A MAPE-K and Queueing Theory Approach for VNF Auto-scaling in Edge Computing

Thiago P. Silva*, Thais V. Batista*, Anselmo L. Battisti†, Andre Saraiva†, Antonio A. Rocha†, Flavia C. Delicato†, Ian Vilar Bastos‡, Evandro L. C. Macedo§, Ana C. B. de Oliveira¶, and Paulo F. Pires¶
*Universidade Federal do Rio Grande do Norte, Natal, Brazil
†IC - Universidade Federal Fluminense, Rio de Janeiro, Brazil
‡Universidade do Estado do Rio de Janeiro, Rio de Janeiro, Brazil
§Universidade Federal do Rio de Janeiro, Rio de Janeiro, Brazil
¶DELL EMC, Brazil

2023 IEEE 12th International Conference on Cloud Networking (CloudNet) | 979-8-3503-1306-2/23/$31.00 ©2023 IEEE | DOI: 10.1109/CloudNet59005.2023.10490065

***Abstract*—Network Function Virtualization (NFV) and Edge Computing (EC) can accommodate various services on a shared virtualized infrastructure. By using distributed resources available at the network edge, the EC paradigm contributes to share service provisioning. To improve agility and flexibility for service provisioning while reducing deployment costs for infrastructure providers, NFV virtualizes computing, network, and storage resources to decouple network functions from the underlying hardware. Therefore, typical network functions in a virtualized network environment are implemented as software entities called Virtual Network Functions (VNFs), which can run on Virtual Machines (VMs) or containers within off-the-shelf servers. The integration of EC and NFV allows the creation of VNF chains, known as Service Function Chains (SFC), representing end-to-end services and their deployment on edge servers. Edge nodes tend to provide fewer stable services once the environment where they are located is unpredictable. Thus, running SFCs with an unpredictable workload is challenging, and many components may cooperate to meet the required Service Level Agreement (SLA). Therefore, such environments require strategies for automatically scaling VNFs as a function of workload fluctuation. This work addressed the VNF scaling problem by providing a novel MAPE-K-based architecture and a queue-based algorithm to dynamically scale VNF in the edge. We demonstrate that the proposed approach outperforms purely reactive auto-scaling.**

***Index Terms*—Autonomic Computing, Auto-scaling, Edge Computing, Network Function Virtualization, Queueing Theory.**

I. Introduction

The emergence of new applications with heterogeneous demands has required the evolution of computer networking technologies. It is envisioned that next-generation networks will need to accommodate a wide variety of services on a common shared infrastructure. Using resources available in a distributed fashion at the network edge contributes to the sharing and flexibility of service provisioning. However, to achieve the agility and cost savings required to meet new application demands, networks must evolve how they deliver services. Technologies such as NFV are frequently used to provide more agility and flexibility for service provisioning while reducing deployment costs for infrastructure providers. In a virtualized network environment, typical network functions (such as multiple access, caching, firewall, etc.) or application-level functions are implemented as software entities called VNFs. The VNF can be run VMs or containers within off-the-shelf servers. Thus, the demand for deploying these functions on dedicated hardware devices is reduced, providing agility, flexibility, and cost-efficiency.

The integration of EC and NFV allows the creation of VNF chains, also known as *Service Function Chains (SFC)*, representing end-to-end services for end users and their deployment on edge servers. An SFC comprises a set of VNFs chained together in a given order and represents a service (or application) provided to the user. Such services will take advantage of the lower latency provided by the edge and the flexibility offered by VNFs. However, the edge environment comprises several resource-constrained devices with heterogeneous CPU and memory capacities that communicate using various protocols via wired and wireless communication technologies.

Edge nodes often have fluctuating capacities due to the dynamic creation, destruction, migration, and scaling of VNF instances. Besides, in comparison with cloud nodes, the resource constraints of edge nodes tend to provide fewer stable services once the environment is less reliable [1]. Consequently, manually scaling VNF instances in response to unpredictable workloads is impractical. Effective VNF scaling requires a coordinated effort from various components to meet SLA requirements. One key mechanism in this context is the VNF auto-scaling, which dynamically allocates and deallocates VNF resources in response to workload fluctuation and resource availability. The main goal of auto-scaling is adapting the system to the workload demands and optimizing resource usage without human intervention.

According to [2], auto-scaling approaches are mainly classified based on i) operational behavior and ii) scalability technique. Operational behavior distinguishes them as proactive or reactive. Proactive approaches predict computing resource usage by analyzing performance metrics (e.g., latency, CPU usage, and workload). Such performance metrics are continually monitored, producing historical data that can be processed by Machine Learning (ML) and/or statistical algorithms to predict future workload, adjusting the number of resources needed by the VNF instances to meet the QoS requirements.

Reactive approaches are based on threshold limits and on constantly monitoring the workload. A scaling operation is triggered whenever a change is detected in the workload that exceeds the defined thresholds. In this work, a proactive approach is employed to identify environmental conditions triggering scaling operations, allowing for anticipatory scaling to prevent service degradation and reduce resource waste.

Concerning scalability techniques, horizontal scaling replicates VNF instances to distribute the workload among them. In contrast, vertical scaling adjusts the allocated resources, such as CPU and memory, during a VNF instance's lifecycle. In this work, our architecture and algorithm primarily focus on vertical scaling operations.

