Dynamic and Mobility-Aware VNF Placement in 5G-Edge Computing Environments

Juan Lucas Vieira*, Anselmo L. E. Battisti*, Evandro L. C. Macedo†,
Paulo F. Pires‡*, Débora C. Muchaluat-Saade*, Flavia C. Delicato*, and Ana C. B. Oliveira‡
*MídiaCom Laboratory, Universidade Federal Fluminense, Niterói, Brazil
†High-Speed Networks Laboratory, Universidade Federal do Rio de Janeiro, Rio de Janeiro, Brazil
‡Dell EMC Research Center, Rio de Janeiro, Brazil
Emails: juanlucasvieira@id.uff.br; anselmo@midiacom.uff.br; evandro@ravel.ufrj.br;
paulo.pires@dell.com; debora@midiacom.uff.br; fdelicato@gmail.com; ana.oliveira@dell.com

***Abstract*—The combination of both Network Function Virtualization (NFV) and Edge Computing facilitates the Quality of Service (QoS) fulfillment of latency-stringent applications by taking advantage of the proximity of the devices at the edge and the user, and provides the necessary flexibility to enable agile, cost-effective, and on-demand service delivery models for 5G environments. However, to achieve those benefits, VNFs must be deployed so that system resource utilization is optimized and service QoS is maintained. In this context, the selection of the most suitable edge nodes to deploy VNFs is a challenging research issue because multiple — and often conflicting — objectives must be considered by the VNF placement algorithm. The challenges become greater when considering the high dynamism of the edge environment. The availability of edge nodes and users' locations vary over time, making original allocation decisions ineffective. In this paper, we propose DSP, a Dynamic Smart VNF Placement algorithm that considers the dynamic nature of a 5G edge environment. Our proposal focuses on optimizing resource utilization with an online placement approach to handle continuously arriving service requests. Furthermore, we also account for user mobility providing an approach that reduces the performance impact caused by users' movement. Additionally, a two-level resource-sharing mechanism is provided, minimizing the cost for the infrastructure provider with low impact on QoS.**

***Index Terms*—Network Function Virtualization, 5G, Edge Computing, User Mobility**

I. Introduction

The fifth generation of mobile networks (5G) increased the number of new service patterns, imposing new requirements for flexibility and stringent Quality of Service (QoS). To deal with such demands, Network Function Virtualization (NFV) has been used to replace dedicated hardware-based functions with software counterparts known as Virtual Network Functions (VNFs) [1]. NFV decouples network functions from proprietary hardware by virtualizing computing, network, and storage resources, allowing software implementations of these network functions to be run on commodity hardware. This provides the necessary flexibility to enable on-demand service delivery with automated management in an agile and cost-effective manner.

In an NFV environment, the functionalities required by a user are implemented as an organized set of VNFs, known as a Service Function Chain (SFC) [2], that is deployed using the resources provided by an underlying NFV Infrastructure (NFVI). The Cloud Computing paradigm is often used to enable the deployment of VNFs using resources of cloud data centers. However, as services might have stringent latency requirements, it is often desirable that the delay experienced by network packets is adequately short. To this end, Edge Computing is a key technology that enables data processing closer to users and, when combined with NFV, facilitates the QoS fulfillment of low-latency services through the deployment of VNFs at the edge nodes.

A well-known challenge in the NFV field is the placement problem [3]–[5], which consists of selecting the most suitable nodes to deploy VNFs. The major difficulty is that resources must be utilized considering multiple objectives, that often conflict. For instance, one of the goals is to provide enough resources so that services that rely on the VNFs can be provided with good QoS. In contrast, nodes must be selected so that resource utilization is optimized to reduce costs, increasing the revenue of service providers.

Another factor that makes allocating VNFs difficult is the heterogeneity of the environment [6]. As the edge tier is composed of heterogeneous computational and networking resources, the capacity of each node and link must be considered when placing a VNF. Furthermore, as the 5G technology specifies different usage scenarios, requested services will also have different requirements - some services might require low latency communication, while others will prioritize high throughput speeds - which must also be considered during the VNF placement decision.

Offline placement solutions [7]–[9] usually assume that all information regarding the requests is known *a priori* and the placement decisions will remain static throughout the system operation. However, knowing requests information at the beginning of the system operation might be an impractical assumption [10], especially under the 5G edge environment, where the number of users, their locations, and resource availability vary over time, further increasing the complexity of the VNF placement decisions. Service requests continuously arrive as users enter the system or start using a different application. Because of that, the system must be able to receive requests at any time and promptly serve them by deploying new SFCs. Placement decisions will have to be made over time to accommodate new requests that might enter the system in a scenario where other VNFs are already placed and running. This is particularly difficult because the availability of resources at the edge varies every time a new SFC is placed or terminated, which impacts future placement decisions [6].

In 5G environments, users can move freely from one location to another over time while running applications from their devices. Eventually, these movements will impose changes in the network path that connects the user to an ongoing service at the edge. Considering an already placed SFC, users might experience greater latency as they move to access nodes that are further from the service, impacting performance. In this case, resources from nodes in the user vicinity could be used to relocate the service closer to the user, reducing latency. To cope with service degradation caused by user mobility, the system must be able to relocate ongoing SFCs onto a new set of suitable nodes to reduce service latency and maintain the QoS specified in the Service Level Agreement (SLA).

In this paper, we propose a Dynamic Smart VNF Placement (DSP) heuristic to solve the VNF placement problem using a non-optimal, but effective approach that can deploy VNF instances at heterogeneous edge nodes and is aware of latency, computational resource capacity, and energy consumption, supporting different service classes. It chooses the candidate edge nodes that better fit the required characteristics of the SFCs. Our work tackles the dynamics of the 5G environment by considering an online placement solution in which user requests arrive at the system over time, with no *a priori* knowledge of the arrival rate, and arriving requests must be placed in an edge environment where previously placed VNFs are already running. DSP also supports user mobility, triggering a replacement procedure to reduce performance deterioration caused by the user's mobility on executing services. We also provide an efficient two-level resource-sharing mechanism that is able to reduce resource consumption with a low impact on service performance.

