Serverless Computing for NFV: Is it Worth it? A Performance Comparison Analysis

Marco Savi, Alessandro Banfi, Alessandro Tundo, Michele Ciavotta
Department of Informatics, Systems and Communication, University of Milano-Bicocca, Milano, Italy
{marco.savi, alessandro.tundo, michele.ciavotta}@unimib.it, a.banfi8@campus.unimib.it

***Abstract*—Network Function Virtualization has established itself as one of the most important paradigms towards software-based networking. While today Virtual Network Functions (VNFs) are typically deployed in the form of serverful virtual-machine or container-based applications, the emergence of serverless computing opens the door to the possibility of implementing them as serverless functions, with benefits in terms of scalability and resource efficiency. This paper aims to assess whether this really makes sense or not, given the system-level overheads that a serverless computing platform naturally brings. We propose an open source platform designed to optimize the execution of network-intensive VNFs and we implement a data-plane and a control-plane function (i.e., NAT and DHCP responder, respectively) as serverless functions. We carry out extensive benchmarking of performance with their serverful counterparts, implemented as stand-alone containerized applications. Our experience makes it possible to conclude that serverless computing is beneficial for the execution of short-lived and request-based control-plane VNFs, while it should be avoided for the execution of data-plane traffic-intensive VNFs.**

***Index Terms*—Network Function Virtualization, Serverless Computing, Function-as-a-Service**

I. Introduction

In the recent years, *Network Function Virtualization* (NFV) has emerged as one of the most disrupting paradigms for the operation of telecommunication networks. By decoupling software from hardware and relying on virtualization technologies, network functions (e.g., Firewalls, Network Address Translators, etc.) can be implemented in the form of Virtual Network Functions (VNFs) that run on commodity hardware; this ensures that hardware and software can evolve independently, and is a fundamental step towards a widespread diffusion of the so-called *software-based networks* [1].

Serverless computing, in particular the *Function-as-a-Service* (FaaS) cloud service model [2], has recently gained momentum as it promises to reduce time-to-market, simplify deployment, and ensure better responsiveness/scalability to cloud-native applications. *Serverless functions*, which are typically stateless, are the deploy unit in this paradigm: their fine-grained nature allows for better Service Level Agreement (SLA) management, load balancing, and resource planning [3]. Moreover, serverless computing can help optimize operating costs by increasing resource utilization [4].

Given its nature, serverless computing appears to be a promising approach to streamline design, implementation, and operation of short-lived and event-driven VNFs [5]. In particular, as far as development is concerned, the FaaS paradigm promises the programmer to focus only on the implementation of single functions, freely choosing libraries and languages and without conforming to a particular framework. Moreover, compared to virtual machine and container based VNFs, most of the scalability, resource efficiency, server configuration, and management are delegated to the *serverless platform*, with little or no intervention by the system operator.

The benefits of this approach have been pointed out in some NFV-related contexts, such as mobile networking [6] [7] and edge computing [8]. However, most of the existing serverless platforms are ill-suited for the execution of network-intensive applications [9]. The main drawbacks are that *(i)* many VNFs need to maintain per-flow state, but serverless is primarily meant for stateless applications and *(ii)* function invocation on a per-packet basis is not suitable from a performance and cost perspective. Alternatives have thus been proposed, where function invocation occurs instead on a *per-flow* [9] and *per-flowlet*[1] [10] basis and where state is effectively managed [11].

Although these works provide evidence that the limitations of serverless platforms can be overcome, a question still remains unanswered: *is it really worth it*? In fact, serverless computing, although its unquestionable advantages, comes with some inherent performance degradation due to various system-level overheads [12]. With this work, we report our experience and try to give an answer to this question. For the first time, we provide a performance comparison between a *serverful* (i.e., a long-living, typically server-based application) and a *serverless* implementation of two well-known VNFs, namely Network Address Translator (NAT) and Dynamic Host Configuration Protocol (DHCP) responder. These VNFs are deeply different in their nature: NAT is a *data-plane* traffic-intensive application, while DHCP is a *control-plane* request-based function. Our goal is to assess and draw some generalized conclusions for both types of VNFs.