This paper addresses challenges in scaling VNFs at the edge. We propose a MAPE-K architecture for autonomous auto-scaling using a queue-based VNF scaling algorithm. This algorithm calculates packet processing time probabilities, serving as auto-scaling indicators with scaling thresholds. Scaling up occurs when the probability exceeds a threshold, while scaling down happens below it. The algorithm predicts queue lengths and wait times for VNF instances using a M/M/1 queueing model. The main ideas and contributions of our work are summarized as follows:

- We discuss auto-scaling and its importance in the edge environment;
- We discuss the main challenge of making efficient and cost-effective decisions for VNF scaling in a dynamic and heterogeneous edge environment;
- We designed a MAPE-K control loop architecture to provide autonomous properties for auto-scaling;
- We developed a VNF auto-scaling algorithm based on queueing theory to solve the auto-scaling problem.

The rest of this paper is organized as follows. Section II describes background on NFV, EC, and MAPE-K. Section III presents the challenges of making efficient and cost-effective decisions for VNF scaling in the edge. Section IV presents the proposed auto-scaling approach. Section V contains the performance evaluation. Section VI discusses related works. Finally, Section VII contains the final remarks.

II. Background

A. NFV and Edge Computing

NFV and EC are emerging technologies designed to support multiple services on a shared virtualized infrastructure [3]. Using resources available in a distributed fashion at the network edge contributes to the sharing and flexibility of service provisioning. To provide more agility and flexibility for service provisioning while reducing deployment costs for infrastructure providers, NFV virtualizes computing, network, and storage resources, decoupling network functions from the underlying hardware.

In NFV, VNFs represent application-level functions commonly organized into compound SFCs. Figure 1 shows two distinct service function chains, *SFC1* and *SFC2*. Such VNF instances may run on different edge nodes at the network's

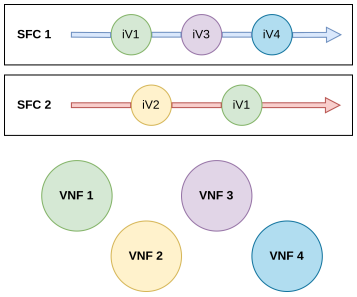

Fig. 1. Example of SFCs and VNFs.

edge. For example, instances of *VNF1* and *VNF2* may share resources on the same edge node. The other VNF instances may run in different edge nodes that use computational resources exclusively. Also, in Figure 1, it can be observed that *VNF1* in the *SFC1* chain acts as ingress VNF and acts as egress VNF in the *SFC2*.

B. MAPE-K based Architecture

The execution environments in NFV-Edge systems are highly dynamic, requiring adaptation and reconfiguration approaches. Adaptive systems dynamically react to changes in their execution environment to increase performance and efficiency. Such systems, also called *self-adaptive* or *autonomic systems*, are also complex and potentially large-scale, requiring automated adaptation actions to avoid errors and slow down decisions and manipulations.

The *Autonomic Computing paradigm*, as described in [4], offers a framework for self-adaptive systems. The framework employs a feedback loop to control the adaptation process. Within this loop, systems use monitors to gather context information and actuators to implement reconfiguration actions. These elements are linked to a control and decision component responsible for dynamic adaptation. The control loop can be defined by adopting the MAPE-K approach, which has subcomponents for analyzing monitored data and planning response actions in case of an adaptation and their execution. All such actions are grounded on a knowledge base with rules, policies, and history concerning the system under administration. Figure 2 illustrates the MAPE-K components.

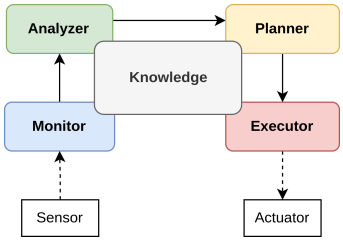

Fig. 2. MAPE-K control loop components.

The primary aim of an auto-scaling mechanism is self-optimization, as it adjusts resources to align with the workload, enhancing overall resource utilization and ensuring timely service requests. This way, an auto-scaling mechanism improves

the overall utilization and ensures that service requests are met promptly. Therefore, auto-scaling is a self-optimizing autonomic manager that automates the tasks that IT professionals must perform to optimize the system.

Considering the role of an auto-scaling mechanism and its autonomic self-optimizing property, the activities associated with the control loop to optimize the system can be categorized into four functions: i) collect information from the environment to identify the symptoms that can trigger a scaling operation, ii) analyze such information and symptoms to determine what should be done to adjust the system, iii) create a plan to adjust the system, and iv) execute that plan.

III. Challenges in VNF auto-scaling at the Edge

Auto-scaling resources is a complex task since it must consider several parameters in the decision process, including the users' requirements and information related to the execution context, such as the availability of resources in the infrastructure. Scaling resources encompasses three main challenging activities: i) detecting the proper moment to scale the resources allocated to the running instances, ii) identifying which instances should be scaled, and iii) determining how many resources to scale in each instance. These activities aim to minimize SLA violations, avoid over and under-provisioning as much as possible, and optimize resource use to accommodate more user requests.

Detecting the right time to scale resources is crucial for scaling quality. Delayed detection can lead to SLA violations, while incorrect decisions on which resource instance to scale may result in over or under-provisioning, affecting system quality. Furthermore, optimizing resource usage hinges on determining the appropriate number of resources to scale, which is essential for successful scaling operations.

Building solutions for VNF auto-scaling in Edge environments is even more challenging due system's heterogeneity and the dynamic nature of the components encompassing the edge network. This work tackles the challenge of making efficient and cost-effective decisions for VNF scaling in a dynamic and heterogeneous edge environment. The following sections discuss these challenges in more depth.