The remainder of this paper is organized as follows. Section 2 presents works that address the VNF placement problem. Section 3 presents the system model under consideration and the proposed dynamic placement solution. Section 4 presents the performance evaluation of the proposal. Finally, Section 5 concludes the work and presents future research remarks.

II. Related Work

VNF instantiation and placement is a broadly researched area, with several works in the literature addressing the allocation of VNFs at an underlying computing infrastructure [3]–[5]. Considering the different classes of services brought by the advance of 5G networks, SFCs might have different QoS priorities to be met. Some services might have stringent latency requirements, so that the problem of VNF allocation will not be restricted to finding the best resource allocation for each VNF demand, but also locate VNFs in a way that the end-to-end latency SFC requirements are satisfied.

Harutyunyan et al. [11] propose three ILP (Integer Linear Programming) approaches for the problem of SFC Placement in 5G networks. The first approach aims at reducing service end-to-end latency, which prioritizes placing SFCs close to the user. The second ILP model focuses on reducing the SFC provisioning cost in terms of radio, CPU and bandwidth resource usage costs. A third approach has the goal of minimizing VNF migrations. In that work, migrations are made to readapt the current placement state to fulfill arriving requests that would be rejected unless VNFs are migrated from one location to another. A heuristic for the later approach is also provided to cope with the poor scalability of the ILP proposal. Despite the reported gains, that work considers that users remain static throughout the whole service duration. The work is extended in a follow-up paper [12] to address user mobility. In the extension, the authors explore four different approaches related to the SFC placement in 5G. The first three have the same goals of reducing end-to-end delay, minimizing service provisioning cost and VNF migrations, while the fourth one tries to minimize the number of handovers between two different Centralized Units (CUs). In the evaluation section, the authors cite that the requests arrive sequentially in batches, but the impact of the batch size on the monitoring overhead is not addressed. In comparison, our proposal considers that service requests can arrive continuously and at any time. Requests that arrive within a time window are grouped in a set of requests to be placed at the end of the time window to reduce the overhead of monitoring available resources at the edge nodes. Our work also evaluates the influence of the time window size in monitoring overhead.

User mobility is also addressed by Zheng et al. [13]. The authors rely on a mobility predictor to decide the best location to place a VNF. The predictor provides the probability of each user to visit a destination node, which is later used by the ILP approach with the objective of reducing placement and routing costs. The authors also provide a heuristic approach to deal with the long execution times required by ILP. In the heuristic, the node with the highest mobility probability is set as the destination, whereas the initial node is chosen so that the routing cost between the initial and destination nodes is minimized. Then, VNFs are placed in the VNF-enabled nodes of the path that connects both initial and destination nodes. The authors report a significant reduction in terms of routing costs. However, their approach requires prior information about user mobility, which might not be available. Also, services with stringent latency requirements often will need to be placed as near as possible to the user, and as the user moves away from the service, simply rerouting the packets might not be enough to maintain latency within the agreed standards, requiring the service to move with the user in some way. Although the authors cite VNF migration as one possible mobility-aware solution, they do not address it in their proposal. In our proposal, upon detecting performance degradation due to user movement, the service chain is moved to nodes closer to the user, reducing the user-service latency.

Ma et al. [14] also use the mobility probability of users in the decision process of where to instantiate VNFs. The authors propose a hybrid mobility support approach that combines a reactive and proactive VNF instantiation mechanism. In their work, replicas of VNFs are placed in strategic locations considering the movement pattern of users – *i.e.*, the probability of each user being in a location. If the user moves to a location and a VNF replica cannot provide the service without violating latency requirements, then a new VNF is instantiated, or a replica is migrated to a nearby node. The authors also propose an online approach, where a time interval is divided into time frames, which in turn can be divided into time slots. VNF replicas are instantiated at the beginning of a time frame. Instances of new VNFs and migrations are performed at the beginning of a time slot. Their work considers that the service is provided by only a single VNF, not addressing VNF chains that form an SFC. In fact, considering a chain of VNFs is a more challenging task. The placement of the corresponding VNFs would have to respect a specific order, the migration of the service would have to consider the migration of multiple VNFs, and the number of replicas would be significantly higher to meet the demand of services. Different from the mentioned work, our proposal considers that one service is provided by a chain of ordered VNFs. Thus, when we detect that an SFC must be moved to reduce service delay, new appropriate nodes must be chosen for each VNF that composes that SFC, considering the SLA delay restrictions.

III. A Novel Solution for VNF Placement in Dynamic Edge Environments

The dynamic nature of the Edge Environment imposes several challenges related to the availability of resources, mobility of users and service load over time. To tackle these challenges, we propose a dynamic placement, herein referred to as Dynamic Smart Placement (DSP). The proposed algorithm follows a dynamic and online approach that considers the arrival of SFC Requests at any time. Our proposal considers latency, capacity, and energy to select the most appropriate edge nodes to place the required VNF Instances derived from SFC Requests. We also address the mobility problem by providing a mechanism that moves VNFs to deal with service degradation caused by changes in user locations. Throughout this section, we highlight the major aspects of our proposal.

A. Online Placement Approach

VNF placement algorithms can be categorized as online or offline. In offline approaches, infrastructure/network operators gather inputs (e.g., service requests, the capacity of each available node), and make decisions to satisfy requests from multiple end-users, considering different constraints [5]. Offline placement decisions consider only the initial input data and remain static throughout system execution. In contrast, online approaches consider the variations that the environment entities suffer over time, such as changes in resource availability, unpredictable arrival of requests, performance fluctuations, and user mobility. These variations make placement decisions more complex since arriving SFC requests must be placed in an environment of constant change in which other VNFs are already placed and running.