Unfortunately, we were unable to leverage existing serverless NFV platforms to carry out the assessment. In fact, they are either stand-alone systems [10] [11] or extensions of existing platforms (i.e., AWS Lambda) [9] that are not made available to the research community. Taking inspiration from [9] [10] [11], we propose a novel per-flowlet architecture, called *Network Function over Serverless* (*NFoS*), which is implemented as a modular extension of the open-source Apache OpenWhisk platform. NFoS is proved to be superior to Apache OpenWhisk in the execution of network-intensive

[1] *Flowlets* are separated burst of packets belonging to the same flow.

The final version of this record is available at http://doi.org/10.1109/PerComWorkshops53856.2022.9767495

VNFs, and our code is publicly released[2].

The remainder of this paper is organized as follows. Section II recalls the background and related work. Section III describes the NFoS architecture, while Section IV provides some details on the implementation of NAT and DHCP as serverless functions. Section V reports our performance comparison analysis and Section VI concludes the paper.

II. Background and Related Work

Literature contains some previous works adopting serverless computing and FaaS for the execution of VNFs. Ref. [9] is the first work pointing out the architectural limitations of canonical serverless computing platforms for the execution of network-intensive VNFs. The main limitations are the high per-flow execution cost and the complexity of per-flow state management. The authors propose a novel *per-flow* FaaS architecture, where all the packets belonging to the same network flow are handled by the same VNF instance. The benefits with respect to a *per-packet* architecture, where each packet triggers a function invocation and is thus naturally enabled by canonical serverless platforms, are also investigated: one major advantage is that the user can be billed on a per-flow basis, ensuring that she is not charged if the flow has not been completely processed. The authors implement such an architecture on AWS Lambda and successfully test it on two network-intensive VNFs, i.e., a Firewall and an Intrusion Detection System (IDS). Ref. [10] extends the work in [9] by proposing a *per-flowlet* architecture that enables *ephemeral statefulness*. Moreover, the architecture envisions the adoption of an external store to simplify state management even further. Ref. [11] proposes Serpens, a serverless platform for NFV that has the explicit goal of reducing the performance overhead of existing solutions. The main contribution is a novel architectural design where the lifecycle of a VNF instance and its state are decoupled. States are kept in executors' memory, which is shared among instances, and they can also be migrated to other executors. The authors extensively test the proposed platform; significant improvements in terms of latency and throughput are guaranteed against other serverless platforms. None of the proposed works focus however on a detailed performance comparison between serverless and serverful VNF implementations. We do so to understand in what cases the adoption of a per-flowlet serverless NFV architecture can be beneficial.

A couple of recent works focus on implementing serverless VNFs by exploiting existing platforms such as OpenFaaS and AWS Lambda. Ref. [13] proposes a *Caching System*, while Ref. [14] proposes the *attach* procedure of a *Mobility Management Entity* (MME) in a 4G network. In both cases, the serverless platform is adopted as is. Especially, in Ref. [14] the *attach* procedure of MME requires nine different function invocations, and a user would be billed for intermediate invocations also if the *attach* procedure was not completed (e.g., due to some packet loss). We also implement two serverless VNFs (i.e., NAT and DHCP responder), but they require only one function invocation for each processed flowlet.

[2]https://github.com/NFoSSystem

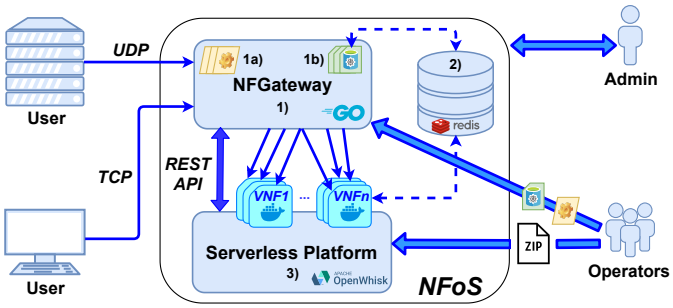


Fig. 1. NFoS System Architecture.

III. NFoS: Serverless NFV Platform Architecture

Figure 1 provides an overview of *NFoS*, our proposed per-flowlet serverless NFV platform. The main components, numbered as in the figure, are the following.