A. Heterogeneity in NFV-Edge Systems

Heterogeneity in NFV-Edge systems stems from multiple sources. EC is designed to take advantage of different types of edge devices, ranging from smart gateways deployed at the user's premises to microdata centers, including base stations of new-generation mobile networks. Therefore, edge nodes have high heterogeneity in terms of technologies, computing, and networking. Heterogeneity creates asymmetry in the individual nodes' capabilities. Collaboration is essential to solving complex tasks, for example, when the information, rights, or resources available on an individual edge device are not enough for it independently performing all the computational tasks required by a VNF. Task collaboration introduces dependencies that impact the scaling process. For instance, scaling the resources for one VNF may require similar adjustments for related VNFs within the same SFC, potentially creating bottlenecks in the system.

Next-generation network environments, such as 5G, are designed for diverse applications and services. According to the International Telecommunications Union, 5G systems support three main types of services: enhanced Mobile Broadband (eMBB) for high data rates; massive Machine Type Communications (mMTC) for numerous sporadically active devices transmitting small data loads (commonly for IoT); and Ultra-reliable low-latency communication (URLLC) for low-latency, highly reliable communication from specific terminals, typically triggered by external events. Therefore, there will also be a high heterogeneity in the service requests that the system will receive. This heterogeneity is both in terms of VNF types and their QoS parameters. To be successful, any resource allocation or scaling solution in these systems needs to consider such heterogeneity.

B. Dynamism in the Execution Context

Unpredictable workloads and variable resource availability in edge environments add complexity to auto-scaling. The workload varies in an NFV system for several reasons. A VNF is placed on a given node for execution, taking into account a snapshot of the system or using average values for resource provisioning and utilization. However, the available resources vary over time in practical scenarios. Moreover, the actual resource utilization of the users is time-dependent. For example, the data traffic volume in the daytime differs from the data traffic volume at night. Regarding resource availability, edge nodes are not dedicated devices; instead, they share their resources and multiplex their time over several tasks. Therefore, their available capacity varies over time. This way, scaling in dynamic, distributed, and potentially large-scale edge systems requires minimal human intervention, as scaling decisions need to be constantly revised to maintain SLA compliance in the presence of workload and resource availability variations. Therefore, scaling solutions must emphasize high autonomy through ongoing performance-based parameter adjustments.

IV. Auto-Scaling Based on MAPE-K and Queueing Theory

Our proposal solution includes i) an architecture based on the MAPE-K autonomic loop and ii) an algorithm based on queueing theory to perform VNF auto-scaling in edge environments. This combined architecture and algorithm effectively tackle the challenges of dynamism and environmental heterogeneity. Our architecture draws inspiration from autonomic computing and adaptive systems to address high dynamism. It explicitly considers the heterogeneity of network edge resources and VNFs through system model variables. The proposed auto-scaling algorithm was based on queueing theory, which considers variations in the workload and tries to accommodate the scaling decisions accordingly. The adoption of queueing theory provides the tools to estimate whether the current resources of a VNF instance are sufficient to handle

the incoming workload based on the maximum delay imposed by the SFC SLA.

Our auto-scaling approach is based on the MAPE-K control loop [5]. We mapped the MAPE-K phases to the main components of the architecture (see Figure 3). These components are in charge of implementing the functions of the MAPE-K loop. It is worth noting that the proposed architecture is designed for dynamic systems. The MAPE-K-based architecture addresses the challenge of the dynamic environment, since the autonomous loop provides mechanisms to constantly monitor and act on the system to adapt it according to modifications in the execution environment.

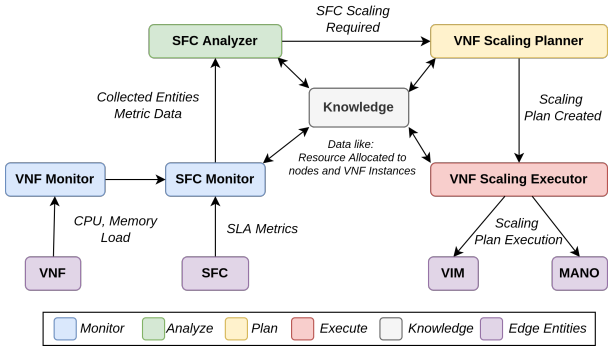

Fig. 3. Proposed auto-scaling architecture based on MAPE-K control loop.

As shown in Figure 3, the first phase of MAPE-K (Monitoring) is performed by the components *VNF Monitor* and $SFC\_Monitor$ that collect different metrics from the currently running VNFs and SFCs instances. Note that an SFC encompasses several VNF instances. Thus, the SFC-related metrics involve the metrics of all the respective VNF instances. VNF metrics indicate the individual conditions of the monitored entities, e.g., CPU and memory consumption of the container running the VNF instance.

The $SFC\_Analyzer$ component receives the information (symptoms) monitored by $SFC\_Monitor$, and uses such information as input for the decision-making processes. That is, the $SFC\_Analyzer$ is responsible for the Analysis phase of MAPE-K and decides whether it is necessary to perform a scaling action. Note that this decision can be based on thresholds, so the behavior is reactive. Alternatively, the decision can be based on the predictions made by a statistical or ML model, so it behaves proactively, anticipating the scaling actions. The architecture is flexible enough to accept both types of scaling (reactive and proactive).

Both the threshold values, used by reactive algorithms, and machine learning models, employed by proactive approaches, are maintained by the component $knowledge$. The $knowledge$ base is used throughout the entire MAPE cycle to inform decision-making in all phases. In addition, this component keeps a history of the system's states over time so that the scaling actions already taken can be checked for assertiveness. Hence, the system can improve itself from previous decisions, understand its environment, identify problems, and execute actions to achieve its goals.