Considering the unpredictable arrival of SFC requests in an online system, the immediate execution of the placement decision algorithm when a request arrives at the system might impose some limitations. First, the current state of the available resources is required at the moment of the placement since we must consider which nodes can execute new VNF Instances without inducing a node overload. Therefore, data regarding currently available resources must be periodically collected by a monitoring entity. Second, the arrival frequency of the requests might impose an overhead on the entity responsible for the placement decision. Finally, offline approaches have the advantage of knowing the batch of requests that must be placed. Thus, individual placement decisions can be made by considering the whole batch of requests to increase profit or reduce resources. This advantage is lost if we consider the immediate execution of the placement algorithm when a request arrives, leading to extra use of resources, a lower number of placed requests and a smaller profit.

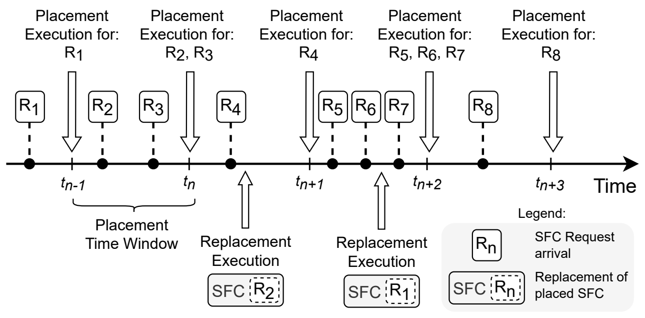


Fig. 1. Example of a timeline containing SFC Requests arrival, placement events and replacement events over time.

To deal with these limitations, we propose a time window approach to group incoming requests. In this approach, rather than executing the placement immediately when a request arrives, the placement execution will occur at a determined time interval. Requests that arrive at any given time within the current time window will be grouped in a batch of requests that will be considered for placement at the end of the time window. The interval between placement executions (*i.e.*, time window size) can be adjusted when necessary. Figure 1 shows an example of the behavior of the time window grouping strategy. In the example, requests R2 and R3 arrive at times within the interval between t_{n-1} and t_n. These requests are grouped and will be considered for placement at the next placement execution at time t_n.

The proposed strategy allows dealing with the unpredictable arrival of SFC Requests effectively. It reduces the need for frequent sampling of available resources by the monitoring entity since this information will only be required at a predictable interval, prior placement. Also, the strategy allows taking advantage of having a set of requests to be placed at the same execution of the placement solution. For instance, the placement decision could be made considering the joint requirements of a batch of requests, instead of deciding over a single request (*i.e.*, the decision regarding the placement of one request would be impacted by the needs of other requests in the batch), optimizing resource utilization. The configurable time window size provides great flexibility. It can have a fixed value and can be adjusted according to the characteristics of the environment. For example, smaller time intervals can be used to reduce the time in which a request waits to be placed, whereas greater intervals can reduce monitoring overhead and improve resource utilization.

B. Instance Sharing Policies

The proposed placement algorithm provides multiple sharing policies to reduce resource utilization when service loads are lower than their requested demand. These increase resource optimization by enabling allocated CPU, RAM, and link resources to be used by more than one VNF when requests with matching SFCs, or VNFs types arrive at the system. Before instantiating new VNF / SFC Instances, our placement approach tries to match requests to instances that are already running in the system. We describe each available sharing policy as follows.

In the VNF Sharing policy, the resources of a VNF Instance can be shared between multiple services when their requests require VNFs of the same type, or when the same service requires more than one VNF of the same type. In this policy, the maximum sharing, *i.e.*, the limit of VNF requests that will be served by the VNF Instance, is static and defined by the administrator. In VNF Sharing, the amount of allocated resources is divided by the number of services that share the VNF Instance, allowing multiple packets to be simultaneously processed with a portion of the resources. In the SFC Sharing policy, the resources of an ongoing SFC Instance can be shared between multiple services when SFC Requests with the same SFC type of the SFC Instance arrive at the system. Since an SFC Instance is composed of chained VNFs, this means that the resources of the VNFs that compose the SFC Instance are also shared with other SFCs of the same type. However, the sharing of resources of such VNFs does not count towards the sharing limit of the VNF Sharing policy. In this policy, the delay of packets that traverse a shared SFC Instance is monitored to avoid further sharing of resources of an SFC Instance that is already overloaded. When packet SLA violations occur in the service(s) that uses(use) an SFC Instance, the sharing flag is disabled for the particular SFC Instance, meaning that upcoming SFC Requests cannot be mapped to the SFC Instance anymore.

These sharing policies can also be disabled so that the resources of a VNF Instance or SFC Instance cannot be shared between multiple requests. In this case, resources allocated for a VNF Instance or SFC Instance are only used by a single VNF, or an SFC Request.

C. Latency, Energy and Capacity Aware Placement

The placement algorithm is responsible for finding suitable nodes that can execute the VNF Instances and implement the SFCs, as requested by the users. Despite finding the nodes that have enough resources to host the needed VNFs, the proposed placement approach also allows the prioritization of different aspects when choosing the nodes that will form the VNF chain and provide the requested service. Three prioritization criteria can be selected: (i) reduce energy consumption; (ii) reduce CPU and RAM utilization in the edge nodes; and (iii) reduce service latency.

This approach provides better flexibility and control of the placement decisions according to the main goals of the system stakeholders. The first criterion chooses edge nodes that consume least amount of energy among the ones that are capable of executing the VNF. The second criterion chooses nodes that have a larger amount of free CPU and RAM resources, avoiding utilization overhead and better balancing load among the edge nodes. The third criterion selects nodes to form a VNF chain with lower latency chain, which is helpful to fulfill the QoS requirements of services with stringent time constraints.

These prioritization criteria are also applied for VNF Instance sharing purposes. For each VNF to be placed to form an SFC, two situations can occur, namely, no VNF Instance already running at the edge environment can serve the request, or there are one or more VNF instances capable of handling the related SFC Request. In the former occasion, an edge node will be chosen according to the prioritization criteria, and a new VNF Instance will be created on the selected node. In the latter, when multiple VNF Instances are able to handle the VNF related to the SFC Request, the prioritization will be applied to choose the best node — among the ones that are executing those VNF Instances — selecting the best one according to the aforementioned criteria. Then, the VNF Instance of the node that best meets the criteria will be mapped to VNF, sharing its resources. If there is only one VNF Instance capable of handling the VNF related to the SFC Request, then the VNF will be mapped to the available VNF Instance.