*1) **NFGateway**:* It is the core component of NFoS and it serves API Gateway functionalities. In particular, it is responsible for: *(i)* handling incoming requests from third-party non-HTTP clients, *(ii)* balancing incoming traffic among function instances exploiting a *Load Balancing Algorithm* that is function-state aware, and *(iii)* scaling in/out serverless functions according to workload. Also, it can employ an external *Database* for state management. In our prototype, NFGateway, implemented in Go language, embeds two main sub-components:

1a) Rule Engine: It is in charge of evaluating *Static* and *Dynamic Matching Rules* upon reception of a packet. *Static Matching Rules* allow NFGateway to understand by which VNF type the incoming packet should be processed, and they are inserted each time a new VNF type is registered. *Dynamic Matching Rules* are used to route the packet to the correct VNF instance (i.e., the one managing the flowlet to which the packet belongs). They are reactively loaded/unloaded at runtime on a per-flowlet basis. The per-flowlet workload allocation proved to mitigate the platform overheads that would occur if a *per-packet* allocation was adopted [9].

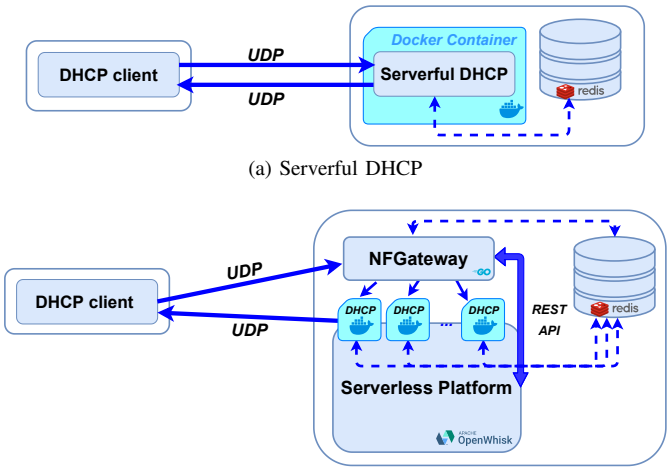
1b) State Management Modules: They embed all the operations executed by the NFGateway for *state management* of VNFs, which may use information stored in the Database. A new module is deployed in NFGateway every time a new VNF type has to be supported. They also include Database initialization and periodic management routines.

*2) **Database**:* It is used to store information for VNF state maintenance and can be accessed by both NFGateway and VNFs (if needed). It is implemented by means of an in-memory key-value store. In this work we employed Redis[3].

*3) **Serverless Platform**:* It is responsible for executing and managing serverless VNFs. It is accessed by other components, in particular the NFGateway, through the exposed *REST API*. In this work, we employed Apache OpenWhisk[4]

[3]https://redis.io

[4]https://openwhisk.apache.org/



(a) Serverful DHCP

(b) Serverless DHCP

Fig. 2. DHCP implementations and testing scenario. Analogous diagrams can be drawn for NAT, but access to the database is not required in this case.

as Serverless Platform. OpenWhisk exploits containerization (using Docker[5]) to manage each instance of registered VNFs.

A. Involved actors and deployment scenarios

Three main actors are expected to interact with NFoS:

1) Users: They send requests to and receive responses from the VNFs executed by NFoS. NFoS is seen as a black box by users, meaning that it takes as input packets/requests towards the VNFs and transparently returns the elaborated packet/response. A user is typically a host in the network (e.g., a smartphone that needs to have assigned an IP address by a DHCP service).

2) Admin: It is responsible of *(i)* the configuration and deployment of the NFoS platform, including the setup of network connectivity between its components and *(ii)* the monitoring of the system status after its start up. The admin is typically a network or a cloud provider.

3) Operators: Upon authentication, they upload to NFoS the binaries of serverless VNFs as ZIP archives, the related State Management Module and the Static Matching Rules for any specific VNF. An operator is typically a service provider.

NFoS can be adopted in different scenarios. Even though nothing prevents to deploy it in centralized cloud, the most appealing case appears to be its adoption in a federated computing environment at the edge [8] [15], where distributed NFoS platform instances are deployed in resource-constrained edge nodes to serve VNF execution requests from near users.

IV. Serverless VNFs: Design and Implementation

While serverless computing has been originally devoted to the execution of stateless applications, many existing VNFs are stateful and thus require state management. One of the main challenges is to re-architect existing stateful VNFs to make them working as serverless functions and executable by a serverless NFV platform (NFoS in this specific case).