Whenever it is necessary to adapt the system, a request is sent to the $VNF\_Scaling\_Planner$, which structures the scaling action (represented as a Scaling Plan in Figure 3). Such a component decides how many resources need to be scaled. There are different approaches to deciding the number of resources provisioned or de-provisioned from VNF instances. However, all approaches must consider the capacity in terms of resources of the nodes hosting VNF instances that will be scaled. For example, using a fixed rate, a linear strategy can increase or decrease the number of resources at each scaling action. On the other hand, an exponential strategy can exponentially increase or decrease the resources with each scaling action. The architecture is flexible to accommodate such different approaches.

The $VNF\_Scaling\_Executor$ performs the Execute phase of the control loop. It executes scaling actions in the virtualized infrastructure based on the actions specified by the planning phase, performed by $VNF\_Scaling\_Planner$. Its actions depend on the underlying environment used to execute the VNF and SFC instances. For example, in the ETSI architecture, the Network Function Virtualization Infrastructure (NFVI) is responsible for scaling up and down. Thus, $VNF\_Scaling\_Executor$ will use the northbound API of the VNFI to coordinate the scaling up and down actions.

The proposed auto-scaling system uses pre-existing components defined by the ETSI NFV Reference Architecture [6]. Reusing such components is critical to avoid rework and maintain interoperability with the underlying hardware infrastructure and software platforms, since MANO is a widely used standard.

A. VNF Auto-Scaling algorithm based on queueing theory

This work proposes a novel algorithm for VNF auto-scaling in EC environments based on the queueing theory [7], [8]. The Queue-based algorithm assumes that each VNF instance is modeled as an M/M/1 queue. Figure 4 summarizes the activities representing the algorithm and the respective transitions between them, thus denoting the algorithm flow. The algorithm works at all MAPE-K phases to perform the autonomous control loop, and how the algorithm is adapted to each MAPE-K phase is briefly described in the following.

The algorithm retrieves the data from the packets processed in the SFC instance in the $SFC\_Monitor$ component at every MAPE-K control loop. In our model, we consider a constant overhead for collecting the packets, i.e. we assume that for each packet, the *Monitor* takes a constant amount of time to retrieve it. In the $SFC\_Analyzer$ component, each VNF instance v that composes the SFC instance has its arrival rate λ^v defined as the sum of each packet size i of n packets received by the VNF instance in the monitoring interval. The maximum processing delay of each VNF instance is also defined in the $SFC\_Analyzer$ component. Each SFC instance has a maximum delay (d^s) that each packet must respect when arriving and leaving the SFC instance.

The SFC instance maximum delay comprises the propagation, transmission, and queue delay (d_l^s) of each link $l \in s$

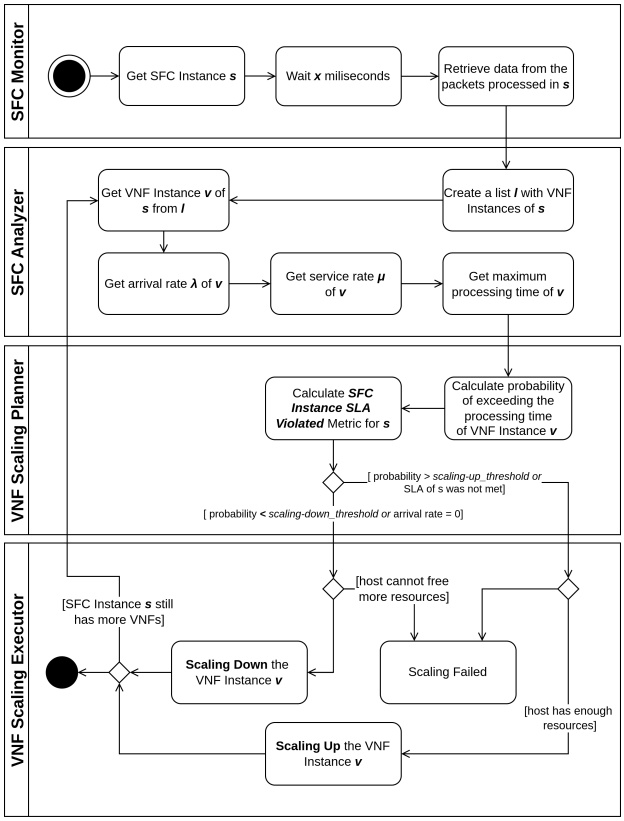

Fig. 4. Activity diagram for the auto-scaling algorithm based on the queueing theory.

connecting the VNF instances and each VNF instance processing delay. The maximum processing delay (d^s) assumed for the VNF instance is equal to the SFC instance maximum delay, disregarding the links associated delays, multiplied by the amount of CPU (CPU^v) over the sum of the CPU instructions of all VNF instances combined($CPU^v / \Sigma_{v' \in s} CPU^{v'}$). Therefore, d^v is the maximum processing delay contribution of each VNF instance in the SFC instance, considering the number of CPU resources that the VNF instance has:

$$d^v = \left(d^s - \sum_{l \in s} d_l^s \right) \left(\frac{CPU^v}{\sum_{v' \in s} CPU^{v'}} \right) \quad (1)$$

The $VNF\_Scaling\_Planner$ component calculates the service rate of each VNF instance (μ^v). The service rate is defined as the estimated rate at which the VNF instance can process packets in terms of packet size. The service rate of a VNF instance v is calculated by the inverse of the sum of each packet size of the m processed packets, multiplied by the number of instructions that are required to process the packet, divided by the number of CPU instructions of the VNF instance (CPU^v).

The computed probabilities can be used as an auto-scaling indicator by both proactive and reactive approaches. In the proactive approach, when the probability exceeds the scaling-up threshold, the algorithm performs the scaling-up procedure, even if the SLA is not being violated. In the reactive approach, the scaling-up procedure is performed only when the algorithm observes SLA violations.

