Network Slicing in 5G Edge Networks with Controlled Slice Redistributions

Samaresh Bera, *Member, IEEE* and Neelesh B Mehta, *Fellow, IEEE*

***Abstract*—Edge computing with network slicing enables 5G networks to meet the diverse and stringent requirements of new services and applications. We study the problem of admitting network slice requests and serving currently active slices in the 5G edge network, while avoiding active slice redistributions in the network. We pose it as a constrained optimization problem that seeks to maximize the total reward or revenue to the network operator from serving the new and active slices minus a term that penalizes slice redistributions in the network. We propose a three-phase polynomial-time, greedy, heuristic approach called `RESET` to solve this NP-hard problem. The first phase employs a cost function that determines the order in which new slice requests and a fraction of the active slices are served. It takes into account the bandwidth, storage, and computing requirements of a slice request. The second phase selects an edge cloud (EC) to assign an admitted request based on the residual bandwidth of its incoming links, and storage and computing resources at the EC. The last phase determines the forwarding path to route the traffic associated with the slice request to the selected EC. Extensive simulation results show that `RESET` achieves a total reward that is competitive with the optimal solution and is higher than benchmark schemes, while requiring far fewer slice redistributions.**

***Index Terms*—5G, Network slicing, Edge network, Resource allocation, Redistribution, Quality-of-service, Optimization**

I. Introduction

5G communications has led to the emergence of new services and applications that are broadly categorized as enhanced mobile broadband (eMBB), ultra reliable low latency communications (uRLLC), and massive machine type communications (mMTC). This includes applications such as video streaming with high throughput, virtual reality with low latency, and factory automation with high reliability. These new services and applications have diverse and stringent quality-of-service (QoS) requirements that range from high bandwidth to ultra-low latency and high reliability. The bandwidth requirements of these services range from 1 Mbps to 1 Gbps, the latency requirements range from 10 to 100 ms, and the reliability guarantees range from 99.999% to 99.9999999% [1], [2]. Network operators can no longer support such diverse requirements using the traditional 'one-size fits all' network architecture.

Network slicing addresses this challenge. It creates multiple logical networks that utilize the same physical network infrastructure [3]. Consequently, network operators can create and deploy network slices dynamically depending on the requirements of the slice requests that arrive. This flexibility has motivated a lot of work on resource management and slicing at 5G radio access network (RAN), and transport and core networks [4], [5]. Recently, network slicing has been combined with edge computing to meet the demanding key performance indicators of 5G networks [6], [7].

With the integration of the edge computing framework, the 5G network typically comprises of fronthaul, midhaul, and backhaul networks [8], as depicted in Figure 1. The fronthaul network refers to the 5G RAN, and the backhaul network refers to the 5G core network. The midhaul network refers to the 5G edge network that comprises of layer two and layer three networking devices and edge clouds (ECs). The links between the networking devices provide bandwidth to route requests in the network. The ECs provide networking resources such as computing and storage. Service requests from users are either served by an EC or forwarded to the backhaul/core network.

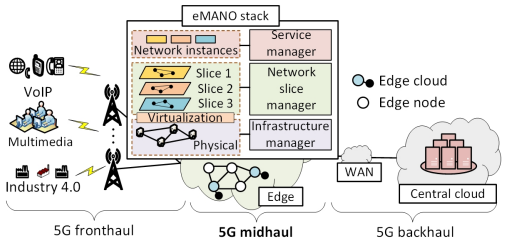

Fig. 1: 5G network slicing architecture consisting of an eMANO stack, fronthaul, midhaul, and backhaul. The midhaul consists of edge nodes and edge clouds.

In this work, we focus on 5G edge network slicing and resource management. We consider ETSI's network management and orchestration (MANO) framework [9] that consists of a service manager, a network slice manager, and an infrastructure manager for deploying network slices. A network slice consists of virtual network functions (VNFs) that serve the applications associated with the slice. For example, a network slice with a video streaming application consists of two VNFs – a video optimizer and a firewall. Each VNF requires a certain amount of storage and computing resources. Furthermore, each slice has a bandwidth requirement to route the slice-specific requests through the VNFs. At the edge network, we have an edgeMANO (eMANO) that places the VNFs at the ECs

S. Bera and N. B. Mehta are with the Department of Electrical Engineering, Indian Institute of Science, Bangalore, 560012, India, Email: s.bera.1989@ieee.org, nbmehta@iisc.ac.in. This work was partially supported by the Centre for Networked Intelligence and the C. V. Raman Postdoctoral Fellowship, IISc Bangalore, India.

and orchestrates the network slices taking into account the available network resources and slice-specific requirements.

A few works have recently focused on 5G edge network slicing [10]–[12]. These works focus primarily on reward/revenue maximization given the network resources and slice requirements. However, they do not take into account the issues associated with active slice redistribution or migration of their VNFs from one location to another when new service requests arrive. Specifically, an active slice may need to be redistributed to free up resources at a bottleneck EC or link when a new request arrives to maximize the total reward or revenue of a network operator.

A slice redistribution entails significant operational costs [13]. First, migrating VNFs associated with an active slice takes time and consumes storage. Second, both stateful and stateless migrations of VNFs pose significant challenges due to the above migration costs and inter-state dependency of the VNFs, respectively. Third, maintaining the network connection during the migration is challenging as additional bandwidth is required to seamlessly migrate the VNFs from one location to another. Moreover, the network slice redistribution process also requires coordination between the network slice and infrastructure managers. Consequently, the cost associated with slice redistributions needs to be accounted for by a policy that determines whether to admit new slice requests and whether to redistribute some of the active slices to facilitate this. Another important to note is that the existing works assume that users are one-hop away from the ECs. However, in a practical edge computing framework, the users may be located multiple hops away from the ECs.