We implemented two VNFs as serverless functions: a NAT and a DHCP responder. In both cases, we first implemented their core functionalities, in Go language, as *serverful* processes, which can be run as containerized stand-alone daemons (Fig. 2(a)); then, we re-engineered them to be executed as serverless functions, reusing code as much as possible (Fig. 2(b)). NAT and DHCP are very different VNFs. Network address translation must be executed for any packet coming from the specific local network(s) requiring NATting, for which some fields in the L3 header (IP address translation) and/or L4 header (port translation) must be re-written. For this reason, NAT is typically identified as a *data-plane* function. Other examples of data-plane VNFs are Firewall and IDS. Conversely, the assignment of an IP address to a user of a local network as guaranteed by a DHCP responder is a request-based *control-plane* operation that is rarely executed (per user). Other VNFs belonging to this type are DNS and MME. Such a fundamental difference between NAT and DHCP responder (and related more general VNF types) translates to distinct design and implementation choices, as detailed below.

1) NAT: We could not find any suitable open source serverful implementation to start with, thus we designed and developed a simple NAT for UDP connections from scratch. We made sure the implementation is compliant with IETF standards [16]. Every time a new NAT function is instantiated by OpenWhisk, the NAT State Management Module assigns to it a dedicated pool of available IP addresses and ports. These can then be used for translation of a feed of incoming UDP connections (i.e., flowlets). The mappings between UDP connections and a serving NAT instance are specified by the Dynamic Matching Rules at the NFGateway, while the association between IP addresses/ports is stored in an in-memory data structure within any VNF instance. An alternative and simpler design (in terms of state management) would be maintaining mappings between addresses/ports in the database, accessed by any NAT instance. However, this would imply a per-packet query to the database, which would cause an unacceptable high packet-processing latency.

2) DHCP: We started from an open-source serverful implementation[6] and we re-engineered it to be executable by OpenWhisk. The implementation is compliant with IETF standards [17]. Each flowlet is composed by a pair p = (*DHCP_Discover, DHCP_Request*) as sent by any host and its related responses (*DHCP_Offer, DHCP_Ack*), or by a *DHCP_Release* message to discard the IP address lease. In this case, given the small number of per-host function execution requests and looser latency requirements, the VNF implementation relies on the database for state management, where the host/IP address association (i.e., the DHCP table) is stored and can be accessed by both NFGateway and DHCP instances. Implementing DHCP required less effort than implementing NAT, due to the simpler procedures for state management.

V. Performance Evaluation

A. Testing environment, tools and configuration

1) Environment: We run our experiments on *CloudLab* [18], a cloud environment for research and education that enables

[5]https://www.docker.com/

[6]https://github.com/krolaw/dhcp4

TABLE I
VCPU AND MEMORY QUOTAS (TIERS) FOR INSTANTIATED FUNCTIONS.

Tier Name	vCPUs	Memory (MB)
t1	1	32
t2	2	64
t3	4	128
t4	6	256

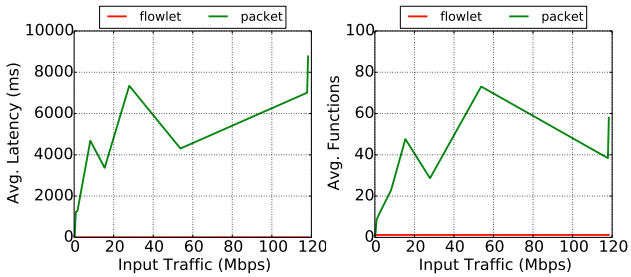
users to define their own experimental setup in terms of network topology, hardware and software equipment. The underlying physical nodes are provisioned with multiple CPUs, hundreds of GBs of memory, and gigabit network interfaces. The topology adopted in the experiments is composed by two dedicated VMs for both serverless and serverful VNF evaluation as shown in Fig. 2

2) Tools: Starting from the source code of *iPerf*[7] and *Netperf*[8] open source testing software, we developed two tools, named *nattester* and *dhcptester*, to test NAT and DHCP, respectively. The former includes the logic to send/receive flowlets of UDP packets at different rates, and collect statistics concerning sent/received packets and end-to-end latency. The latter acts as a DHCP client sending multiple DHCP requests. It is possible to specify the request rate, and the tool collects statistics on end-to-end latency and DHCP association errors. We adopted *Telegraf*[9] to retrieve CPU and memory utilization statistics on the system under test. Collected data is funneled into *InfluxDB*[10]. *Nattester* and *dhcptester* sample statistics on a per-packet basis, *Telegraf* collects statistics every two seconds.