The algorithm can also handle the workload dynamism over time by computing the probabilities. Computed probability values closer to zero indicate that the currently allocated resources for the VNF instance are sufficient to handle the monitored workload or that the VNF instance is over-provisioned. On the other hand, when the calculated probability values are closer to one, the monitored workload is overwhelming for the VNF instance currently allocated resources. Therefore, the computed probabilities capture the workload fluctuation, and the auto-scaling algorithm based on queueing theory dynamically adapts the VNF instances resources according to the monitored workload.

To detect the proper moment to scale the VNF instance v, the Queue-based algorithm verifies the waiting time probability in VNF instance v (T_W^v) of each packet being higher than the maximum processing delay ($P(T_W^v > d^v)$). The waiting time is a random variable that comprises two random variables: i) the time a package spent in the queue waiting to be processed (T_Q^v) in the VNF instance, ii) the package processing time (T_P^v). The queueing discipline is First Come First Served, and immediately after a new packet arrives, the queue length is $k+1$, where k is the number of packets currently in the queue. Therefore, the arriving packet's waiting time at VNF instance is represented by an independent and exponential distribution with parameter μ^v, and X as the residual time of the packet being processed at the arrival instant. Because of the memoryless property of exponential distributions, X has the same exponential distribution as the processing time. Since the arrival process follows a Poisson distribution, the probability that the queue length is k when a packet arrives is the same as the equilibrium probability [9].

V. Performance Evaluation

We compared the proposal with a rule-based auto-scaling approach that uses fixed thresholds to trigger scaling actions. In such an approach, lower and upper limits are specified for the CPU consumption (CPU Load) of each VNF instance. Whenever consumption exceeds these limits, a scaling action is triggered, and a factor increases or decreases the number of CPU resources linearly (e.g., 10%).

Besides, we evaluated our proposed auto-scaling using two strategies (Linear and Exponential) to define the number of CPU and Memory resources provisioned or deprovisioned from VNF instances in case of scaling actions. The *Linear* strategy defines a linear factor so that the number of resources increases or decreases smoothly. The *Exponential* strategy is based on the TCP window growth function, doubling the resources with each scaling up. On the other hand, scaling down actions use linear factors. In the text, we adopt the nomenclature *SmartQueueingScaling - L* to refer to the linear strategy and *SmartQueueingScaling - E* to refer to the exponential strategy.

We evaluated the performance of the scaling approaches in terms of (i) resource consumption, (ii) scale actions, and

(iii) SLA violations. Therefore, we defined the metric *SLA Violations* as the number of packets with a total delay higher than the delay defined in the SLA. *Request Met* is the percentage of requests fulfilled without SLA violation. *CPU Allocated* is the average allocated CPU resource to the SFC. We sum the allocated CPU per VNF instance that compose a given SFC. *Scaling Operations* is defined as the number of scaling actions performed during the simulation. We published all the source code on GIT[1] to foster the reproducibility of results.

A. Experiment Scenario

The simulated EC scenario relies on an extension of the SimPy simulator[2], a Python library that implements a framework for event-driven simulations. Active elements, such as Nodes, Links, VNF Instances, and VNF Requests, are modeled as processes in the simulator. Such processes are Python generators responsible for creating events (for example, a packet arriving in the VNF queue) and yielding them for processing regarding a timeout value. When an event is yielded, the process that yields such an event is suspended and waits for the processing of the event to be finished.

Our experiments considered a 5G Best Effort scenario. In such a scenario, we simulated nodes in a fully connected network with at least one link between two nodes. Also, we simulated the arrival of packets that must be processed through the VNF instance that composes the SFC. The SFC comprises 3 VNF instances, and each VNF instance has a minimum value for the allocated CPU and Memory resources. The requirements in such a scenario are more flexible and less restrictive than in other scenarios, such as URLLC and eMBB. The chosen scenario represents mixed applications with a variety of QoS requirements. The parameters for this scenario are based on [10], [11].

We set a simulated network with 1.13 milliseconds as the transmission delay in each link according to the current 5G computing scenarios. We simulated an environment comprising ten edge nodes with 10,000 Million instructions per second (MIPS) and 30,000 Memory units as capacities. We used 100 ms as a monitoring window, i.e., every 100 ms, the auto-scaling is activated and performs a scaling decision. Such a decision can increase, decrease, or maintain the number of resources allocated to the VNF instance).

As a pre-condition to running the proposed autoscaling, it is first necessary to run a placement algorithm. Such an algorithm ensures that the edge nodes hosting VNF instances have sufficient resources to meet the SFC and VNF requirements. Our research group proposed the VNF placement algorithm described in [12]. After creating the placement plan, packets are generated using a given workload pattern.

We varied the amount of resource increment and decrement (parameters *Resource Increment* and *Resource Decrement*) during the simulations, assuming 10%, 25%, and 50% for *Resource Increment*, and 10% and 25% for *Resource Decrement*. The minimum and maximum CPU loads are specific parameters to the *Ruled-Based* algorithm. It is worth noting that cloud providers typically use 10% and 80% as default thresholds to trigger scaling actions [13], and we choose the same values. Based on empirical knowledge, we set 0.75 and 0.25 for parameters *Scaling Up Threshold* and *Scaling Down Threshold* of the proposed auto-scaling.