D. Mobility Support

As users move to a new connection point, their services might experience higher delays due to the distance between the user and the SFC Instance or the shortage of available resources on the path that connects the user to the service. Consequently, the VNF chain initially established to serve the request might not be the best one for the new user location. To deal with the dynamics of user mobility, the SFC Replacement algorithm is in charge of transferring the ongoing service to a more suitable chain considering the new user location.

Upon detecting the service degradation and user movement, as depicted in Figure 2, the algorithm invokes the SFC Placement to find a suitable chain for the SFC Request. Then, the replacement cost is calculated based on a function that maps the spare resources and financial cost needed to replace the SFC Request. An estimation of the performance gains achieved by the replacement is also made to evaluate if the replacement is worth it or if it is better to keep the original

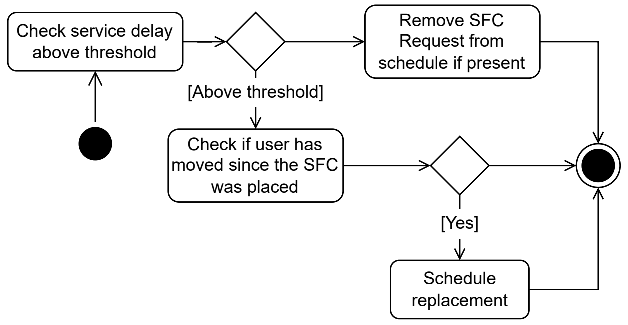

Check service delay above threshold
Remove SFC Request from schedule if present
[Above threshold]
Check if user has moved since the SFC was placed
[Yes]
Schedule replacement

Fig. 2. Diagram of the movement / deterioration monitoring of the replacement procedure.

SFC Request placement plan. During the replacement procedure (detailed in Figure 3), a better placement plan for the corresponding VNFs might not be available due to insufficient network resources. In this case, the replacement procedure exponentially schedules a retry, limited by a maximum number of retries.

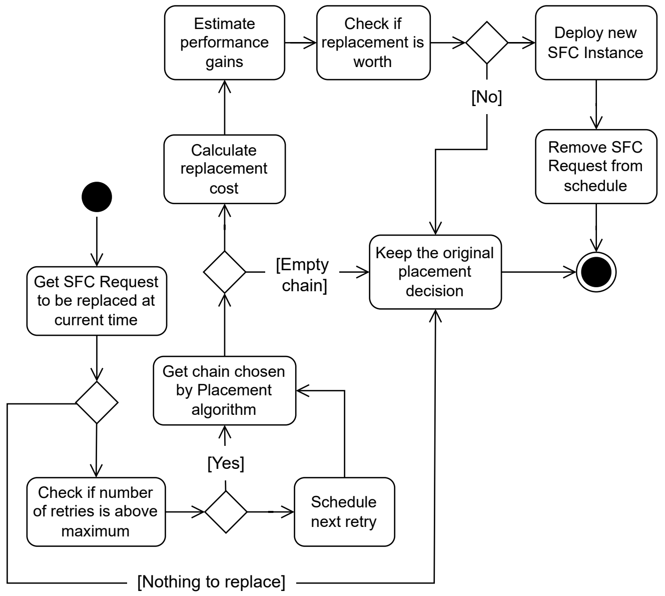

Estimate performance gains
Check if replacement is worth
Deploy new SFC Instance
[No]
Calculate replacement cost
Remove SFC Request from schedule
[Empty chain]
Keep the original placement decision
Get SFC Request to be replaced at current time
Get chain chosen by Placement algorithm
[Yes]
Check if number of retries is above maximum
Schedule next retry
[Nothing to replace]

Fig. 3. Diagram of the execution of the replacement.

As previously stated, the placement algorithm is periodically executed at the beginning of each time window. However, as depicted in Figure 1, the replacement algorithm does not wait for the next time window to replace an SFC Request. As soon as the service performance degradation reaches a specified threshold, the replacement procedure will be called for the corresponding SFC. Thus, we provide a solution where the SFC Request not meeting the SLA is replaced as soon as it fails.

E. System Model

Our work considers the resource allocation problem related to incoming SFC Requests that must be placed according to their service requirements and node available resources. We assume that SFCs will be instantiated in an underlying infrastructure that provides computing and networking resources.

TABLE I
SYSTEM MODEL NOTATIONS

Notation	Description
$G = (H, L)$	Graph that represents the underlying infrastructure
$H = \{h_1, h_2, \ldots, h_n\}$	Set of computing nodes (*e.g.*, edge devices)
$L = \{l^1_{h_1,h_2}, \ldots, l^k_{h_n,h_m}\}$	Set of virtual links
$d^k_{h_n,h_m}$	Transmission delay of link k between h_n and h_m
ca_h	Available CPU at node h
ma_h	Available memory at node h
$IM_h = \{vi_1, vi_2, \ldots, vi_n\}$	Set of VNF images available at node h
v_j	j-th VNF Type
$Img(v_j)$	VNF Image of VNF Type v
$D_c(v_j)$	CPU demand of VNF Type v
$D_m(v_j)$	RAM demand of VNF Type v
SFC_i^{req}	i-th SFC Request
SFC_i^{inst}	i-th SFC Instance
$T(SFC_i^{req})/T(SFC_i^{inst})$	SFC Type associated to an SFC Request / SFC Instance
$Ig(SFC_i^{req})/Ig(SFC_i^{inst})$	Ingress node associated to an SFC Request / SFC Instance
$Eg(SFC_i^{req})/Eg(SFC_i^{inst})$	Egress node associated to an SFC Request / SFC Instance

The underlying infrastructure is represented as an undirected graph $G = (H, L)$ that is composed of a set of computing nodes $H = \{h_1, h_2, \ldots, h_n\}$ connected through a set of virtual links $L = \{l^1_{h_1,h_2}, l^2_{h_1,h_2}, \ldots, l^k_{h_n,h_m}\}$. Each node h has an available amount of CPU and RAM resources, denoted as ca_h and ma_h, reported by the monitoring entity. Each virtual link $l^k_{h_n,h_m}$ is an abstraction of a path composed of physical links that connect node h_n to node h_m. Under the assumption that multiple paths might be available between two nodes, we consider that, for each pair of nodes, there is a set of k virtual links between them in the G graph that, despite having the same source and destination, have different transmission delays $d^k_{h_n,h_m}$.