3) Configuration: We defined four different *tiers* (*t1-t4*) for any VNF instance, each with different vCPU and memory requirement. The tiers specify the amount of resources assigned to the container running the VNF, both for serverless and serverful implementation (Table I). OpenWhisk imposes that functions cannot be equipped with less than 64MB of memory; this prevents to execute functions based on tier *t1*. However, we kept it in our evaluation of serverful VNFs as it is the smaller tier able to smoothly execute them. Each experiment lasted five minutes and statistics were collected and averaged within such a time frame.

B. Workload allocation strategies

Fig. 3 compares *per-flowlet* and *per-packet* workload allocation for NAT VNF execution (tier: *t2*). The plotted values refer to average statistics collected during one single experiment per input traffic. Results show that a per-packet workload allocation is clearly ineffective: the average packets' end-to-end latency is high (in the order of seconds) due to cold start, and tens of VNF instances are on average active. The platform is overloaded by requests for the activation of new functions and packet loss (not shown in the figure) is close to 100% even for low input rates (20 Mbps). Conversely, a per-flowlet allocation leads to an average end-to-end latency in the order of tens of milliseconds and the deployment of a very limited number of concurrent VNF instances. No packet loss is experienced in the considered input rate range. These results justify the design of an external component like NFGateway to extend OpenWhisk with per-flowlet allocation capabilities.



(a) (b)

Fig. 3. Comparison between *per-packet* and *per-flowlet* workload allocation in terms of average (a) latency and (b) number of instantiated functions, as a function of the input rate.

C. Performance comparison

1) NAT: Table II reports a comparison in terms of packet loss and average end-to-end latency between serverless and serverful NAT for the defined tiers and for input rates of 100 and 200 Mbps. Focusing on an input rate of 100 Mbps, we can see that negligible packet loss is experienced for both implementations, with marginally better performance (around 0.005%) in the case of the serverless VNF. This comes at the expense of a three-times higher latency, due to NFGateway and OpenWhisk related operations. Performance is comparable for all the considered tiers. Slightly different results are obtained for an input rate of 200 Mbps. For all the tiers, both in the case of serverless and serverful NAT, less than 1.5% packet loss is experienced. Although still manageable, this is higher than the loss experienced for 100 Mbps. Higher loss occurred for serverless NAT especially, and it is mainly related to overheads introduced by NFGateway, due to the higher number of packets that have to be processed and balanced among VNF instances in the time unit. This is also confirmed by the fact that packet loss is almost invariant to the tier for serverless NAT, while this is not true for serverful NAT, where more memory and computational resources reflect to lower packet loss. Moreover, as for 100 Mbps, latency is higher for the serverless implementation.

Figure 4 reports CPU and memory usage over time for the two implementations in the case of 100 Mbps input traffic. In both cases, only one vCPU is used for any tier, although more vCPUs could be potentially be assigned. For this reason, CPU usage (in percentage) refers to the used vCPU. Instead, memory usage is normalized to the memory assigned for each tier. In the case of serverless NAT, the graphs represent the average over all the containers running VNF instances, while for serverful NAT the graphs refer to the sole deployed container. Similar trends (not reported) are obtained in the case of 200 Mbps traffic but with higher experienced CPU and memory consumption. Concerning CPU consumption for serverless NAT (Fig. 4(a)), we can see that, in percentage, around 55% of the vCPU capacity is used for all the tiers. Pe-

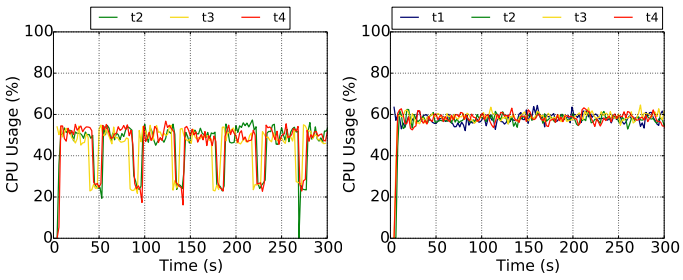
[7]https://iperf.fr/
[8]https://github.com/HewlettPackard/netperf
[9]https://www.influxdata.com/time-series-platform/telegraf/
[10]https://www.influxdata.com/products/influxdb/

TABLE II
SERVERLESS AND SERVERFUL NAT PERFORMANCE COMPARISON.