[1]https://github.com/thiagopereirasilva/cloudnet2023

[2]https://simpy.readthedocs.io/en/latest/

B. Workload Pattern

During the simulations, several packets are generated and sent to the SFC to be processed. The ingress VNF receives the packets, processes them, and sends the output to the other VNF instance in the chain. Each user generates 2,000 packets following a Poisson distribution. As illustrated in Figure 5-B, the workload is challenging because there is not only variation in the number of packets per second but also in the size of each packet. The SFC's maximum tolerated delay (SLA) was 30 milliseconds. A packet must be processed through the SFC instance in less than 30 milliseconds; otherwise, an SLA violation occurs. Although real network traffic can be more complex due to variations in arrival rates, bursty traffic patterns, and factors like routing or user behavior, the Poisson arrival process is a valuable simplification for modeling network arrivals and is widely used.

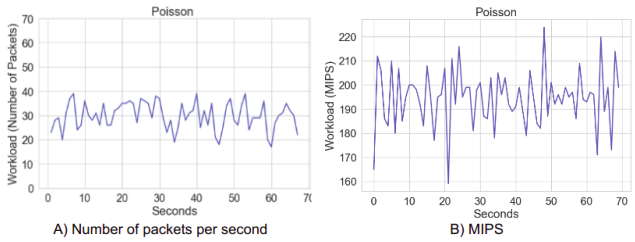

Fig. 5. Poisson Workload Pattern used in the simulations.

C. Results and Analysis

Considering all algorithm's parameters, six different setups of each auto-scaling algorithm were evaluated. We used a box-plot, a graphical representation of the distribution, to present the results because it provides a summary of key statistics and allows visualize the spread and central tendency of the data. In the following Figures, the central box represents the middle 50% of the data, the line inside the box is the median, and the whiskers extend from the box to the minimum and maximum values. The following are the results of the evaluations. Considering the number of SLA violations, as depicted in Figure 6, the *Ruled-Based* approach showed a higher number of violations. On average, was observed 345 SLA violations per simulation, corresponding to approximately 17.29% of violations of the total packets. A possible reason for the high number of SLA violations is the reactive behavior of this algorithm.

On average, the *SmartQueueingScaling - L* and *SmartQueueingScaling - E* violated 130 and 99 packets,

respectively. The quantity of SLA violations, on average, corresponds to 6.51% and 4.98% of the total packets.

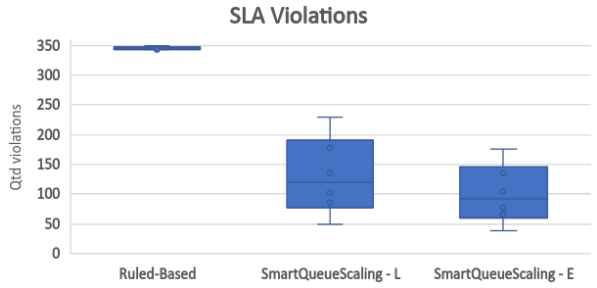

Fig. 6. SLA Violations observed per algorithm during the simulations under Poisson workload.

The *SmartQueueingScaling - E* violated, on average, 23.84% fewer packets than the version with linear strategy. The results demonstrate that the proposed SmartQueueingScaling with exponential strategy outperforms the other baseline approaches regarding SLA compliance.

Figure 7 illustrates the quantity of CPU allocated during the simulations. The *Ruled-Based* showed the least variation in CPU resource allocation throughout the simulations. Such an approach performed few scaling actions, which implied many SLA violations. The *Ruled-Based* allocated an average of 751 MIPS per SFC. The *SmartQueueingScaling - L* allocated an average of 4,002 MIPS, while the exponential strategy demanded an average of 5,001 MIPS. These values indicate that the SmartQueueingScaling algorithm performed several scaling actions to meet the workload.

When analyzing the behavior of the SmartQueueingScaling algorithm, it is possible to observe a greater range in the variation of CPU resources allocated during the simulations. This shows that SmartQueueingScaling is sensitive to the parameters used in the different setups. The *SmartQueueingScaling - E* violated fewer packets than the version with linear strategy at the cost of allocating more resources.

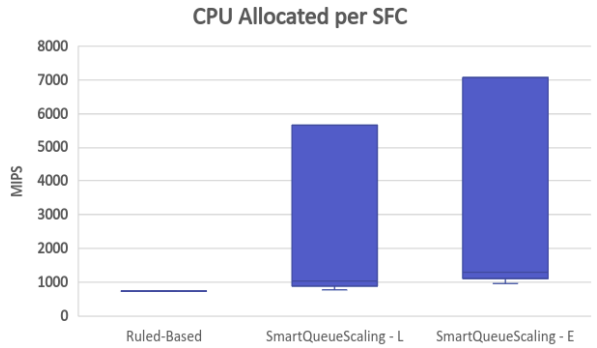

Fig. 7. CPU resource allocated per SFC during the simulations under Poisson workload.

Figure 8 illustrates the amount of scaling actions performed per each algorithm during the simulations. In all algorithms, the amount of scaling-down actions is greater than the scaling-up actions, reflecting the values set to parameters *Resource Increment* and *Resource Decrement*.

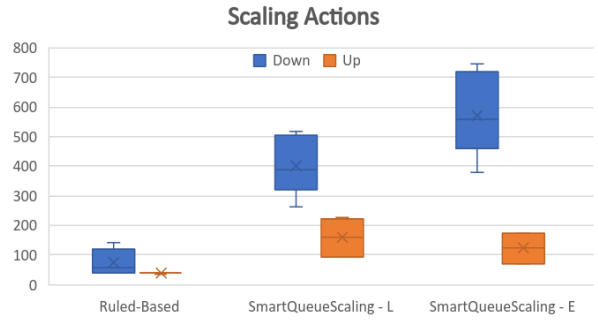

Fig. 8. Scaling actions performed per algorithm during the simulations under Poisson workload.