Regarding the available functions, we consider that a set of VNF Types is available, where each type provides a different function (*e.g.*, firewall, image processing). Each VNF Type v_j demands an amount of CPU and RAM resources, denoted by $D_c(v_j)$ and $D_m(v_j)$. Also, clients can request SFCs based on the available SFC Types, where each type provides a specific kind of service (*e.g.*, video streaming) and is implemented as an ordered chain of VNF Types – *i.e.*, each SFC Type contains an ordered set of VNF Types that are required by the SFC. Each node h has a list of VNF images IM_h, each image corresponding to a different VNF Type, which can be instantiated at the node.

At the end of each time window, we consider that the placement algorithm receives a batch of SFC Requests that have arrived within the window interval, with each request denoted by SFC_i^{req}. Each SFC Request contains the service requirements defined in the SLA, the SFC Type to be instantiated, an ingress node, an egress node, and the user that will be served. The placement of an SFC Request can result in the creation of a new SFC Instance (SFC_i^{inst}) and their corresponding VNF Instances. When sharing is possible, an SFC Request can be mapped to an SFC Instance that is already running and serving other requests. VNF Instances can also be shared. The SFC Instance is a logical structure that keeps track of the VNF Instances that are being used to process the users' packets and the route that the packets should traverse. Each VNF Instance is executed at one node and responsible for providing one or several functionalities, processing packets related to mapped services. Table I summarizes the system model notations.

F. Placement Algorithms

As mentioned previously, the placement procedure is executed at the end of each time window. The algorithm receives as input the batch of SFC Requests that arrived within the last time window interval. The SFC Placement algorithm can be configured to choose a placement plan that prioritizes service latency, energy consumption, or the capacity of each node. It also supports sharing of SFC and VNF Instances to reduce overall resource consumption. The remainder of this section details the behavior of Algorithms 1 and 2.

Algorithm 1 is responsible for mapping the SFC Request to new or existing SFC Instances. First, the algorithm checks if the SFC Request can be mapped to an SFC Instance that is already executing. To share an SFC Instance, the SFC Request in question must have the same SFC Type, ingress, and egress node of the evaluated SFC Instance. Also, the sharing flag of the SFC Instance must be enabled. The state of the sharing flag can change according to the policies mentioned in Section III-B. If these are satisfied, the SFC Request will be mapped by sharing an executing SFC Instance. If not, a new SFC Instance will be created at the end with the chosen VNF Instances. To choose these instances, first the algorithm tries to explore VNF Instances that are already running in the environment. At this step, nodes that have the VNF Instance that corresponds to a VNF Type required by the SFC are gathered. Only VNF Instances that can be shared are selected, according to the VNF Instance sharing policy described in Section III-B. Then, Algorithm 2 is called to find the most appropriated node from those that already have a VNF Instance of the required VNF Type. If an appropriate node is found by Algorithm 2, the VNF Type is mapped to the VNF Instance running at the chosen node. If not, a new VNF Instance must be created. Therefore, the Algorithm 1 gathers nodes that have the necessary resources and VNF Image to execute a new VNF Instance. With the gathered nodes, Algorithm 2 is called to

Algorithm 1 SFC Placement Procedure - PlaceSFC()

```
1: function PLACESFC($SFC_i^{req}$)
2:     for $SFC_i^{inst}$ in getSFCInstances() do
3:         if $T(SFC_i^{req})$ = $T(SFC_i^{inst})$ and $Eg(SFC_i^{req})$ = $Eg(SFC_i^{inst})$ and isShareFlagTrue($SFC_i^{inst}$) then
4:             mapRequestToInstance($SFC_i^{inst}$, $SFC_i^{req}$)
5:             return success
6:         end if
7:     end for
8:     for each $v_j$ in $SFC_i^{req}$ do
9:         for $h$ in $H$ do
10:            if $h$.hasVNFInstanceOfType($v_j$) then
11:                $C \leftarrow h$
12:            end if
13:        end for
14:        $chosen_{node} \leftarrow$ ChooseVNFNode($C$,$SFC_i^{req}$,$v_j$)
15:        if $chosen_{node}$ is not NULL then
16:            $chosen_{node}$.mapRequestToVNF($v_j$)
17:            continue
18:        end if
19:        $C \leftarrow \emptyset$
20:        for $h$ in $H$ do
21:            if $ca_h > D_c(v_j)$ and $ma_h > D_m(v_j)$ and $Img(v_j)$ in $IM_h$ then
22:                $C \leftarrow h$
23:            end if
24:        end for
25:        $chosen_{node} \leftarrow$ ChooseVNFNode($C$,$SFC_i^{req}$,$v_j$)
26:        if $chosen_{node}$ is not NULL then
27:            $chosen_{node}$.createVNFInstance($v_j$)
28:        else
29:            return fail
30:        end if
31:    end for
32:    return success
33: end function
```

find the most appropriate node from those that can run a new VNF Instance. If at least one node is returned by Algorithm 2, then the VNF Instance is created at the node. This procedure is repeated for each VNF Type required by the SFC Request. The SFC Request is considered placed when the mapping/creation of each VNF Instance is completed successfully or the SFC Request is mapped to an already executing SFC Instance. Otherwise, the placement returns a failure event.