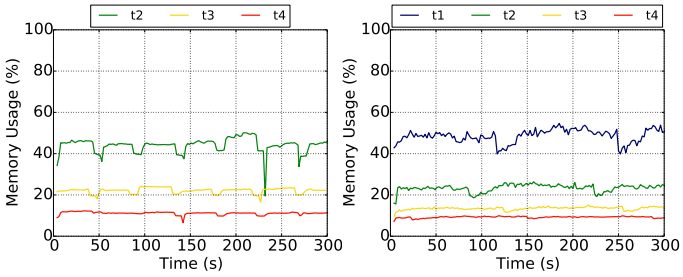
		Serverless NAT		Serverful NAT	
Input (Mbps)	Tier	Packet Loss (%)	Latency (ms)	Packet Loss (%)	Latency (ms)
100	t1	-	-	0.006	0.036
	t2	0.001	0.122	0.006	0.036
	t3	0.001	0.120	0.006	0.037
	t4	0.001	0.122	0.007	0.037
200	t1	-	-	1.047	0.284
	t2	1.316	0.571	0.037	0.212
	t3	1.182	0.525	0.007	0.253
	t4	1.327	0.607	0.000	0.183

TABLE III
SERVERLESS AND SERVERFUL DHCP PERFORMANCE COMPARISON.

		Serverless DHCP		Serverful DHCP	
Input (Req/s)	Tier	Errors (%)	Latency (ms)	Errors (%)	Latency (ms)
20	t1	-	-	0.000	2.000
	t2	0.000	0.998	0.000	2.007
	t3	0.000	0.996	0.000	2.002
	t4	0.000	1.001	0.000	2.020
100	t1	-	-	11.60	0.365
	t2	0.010	0.807	29.84	0.365
	t3	0.000	0.771	10.64	0.144
	t4	0.000	0.718	86.98	2.043

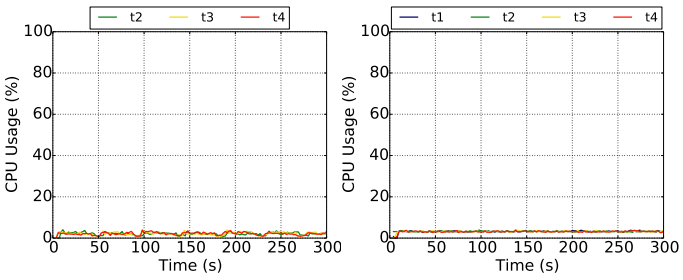


(a) Serverless NAT (b) Serverful NAT

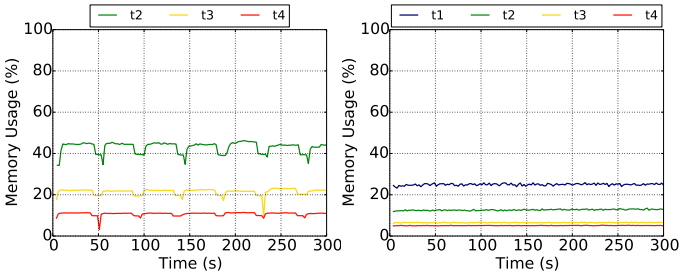


(c) Serverless NAT (d) Serverful NAT

Fig. 4. CPU and memory usage over time for serverless (a)(c) and serverful (b)(d) NAT (input: 100 Mbps).



(a) Serverless DHCP (b) Serverful DHCP



(c) Serverless DHCP (d) Serverful DHCP

Fig. 5. CPU and memory usage over time for serverless (a)(c) and serverful (b)(d) DHCP (input: 20 Req/s).

riodical tier-independent drops can also be noticed. These are due to the automatic deployment of new function instances to replace the expiring ones (function timeout). Temporary lower CPU consumption occurs right before such a replacement takes place as triggered by NFGateway. Regarding memory (Fig. 4(c)), its absolute consumption is around 30 MB for all the tiers. With respect to CPU consumption of serverful NAT (Fig. 4(c)) a slightly higher (but comparable) CPU usage is experienced (around 60%) with no drops, as no function replacement occurs. Concerning memory (Fig. 4(d)), lower consumption is experienced for the serverful implementation with respect to the serverless one (around 20 MB). The higher consumption for the latter case may result from the management overhead introduced by the OpenWhisk execution runtime wrapping the VNF implementation. The execution runtime serves several functionalities (i.e., init, activation and logging) to enable smooth platform operations, but that account for higher resource consumption.