The *SmartQueueingScaling - E* doubles the number of resources to the VNF instances with each scaling-up action. In this sense, by rapidly increasing the number of resources, such a strategy performs more scaling-down actions to adjust the number of resources to the initial amount. It can be observed that the fulfillment of more requests without violating the SLA was conditioned to the increase in the average consumption of CPU resources.

VI. Related Work

Although auto-scaling techniques are widely used in Cloud computing, they cannot be directly employed in Edge environments, mainly due to the high heterogeneity and dynamism. Next, we present some related work, and a comparison with our approach is presented in Table I.

Bali *et al.* [14] propose a reactive auto-scaling method utilizing clustering to group edge nodes and employing the MAPE-K control loop to monitor cluster state and service performance on its devices. This auto-scaling adjusts the number of service instances (horizontal scaling), corresponding to service replicas, based on rules and in response to workload changes. It is suitable for scenarios with unknown data behavior a priori. However, reactive approaches tend to perform more scaling actions, potentially degrading the system's performance. Unlike our approach, their solution neither anticipates scaling actions nor mitigates oscillations in workload.

[15] propose a proactive auto-scaling architecture for containerized applications. Such architecture provides horizontal scaling and employs a prediction model based on Long Short-Term Memory (LSTM), which requires a training dataset. However, LSTM's complex structure can lead to slower performance, with predictions taking several seconds [21]. While this architecture effectively reduces workload demand oscillations, unlike our approach, it's unsuitable for scenarios lacking prior dataset representation of application workload behavior. Additionally, the model may become outdated as the workload pattern evolves, making accurate predictions challenging.

Tseng *et al.* [16] present a reactive fuzzy-based auto-scaling for the edge environment using hypervisor and container

TABLE I
COMPARISON AMONG RELATED WORKS.

Work	Technique	Scaling Method	Strategy	Mitigate Oscillations	Require Dataset
[14]	Rule-based + MAPE-K	Horizontal	Reactive	No	No
[15]	LSTM + MAPE	Horizontal	Proactive	Yes	Yes
[16]	Fuzzy Theory + MAPE-K	Horizontal	Reactive	No	No
[17]	Rule-based	Vertical	Reactive	Yes	No
[18]	RNN + DL	Vertical	Proactive	Yes	No
[19]	DQN	Horizontal	Proactive	Yes	No
[20]	Queueing Theory	Horizontal	Proactive	Yes	No
This proposal	Queueing Theory + MAPE-K	Vertical	Proactive	Yes	No

virtualization technologies. This approach controls the number of service instances (horizontal scaling) by considering CPU, memory, and network as performance metrics. A set of thresholds for such metrics are previously defined, and "if-then" rules for fuzzy inference are used to determine the scaling actions to be performed without using the MAPE-K control loop. There is no mechanism to prevent scaling oscillations, and this work does not systematically organize the scaling process (e.g., MAPE-K control loop).

In [17], Wang *et al.* present a framework to manage edge nodes, which provides mechanisms for provisioning and auto-scaling edge node resources. The authors proposed a reactive auto-scaling approach that uses application latency as the scaling indicator to increase or decrease the number of resources (vertical scaling) associated with the container running a given application. Although the framework does not systematically organize the scaling process, it mitigates oscillations by running the containers with different priorities. This way, the auto-scaling approach does not progressively scale down containers with lower priority. Instead, the container with the lowest priority is terminated until there are sufficient resources for the container with the highest priority to scale up.

In [18], Etemadi *et al.* propose an auto-scaling framework for resource scaling in the fog tier by applying Recurrent Neural Network (RNN). To handle dynamic workloads in the fog environment, the *Resource Manager* (RM) module of the framework decides what to do (scale up/scale down/no action) according to the IoT device input request and the RNN method. Such a framework minimizes the total cost (resource usage) and QoS violation. The metrics used are CPU utilization, delay violation, and network usage. A request is sent to the cloud nodes when an IoT device request has a deadline (user QoS latency response) that exceeds a given threshold. Otherwise, RM must decide which fog node will process the request according to the "memory" (previous requests) associated with each one.

Lee *et al.* [19] propose an auto-scaling method based on deep Q-networks (DQN) for resizing the number of instances (horizontal scaling) in a service within the MEC environment. This approach was implemented as an open-source module within OpenStack. In the MEC environment proposed by the authors, services consist of multiple components deployable independently in VMs or containers. The approach was evaluated using an OpenStack-based testbed with several servers connected through OpenFlow switches. The evaluation employed a Network Service Chaining (SFC) composed of four network functions (VNFS): a firewall, flow monitor, deep packet inspection, and intrusion detection system. According to the authors, a VNF instance and SFC are considered microservices and services in the evaluation.

[20] presents a queueing-based modeling of SFC and a performance analysis that shows a clear correlation between analytical and experimental results. The results corroborate that queueing theory can be used to define the capacity of elements in a SFC.

VII. CONCLUSION AND FUTURE WORK

We present a novel MAPE-K-based architecture and a queueing-based algorithm to scale VNFs in NFV-Edge. The proposed architecture has the novel aspect of applying self-optimizing property to resource auto-scaling. The auto-scaling is an autonomic manager that automates the scaling tasks, allowing an innovative human-out-of-the-loop approach in a timely, efficient, and less error-prone way. We designed the four steps of the MAPE-K model as components, which allows different implementations of each component according to the requirements of individual systems.

Some future research directions are planned. First, we intend to use real datasets to perform further experiments, and experimentally compare our work with proposals reported in the current state of the art. These are ongoing work due to the difficulty of finding quality datasets representing application workloads running at the network's edge. Likewise, all proposals mentioned in our related work section do not make their source code available, leading to the need to implement their solutions from scratch. The second future direction is the adoption of Reinforcement Learning (RL) agents for dynamic adjustment of the number of resources to be scaled in each scaling operation. Considering the current system state and the "knowledge" acquired over time, the RL agent can determine the near-optimal amount of resources for the VNF instance to meet the demand and prevent wastage.