Algorithm 2 receives a list of candidate nodes that have enough resources to execute the VNF Instance, the SFC Request and a required VNF Type. It first gets the previous node of the SFC chain, which can be the ingress node when the VNF Type corresponds to the first required VNF or the node where the previous VNF was mapped. Then, it finds the most appropriated nodes considering the defined order of priority criteria, as mentioned in Section III-C. If latency is the first priority criterion, for example, the algorithm will get the nodes that have the smaller transmission delay, considering

Algorithm 2 VNF Placement Procedure - ChooseVNFNode()

```
1:  function ChooseVNFNode($C$,$SFC_i^{req}$,$v_j$)
2:      if index($v_j$) $= 0$ then
3:          previous $\leftarrow Ig(SFC_i^{req})$
4:      else
5:          previous $\leftarrow$ getNodeByVNFInstance($v_{j-1}$)
6:      end if
7:      for priority in SFCPriorityOrder($SFC_i^{req}$) do
8:          if priority = Latency then
9:              $C \leftarrow \arg\min_{c \in C}(\min_{k \in K} d^k_{previous,c})$
10:         else if priority = Capacity then
11:             $C \leftarrow \arg\max_{c \in C}(ca_c + ma_c)$
12:         else if priority = Energy then
13:             $C \leftarrow \arg\min_{c \in C}(e_c)$
14:         end if
15:         if $|C| = 1$ then
16:             break
17:         end if
18:     end for
19:     if $|C| = 0$ then
20:         return NULL
21:     end if
22:     if $|C| > 1$ then
23:         $chosen_{node} \leftarrow$ getRandomNode($C$)
24:     else
25:         $chosen_{node} \leftarrow$ getFirstNode($C$)
26:         return $chosen_{node}$
27:     end if
28: end function
```

all virtual links that connect the previous node to the list of nodes received as input. When more than one node has the smallest delay value, the second priority criterion is used. If the second criterion is energy, then the node that has the lowest energy consumption will be selected to break the latency tie. In the case that there is still more than one node after using all priority criteria (*i.e.*, multiple candidate nodes have the same capacity, energy consumption, and associated delays), a random candidate node is chosen.

IV. Evaluation

In this section, we evaluate the proposed approach regarding monitoring overhead reduction, resource utilization optimization, and mobility support. To this end, we implemented our proposal on a simulated environment based on the SimPy [15] framework. In the simulation, requests arrive following a Poisson arrival process and are considered for placement at the end of each time window.

In our proposal, we consider that users might switch access nodes while a requested service is being provided. To deal with the impact caused by user mobility, we defined a strategy for the relocation of SFCs to near the user, when user movement and performance issues are detected. Therefore, to investigate the potential of the proposed replacement mechanism, we held an experiment analyzing the average percentage of SLA violations per packet when the replacement strategy is enabled or disabled. In this experiment, 50 users — among static users and moving ones that traverse between access nodes — request one service each. User movement information was generated based on samples of the dataset in [16]. Each service is composed of 3 VNFs, that are placed in an environment of 32 edge nodes. Regarding the replacement triggering, we also considered that executing the replacement is always worth it. The experiment was repeated 50 times.

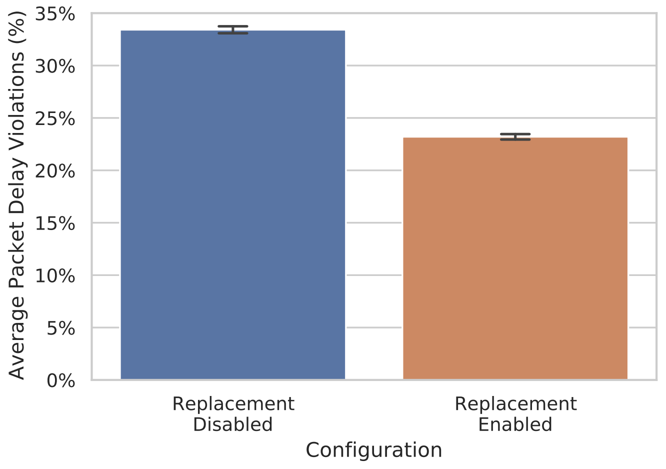
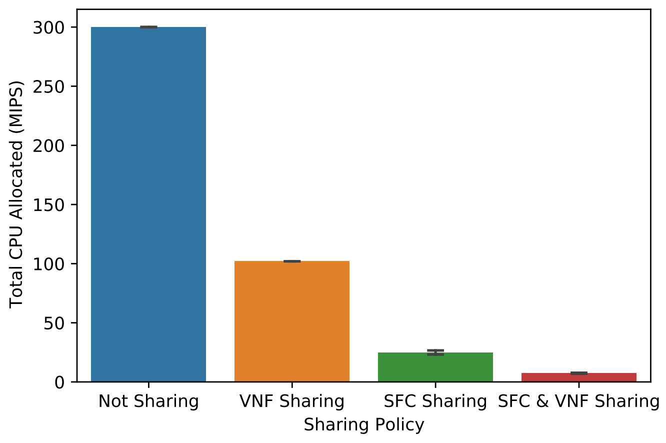


Fig. 6. Average percentage of packets whose processing delay exceeded the maximum established in SFC's SLA, considering only packets of services that have undergone a replacement. The confidence interval is 95%.