Results allow us to conclude that implementing a NAT as a serverless function does not bring to real benefits for any of the considered metrics. Conversely, for high input rates NFGateway risks to become a system bottleneck and to introduce an unacceptable performance degradation.

2) DHCP: Table III reports a performance comparison between serverless and serverful DHCP in terms of association errors and average end-to-end latency for 20 and 100 DHCP Req/s. Both implementations do not incur association errors in the former case, with twice higher latency for the serverful DHCP. This is probably because load balancing, as performed by NFGateway, introduces a short latency in request processing that, as a side effect, reduces the number of concurrent accesses to the database. In the latter case, results are instead much different. Serverless DHCP implementation does not incur relevant errors, while the behavior of serverful DHCP becomes unstable with a high number of errors for all the tiers. This happens because NFoS is able to automatically scale up the serverless DHCP. In contrast, the serverful DHCP starts dropping messages when the amount of requests becomes not manageable by the VNF for lack of resources. Even though we are aware that forwarding 100 Req/s to a DHCP is a stress test, this is a situation that may occur in real scenarios (e.g., flash crowd). In this case, a serverful DHCP not well-dimensioned to handle such a peak of requests would not properly work. Moreover, latency is, on average, higher for serverless DHCP

than for serverless NAT, as an external database is used for state management. It is also important to note that the lower latency experienced by the serverful DHCP in the case of 100 Req/s with respect to 20 Req/s is due to the high error percentages. In fact, end-to-end latency can be computed only for successful requests, but it should be considered infinite (and not nil) in the case of errors.

Figure 5 reports CPU and memory usage over time for both implementations in the case of 20 Req/s as input, that is, when no association error is experienced. Also for DHCP, only one vCPU is used in the two cases, and memory usage is normalized as done for NAT. As a first consideration we can see that the average CPU usage, for both serverless (Fig. 5(a)) and serverful (Fig. 5(b)) DHCP is an order of magnitude lower than that of serverless (Fig. 4(a)) and serverful (Fig. 4(b)) NAT. This is a clear indication that NAT is a much more CPU-intensive function than DHCP. In absolute terms, as also happens for NAT, a marginally higher CPU consumption is experienced by serverful DHCP compared to serverless DHCP (around 1% higher). Concerning memory consumption of serverless DHCP (Fig. 5(c)), it is very similar to that of serverless NAT (Fig. 4(c)), with an average consumption of around 30 MB. Memory consumption of the serverful implementation (Fig. 4(d)) is on average much lower (around 12 MB), but in both cases it is well below the maximum allocated memory for any tier. The serverless implementation, as with NAT, pays the OpenWhisk execution runtime overhead in terms of memory consumption.

These results show how implementing DHCP as a serverless function may be beneficial in the case of unexpected traffic spikes. In fact, a serverless NFV platform as NFoS is elastic by design, and more function instances are automatically instantiated when needed to cope with such an increased amount of traffic. However, this is not true for the serverful DHCP. In this case, either adequate resource provisioning is guaranteed, or the scaling policy of the VNF must be handled by an additional component, such as a VNF Manager (VNFM) if we refer to ETSI NFV Management and Orchestration.

VI. Conclusion

This paper proposes NFoS, an open-source serverless NFV platform that extends OpenWhisk functionalities to make it suitable to the execution of stateful VNFs. The platform was designed and implemented with the specific goal of executing two serverless VNFs, a NAT and a DHCP, and perform a thorough comparison analysis with their serverful counterparts. Implementing serverless NAT required a much higher effort than serverless DHCP for two reasons: *(i)* it was not possible to find an existing suitable serverful implementation to start with, and *(ii)* we had to architect the VNF (e.g., in terms of state management) in such a way that it can be executed by NFoS without incurring high system-level overheads, being a data-plane function with strict requirements in terms of latency and packet loss. A comparison with its serverful version shows how such an effort cannot be justified, as no performance gain is experienced (rather, some loss occurs). Conversely, implementing the serverless DHCP was straightforward starting from a server-based implementation. Keeping the VNF state in an external database, given the looser requirements of such control-plane function in terms of latency, simplified this task. Also, the overall performance is promising compared to the serverful version: NFoS is able to seamlessly scale up/down the DHCP VNFs in response to traffic variations, guaranteeing lower association errors.

