 2020.

[6] A. Huff *et al.*, "Building multi-domain service function chains based on multiple nfv orchestrators," in *IEEE Conference on Network Function Virtualization and Software Defined Networks*, 2020.

[7] J. Halpern *et al.*, "Service Function Chaining (SFC) Architecture," IETF, RFC 7665, 2015.

[8] V. Fulber-Garcia *et al.*, "Network service topology: Formalization, taxonomy and the custom specification model," *Computer Networks*, vol. 178, p. 107337, 2020.

[9] S. Sharma *et al.*, "Vnf availability and sfc sizing model for service provider networks," *IEEE Access*, vol. 8, pp. 119 768–119 784, 2020.

[10] D. Cotroneo *et al.*, "How bad can a bug get? an empirical analysis of software failures in the openstack cloud computing platform," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2019, pp. 200–211.

[11] B. Han *et al.*, "On the resiliency of virtual network functions," *IEEE Communications Magazine*, vol. 55, no. 7, pp. 152–157, 2017.

[12] H. S. Gunawi *et al.*, "Why does the cloud stop computing? lessons from hundreds of service outages," in *Proceedings of the Seventh ACM Symposium on Cloud Computing*, 2016, pp. 1–16.

[13] P. Gill *et al.*, "Understanding network failures in data centers: measurement, analysis, and implications," in *Proceedings of the ACM SIGCOMM 2011 Conference*, 2011, pp. 350–361.

[14] M. Pattaranantakul *et al.*, "Nfv security survey: From use case driven threat analysis to state-of-the-art countermeasures," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 4, pp. 3330–3368, 2018.

[15] L. Wang *et al.*, "Ddqp: A double deep q-learning approach to online fault-tolerant sfc placement," *IEEE Transactions on Network and Service Management*, vol. 18, no. 1, pp. 118–132, 2021.

[16] M. Ghaznavi *et al.*, "Fault tolerant service function chaining," in *ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, 2020, pp. 198–210.

[17] J. Khalid and A. Akella, "Correctness and performance for stateful chained network functions," in *The 16th NSDI*. Boston: USENIX Association, 2019, pp. 501–516.

[18] J. Kong *et al.*, "Guaranteed-availability network function virtualization with network protection and vnf replication," in *Global Communications Conference*, 2017, pp. 1–6.

[19] S. G. Kulkarni *et al.*, "Reinforce: Achieving efficient failure resiliency for network function virtualization based services," in *Proceedings of the 14th International Conference on emerging Networking EXperiments and Technologies*. Heraklion: ACM, 2018, pp. 41–53.

[20] G. Venâncio and E. P. Duarte Jr, "NHAM: An nfv high availability architecture for building fault-tolerant stateful virtual functions and services," in *11th Latin-American Symposium on Dependable Computing (LADC)*. IEEE, 2022.

[21] F. Esposito *et al.*, "Necklace: An architecture for distributed and robust service function chains with guarantees," *IEEE Transactions on Network and Service Management*, vol. 18, no. 1, pp. 152–166, 2020.

[22] L. Lamport *et al.*, "The byzantine generals problem," in *Concurrency: the works of leslie lamport*. ACM, 2019, pp. 203–226.

[23] M. Castro and B. Liskov, "Practical byzantine fault tolerance and proactive recovery," *ACM Transactions on Computer Systems (TOCS)*, vol. 20, no. 4, pp. 398–461, 2002.

[24] T. N. Tavares *et al.*, "Niep: Nfv infrastructure emulation platform," in *International Conference on Advanced Information Networking and Applications*, 2018, pp. 173–180.