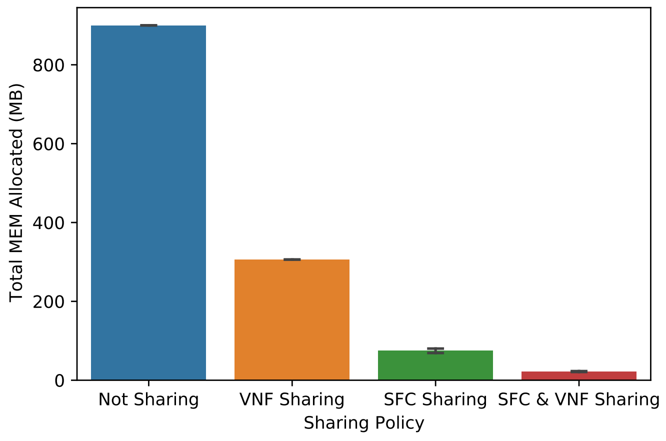
Figure 6 shows the obtained results of the experiment that analyzes the influence of the replacement on packet delay violations with and without the replacement procedure. To highlight the effectiveness of the replacement, the graph shows the average percentage of packets that were processed in an interval greater than the maximum service delay agreed only for the services that were replaced when the replacement is enabled (*i.e.*, packets from services that were not replaced in the replacement enabled scenario were not considered for both scenarios). When the replacement is disabled, the average packet delay violation is around 33%. Enabling the replacement procedure reduces the average packet delay violation to approximately 23%, which represents a reduction of around 30%.

To analyze the influence of the sharing mechanisms in terms of resource allocation and compliance with the maximum delay established at each SFC's SLA, we held a second experiment considering different sharing policies in place. The environment was configured to have enough resources for the allocation of all SFC Requests. The experiment was executed 50 times with varying CPU and memory resources in each round. One hundred requests arrive within 10 seconds of simulation and must be placed using the resources of 16 edge nodes. In this setup, we used a maximum VNF Instance sharing parameter of 3 (except in the Not Sharing policy).

Figure 7 shows the total amount of CPU and memory allo-



(a) CPU Allocation



(b) Memory Allocation

Fig. 7. Comparison of each sharing policy in terms of total CPU and memory allocated. The confidence interval is 95%.

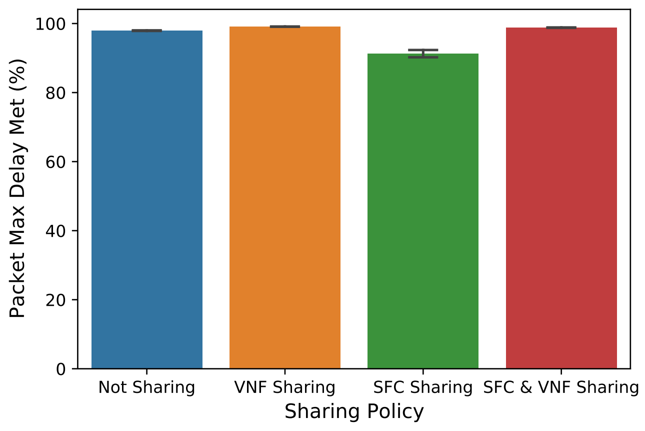


Fig. 8. Average percentage of packets that met the maximum service delay in each sharing policy. The confidence interval is 95%.

cated for each sharing policy. As expected, a greater number of resources were allocated in the Not Sharing scenario, as this policy allocates dedicated resources for each SFC Request. The VNF Sharing policy allocated around one-third of the resources in comparison to the Not Sharing, since each VNF was set to be shared at most three times (*i.e.*, VNF Max Share has a value of 3). The SFC Sharing further reduces resource utilization since the whole SFC is shared, as opposed to a single VNF. The combination of both sharing policies (SFC & VNF Sharing) results in the scenario where the least number of resources is allocated.

Regarding the impact on SLA compliance, Figure 8 shows the percentage of packets that were processed by the SFC within the maximum service delay. In the Not Sharing policy, most packets were processed within the maximum delay. However, some packets were processed in a time that exceeded the delay limit processing, even with dedicated resources. In contrast, the VNF Sharing and SFC & VNF Sharing scenarios allowed all packets to be processed without violating the maximum delay, even with sharing resources. This behavior can be explained by the fact that the VNF Sharing mechanism allows multiple packets to be simultaneously processed using the allocated resources, which reduces the time that a packet waits in the queue. In addition, the sharing mechanisms produce placement mappings in nodes that are closer to each other since services are often held by the same instance (and node). When new allocations are necessary, they are made in nearby nodes, whereas in the Not Sharing mechanism, the need for dedicated allocations depletes available resources with fewer requests, and new requests use resources of other nodes. This makes the path of nodes that compose the VNF chain longer for the Not Sharing strategy and shorter for the VNF Sharing. In longer chains, packet delays are higher because they have to traverse more links. Finally, the SFC Sharing policy, despite having more violations than the other three scenarios – because the SFC Instance sharing is stopped only when violations are detected – was able to process around 90% of the packets within the maximum delay.

To investigate the effectiveness of the time window size in the reduction of the monitoring overhead, we held a third experiment with parameters similar to the previous experiment. One hundred requests arrive continuously within the first 10 seconds of the simulation, and a total of 16 edge nodes are available at the underlying infrastructure with resources sufficient to allow the placement of all incoming requests.

To model the monitoring overhead, one edge node is randomly chosen as a monitoring node. At the end of each time window, – *i.e.*, prior to the execution of the placement – the monitoring node sends probes to other nodes in the edge environment, requesting the report of their available resources. When the monitoring node receives the information from all other nodes, the available resources in each of them are known, and the placement can be executed. The probe packet has a size of 500 bytes, while the response has a size of 1000 bytes. We analyzed the monitoring traffic, *i.e.*, the sum of the size of each monitoring packet, and the average request wait time – which is the interval between the placement execution start and the request arrival – with different time window sizes. Each of the 100 executions has a duration of 12 seconds.

Table II shows the average monitoring traffic and request wait time of the executions. As it can be seen, there is a

TABLE II
TIME WINDOW EXPERIMENT RESULTS. THE CONFIDENCE INTERVAL IS 95%.

Time Window Size (ms)	Average Total Monitoring Traffic (MB)	Average Request Wait Time (ms)
1	1.335 ± 0.004	12.075 ± 0.021
100	1.005 ± 0.005	48.776 ± 0.421
500	0.300 ± 0.001	266.401 ± 1.916
1000	0.150 ± 0.001	519.182 ± 3.927

tradeoff between monitoring overhead and the wait time of the request. Bigger time window sizes significantly reduce the monitoring overhead since the probing events occur less frequently, whereas the request wait time increases. On the other hand, smaller time window sizes reduce the interval that an SFC Request must wait to be placed — as the placement executions will occur more often — but incurs higher monitoring traffic.