Our experience suggests that adopting a per-flowlet serverless NFV platform, able to perform state management of running VNFs, can be suitable for the execution of control-plane, short-lived, and request-based functions. In the future, we plan to implement and test other control-plane VNFs such as a DNS and a MME, to deeper evaluate NFoS both from a functional and non-functional perspective and to integrate our platform with technologies for augmented packet processing performance such as Data Plane Development Kit (DPDK).

References

[1] R. Mijumbi, J. Serrat, J.-L. Gorricho *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] P. Castro, V. Ishakian, V. Muthusamy *et al.*, "The Rise of Serverless Computing," *Comm. of the ACM*, vol. 62, no. 12, pp. 44–54, 2019.

[3] A. Tariq, A. Pahl, S. Nimmagadda *et al.*, "Sequoia: Enabling Quality-of-Service in Serverless Computing," in *ACM Symposium on Cloud Computing*, 2020.

[4] G. Adzic and R. Chatley, "Serverless Computing: Economic and Architectural Impact," in *Joint Meeting on Foundations of Software Engineering (ESEC/FSE)*, 2017.

[5] P. Aditya, I. E. Akkus, A. Beck *et al.*, "Will Serverless Computing Revolutionize NFV?" *Proceedings of the IEEE*, vol. 107, no. 4, pp. 667–678, 2019.

[6] M. Gramaglia, P. Serrano, A. Banchs *et al.*, "The Case for Serverless Mobile Networking," in *IFIP Networking Conference*, 2020.

[7] U. Acar, R. F. Ustok, S. Keskin *et al.*, "Programming Tools for Rapid NFV-Based Media Application Development in 5G Networks," in *IEEE Conference on Network Function Virtualization and Software Defined Networks*, 2018.

[8] C. Cicconetti, M. Conti, A. Passarella *et al.*, "Toward Distributed Computing Environments with Serverless Solutions in Edge Systems," *IEEE Communication Magazine*, vol. 58, no. 3, pp. 40–46, 2020.

[9] A. Singhvi, S. Banerjee, Y. Harchol *et al.*, "Granular Computing and Network Intensive Applications: Friends or Foes?" in *ACM Workshop on Hot Topics in Networks*, 2017.

[10] A. Singhvi, J. Khalid, A. Akella *et al.*, "SNF: Serverless Network Functions," in *ACM Symposium on Cloud Computing*, 2020.

[11] J. Shen, H. Yu, Z. Zheng *et al.*, "Serpens: A High-Performance Serverless Platform for NFV," in *IEEE/ACM International Symposium on Quality of Service*, 2020.

[12] M. Shahrad, J. Balkind, and D. Wentzlaff, "Architectural Implications of Function-as-a-Service Computing," in *IEEE/ACM International Symposium on Microarchitecture*, 2019.

[13] A. Wang, J. Zhang, X. Ma *et al.*, "InfiniCache: Exploiting Ephemeral Serverless Functions to Build a Cost-Effective Memory Cache," in *USENIX Conference on File and Storage Technologies*, 2020.

[14] S. Jindal and R. Ricci, "MME-FaaS Cloud-Native Control for Mobile Networks," in *ACM Symposium on Cloud Computing*, 2019.

[15] M. Ciavotta, D. Motterlini, M. Savi *et al.*, "DFaaS: Decentralized Function-as-a-Service for Federated Edge Computing," in *IEEE International Conference on Cloud Networking (CloudNet)*, 2021.

[16] P. Srisuresh and K. Egevang, "Network Address Translation (NAT) Behavioral Requirements for Unicast UDP," RFC 4787, 2007. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc4787

[17] B. Volz, "Reclassifying Dynamic Host Configuration Protocol version 4 (DHCPv4) Options," RFC 3942, 2004. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc3942

[18] D. Duplyakin, R. Ricci, A. Maricq *et al.*, "The Design and Operation of CloudLab," in *USENIX Annual Technical Conference*, 2019.