ACKNOWLEDGMENT

This work was partially funded by DELL EMC, CAPES, CNPq, FAPERJ and FAPESP (Grant 2015/24144-7). Flavia Delicato, Paulo Pires and Thais Batista are CNPq Fellows. Professor Delicato is also FAPERJ Fellow.

REFERENCES

[1] R. Roman, J. Lopez, and M. Mambo, "Mobile edge computing, fog et al.: A survey and analysis of security threats and challenges," *Future Generation Computer Systems*, vol. 78, pp. 680–698, 2018.

[2] S. Taherizadeh and V. Stankovski, "Auto-scaling applications in edge computing: Taxonomy and challenges," in *Int. Conf. on Big Data and IoT*. ACM, 2017, p. 158–163.

[3] G. Baldoni, P. Cruschelli, M. Paolino, C. C. Meixner, A. Albanese, A. Papageorgiou, H. Khalili, S. Siddiqui, and D. Simeonidou, "Edge computing enhancements in an nfv-based ecosystem for 5g neutral hosts," in *2018 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, 2018, pp. 1–5.

[4] J. Kephart and D. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[5] P. Arcaini, E. Riccobene, and P. Scandurra, "Modeling and Analyzing MAPE-K Feedback Loops for Self-Adaptation," in *2015 IEEE/ACM 10th SEAMS*, 2015, pp. 13–23.

[6] ETSI GS NFV 002, "Network functions virtualization (NFV); architectural framework v1.1.1," ETSI, Tech. Rep., October 2013.

[7] E. Jafarnejad Ghomi, A. M. Rahmani, and N. N. Qader, "Applying queue theory for modeling of cloud computing: A systematic review," *Concurrency and Computation: Practice and Experience*, vol. 31, no. 17, p. e5186, 2019, e5186 CPE-18-0152.R1.

[8] R. B. Cooper, "Queueing theory," in *ACM '81 Conference*. New York, NY, USA: Association for Computing Machinery, 1981, p. 119–122.

[9] R. Wang, *Equilibrium Probability Distribution*. Springer New York, 2013, pp. 673–674.

[10] J. J. Alves Esteves, A. Boubendir, F. Guillemin, and P. Sens, "Heuristic for edge-enabled network slicing optimization using the "power of two choices"," in *2020 16th International Conference on Network and Service Management (CNSM)*, 2020, pp. 1–9.

[11] G. Mountaser, M. Condoluci, T. Mahmoodi, M. Dohler, and I. Mings, "Cloud-ran in support of urllc," in *2017 IEEE Globecom Workshops (GC Wkshps)*, 2017, pp. 1–6.

[12] A. L. E. Battisti, E. L. C. Macedo, M. I. P. Josue, H. Barbalho, F. C. Delicato, D. C. Muchaluat-Saade, P. F. Pires, D. P. d. Mattos, and A. C. B. d. Oliveira, "A novel strategy for vnf placement in edge computing environments," *Future Internet*, vol. 14, no. 12, 2022. [Online]. Available: https://www.mdpi.com/1999-5903/14/12/361

[13] M. Catillo, U. Villano, and M. Rak, "A survey on auto-scaling: How to exploit cloud elasticity," *Int. J. Grid Util. Comput.*, vol. 14, no. 1, p. 37–50, jan 2023. [Online]. Available: https://doi.org/10.1504/ijguc.2023.129702

[14] A. Bali, M. Al-Osta, S. B. Dahsen, and A. Gherbi, "Rule based auto-scalability of IoT services for efficient edge device resource utilization," *JAIHC*, pp. 1–18, 2020.

[15] M. Imdoukh, I. Ahmad, and M. Alfailakawi, "Machine learning-based auto-scaling for containerized applications," *Neural Computing and Applications*, vol. 32, pp. 9745–9760, 07 2019.

[16] F.-H. Tseng, M.-S. Tsai, C.-W. Tseng, Y.-T. Yang, C.-C. Liu, and L.-D. Chou, "A lightweight autoscaling mechanism for fog computing in industrial applications," *IEEE Trans. on Industrial Informatics*, vol. 14, no. 10, pp. 4529–4537, 2018.

[17] N. Wang, B. Varghese, M. Matthaiou, and D. S. Nikolopoulos, "Enorm: A framework for edge node resourcemanagement," *IEEE Trans. on Services Computing*, vol. 13, no. 6, pp. 1086–1099, 2020.

[18] M. Etemadi, M. Ghobaei-Arani, and A. Shahidinejad, "A cost-efficient auto-scaling mechanism for iot applications in fog computing environment: a deep learning-based approach," *Cluster Computing*, vol. 24, no. 4, 2021.

[19] D. Lee, S. Jeong, K. Ko, J. Yoo, and J. W. Hong, "Deep Q-Network-Based Auto Scaling for Service in a Multi-Access Edge Computing Environment," *Int. J. Netw. Manag.*, vol. 31, no. 6, nov 2021.

[20] A. Heideker, I. Zyrianoff, and C. A. Kamienski, "Profiling service function chaining behavior for nfv orchestration," in *2018 IEEE Symposium on Computers and Communications (ISCC)*, 2018, pp. 01 020–01 025.

[21] Y. Zhang, X. Hao, and Y. Liu, "Simplifying long short-term memory for fast training and time series prediction," *Journal of Physics: Conference Series*, vol. 1213, p. 042039, 06 2019.