V. CONCLUSION

The VNF placement problem consists of selecting the most suitable edge nodes to deploy and execute VNFs meeting application requests from multiple users. This must be done in such a way that the application QoS requirements are properly fulfilled while trying to optimize resource utilization and energy consumption at the edge tier. The VNF placement problem is NP-hard. Thus, optimal approaches such as MIP (Mixed-Integer Programming) can find feasible solutions only for small-scale edge systems, presenting poor scalability.

To address large-scale instances of the problem, in this paper we propose a suboptimal, scalable approach for the VNF placement that finds and selects the appropriate nodes to host the VNFs of an SFC. We also consider a challenging, dynamic, and heterogeneous environment where SFCs with different requirements must be placed in an infrastructure with varying resources. The dynamism stems from the continuous arrival of SFC requests over time and the user mobility.

A novel aspect of our proposal is the mechanism for sharing not only VNF but also SFC instances. This allows multiple SFC Requests to be handled by the same SFC Instance, consequently providing a high degree of reuse of the resources allocated to run VNFs. Our solution also supports user mobility through a replacement strategy. Considering an already fulfilled SFC Request, the mobility of a user related to the request triggers a replacement depending on the measured deterioration of the service. The replacement procedure finds a suitable VNF chain considering the new location of the user, reducing the user mobility impact on the service performance.

Performed experiments showed that the proposed approach reduces resource utilization through the sharing of same-type instances, with little impact on SLAs. Also, the replacement procedure reduces delay violations under user mobility conditions. Finally, grouping requests in a time window can reduce the monitoring overhead but with a straightforward tradeoff in the placement wait time of the requests, depending on the size of the time window.

As a future work, we intend to further analyze the influence of the time window strategy in terms of the number of successfully placed requests when an ILP model is used, since the knowledge of the request batch to be placed in advance could render better solutions, compared to the sequential placement approach of the current heuristic. We also intend to change our heuristic to benefit from this request grouping behavior.

ACKNOWLEDGMENT

This work was partially funded by DELL EMC, CAPES, CNPq, FAPERJ and FAPESP (grant number 2015/24144-7). Professors Débora Muchaluat-Saade, Flavia Delicato and Paulo Pires are CNPq Fellows. Professors Débora Muchaluat-Saade and Flavia Delicato are also FAPERJ Fellows.

REFERENCES

[1] B. Yi, X. Wang, K. Li, M. Huang *et al.*, "A comprehensive survey of network function virtualization," *Computer Networks*, vol. 133, pp. 212–262, 2018.

[2] H. U. Adoga and D. P. Pezaros, "Network function virtualization and service function chaining frameworks: A comprehensive review of requirements, objectives, implementations, and open research challenges," *Future Internet*, vol. 14, no. 2, p. 59, 2022.

[3] S. Yang, F. Li, S. Trajanovski, R. Yahyapour, and X. Fu, "Recent advances of resource allocation in network function virtualization," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 2, pp. 295–314, 2020.

[4] I. Benkacem, T. Taleb, M. Bagaa, and H. Flinck, "Optimal VNFs placement in CDN slicing over multi-cloud environment," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 3, pp. 616–627, 2018.

[5] N. Reyhanian, H. Farmanbar, S. Mohajer, and Z.-Q. Luo, "Joint resource allocation and routing for service function chaining with in-subnetwork processing," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2020, pp. 4990–4994.

[6] Y. Guo, Q. Duan, and S. Wang, "Service orchestration for integrating edge computing and 5g network: State of the art and challenges," in *2020 IEEE World Congress on Services (SERVICES)*. IEEE, 2020, pp. 55–60.

[7] A. Tomassilli, F. Giroire, N. Huin, and S. Pérennes, "Provably efficient algorithms for placement of service function chains with ordering constraints," in *IEEE Conference on Computer Communications*, 2018, pp. 774–782.

[8] G. Sallam and B. Ji, "Joint placement and allocation of virtual network functions with budget and capacity constraints," in *IEEE Conference on Computer Communications*, 2019, pp. 523–531.

[9] Z. Xu, W. Liang, A. Galis, and Y. Ma, "Throughput maximization and resource optimization in nfv-enabled networks," in *IEEE International Conference on Communications (ICC)*, 2017, pp. 1–7.

[10] J. Zhang, Z. Wang, N. Ma, T. Huang, and Y. Liu, "Enabling Efficient Service Function Chaining by Integrating NFV and SDN: Architecture, Challenges and Opportunities," *IEEE Network*, vol. 32, no. 6, pp. 152–159, 2018.

[11] D. Harutyunyan, N. Shahriar, R. Boutaba, and R. Riggio, "Latency-aware service function chain placement in 5G mobile networks," in *IEEE Conference on Network Softwarization (NetSoft)*, 2019, pp. 133–141.

[12] ——, "Latency and mobility-aware service function chain placement in 5g networks," *IEEE Transactions on Mobile Computing*, 2020.

[13] G. Zheng, A. Tsiopoulos, and V. Friderikos, "Dynamic VNF chains placement for mobile IoT applications," in *IEEE Global Communications Conference*, 2019, pp. 1–6.

[14] Y. Ma, W. Liang, J. Li, X. Jia, and S. Guo, "Mobility-aware and delay-sensitive service provisioning in mobile edge-cloud networks," *IEEE Transactions on Mobile Computing*, vol. 21, no. 1, pp. 196–210, 2020.

[15] "SimPy - Discrete event simulation for Python," https://simpy.readthedocs.io/, Accessed in 2023-01-04.

[16] M. Pielot, B. Cardoso, K. Katevas, J. Serrà, A. Matic, and N. Oliver, "Beyond interruptibility: Predicting opportune moments to engage mobile phone users," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 1, no. 3, pp. 1–25, 2017.