In this paper, we study edge network slicing as a reward maximization problem while introducing a penalty term that incentivizes the network orchestrator to reduce the number of slice redistributions. The objective function to be maximized is framed as the difference between a term that equals the total reward obtained by serving new and existing slice requests, while satisfying the network resource and slice-specific QoS constraints, and a penalty term that increases as the number of active slice redistributions increases. As the optimization problem turns out to be NP-hard, we propose a greedy, heuristic approach called `RESET` to solve the problem in polynomial time. `RESET` first sets aside a fraction of the active slices for possible redistribution or even dropping. These active slices and new requests are then admitted into the network using a procedure that consists of three phases: a) a request selection phase that orders and admits the new slice requests, b) an EC selection phase to assign an EC for a given slice request, and c) a forwarding path selection phase to select the route from the source to the selected EC. We present extensive simulation results that show that the proposed approach achieves a total reward that is competitive with the optimal solution and is higher than benchmark schemes, while requiring far fewer slice redistributions.

The rest of the paper is organized as follows. Section II discusses the existing works in the context of network slicing in 5G. Section III presents the detailed system model and the optimization problem. Section IV presents the proposed approach with algorithms. The performance of the proposed approach is benchmarked in Section V. Finally, we present our conclusions and some future research directions in Section VI.

II. Related Work

We now discuss works that focus on slicing and resource management at 5G edge networks [10]–[12], [14]. Castellano et al. [10] study resource allocation for network slices in an edge computing framework as a revenue maximization problem. They propose an online voting approach to assign network resources to the slices in a distributed manner. Liu and Han [11] propose a distributed resource orchestration scheme for network slicing in a cellular network. The authors consider radio resources at the RAN and computing resources at the edge servers in the network. The resource orchestrator manages these resources to facilitate cross-domain functional isolation between the network slices. Complete isolation between network services is assumed, i.e., the performance of one service request is assumed to be unaffected by other service requests in the network. However, in a practical scenario, the performance of one service is affected by other services in the network [12]. The authors in [12] propose an edge network slicing scheme to maximize revenue considering the dependencies between service requests in a 5G edge network.

The works closest to ours are [10], [12] as they also focus on maximizing the total reward or revenue. However, these works do not penalize active slice redistributions, and they assume that the users are one-hop away from the ECs and consider only single-hop routing to allocate bandwidth resources. On the other hand, we consider multi-hop routing for serving network slices, costs associated with slice redistributions, and bandwidth, computation, and storage constraints.

III. Network Model

5G Edge Network: We represent the 5G edge substrate network as an undirected graph $G(\mathcal{V}, \mathcal{L})$, where $\mathcal{V}$ is the set of edge nodes and $\mathcal{L}$ is the set of links between the nodes in the network. Each link $(i, j) \in \mathcal{L}$ has a bandwidth $B_{i,j}$ to forward traffic. Furthermore, some of the edge nodes offer computing and storage facilities. Such edge nodes are called ECs, as shown in Figure 1. The set of ECs is denoted as $\mathcal{V}^{'}$. Thus, the non-EC nodes, $\mathcal{V} \setminus \mathcal{V}^{'}$, simply forward the traffic associated with the slices being served by the network. We consider bandwidth, computing, and storage resources in the edge network, where each link is associated with a bandwidth, and each EC is associated with computing and storage resources. All the networking resources are virtualized.

Slice Request: A slice request is characterized by bandwidth, computing, and storage requirements. The computing and storage requirements are determined by the VNFs associated with the slice, examples of which are discussed in Section I. The resources required by a VNF depend on the network slice type. They are specified in terms of computing and storage requirements [15], and a bandwidth requirement for being routed in the network. Furthermore, each slice

request is also associated with a reward, a source edge node, and an active time, which is the duration for which it stays in the network. Mathematically, a slice request $r \in \mathcal{R}$ is represented as a six-tuple $(b^{[r]}, c^{[r]}, d^{[r]}, \zeta^{[r]}, s^{[r]}, \tau^{[r]})$, where $b^{[r]}$, $c^{[r]}$, and $d^{[r]}$ denote the bandwidth, computing, and storage requirements, respectively. The symbols $\zeta^{[r]}$, $s^{[r]}$, and $\tau^{[r]}$ denote the reward, source edge node, and active time of the request r, respectively. Slice requests can arrive at different and random times to the network.

A. Optimization Problem Formulation

Objective: We are given a 5G edge network that consists of edge nodes and links, each having a bandwidth resource, and ECs, each having computing and storage resources. Furthermore, we are given slice requests, each with bandwidth, computing and storage requirements, a reward, a source node, and an active time. Additionally, at any point in time, we are given the set of active slice requests that are currently being served by the network. The objective is to optimally allocate bandwidth, computing, and storage resources to the slice requests so that the total reward minus the penalty associated with slice redistributions is maximized while adhering to the constraints on network resources and slice-specific QoS requirements. In each time slot, the network orchestrator needs to determine which new slice requests to admit into the network and which active slices to redistribute to different ECs or to drop. Such redistributions can free up resources in bottlenecked ECs and forwarding links, and enable them to serve more requests, which increases the reward. Mathematically, the objective function P is given by

$$P = \max \Big\{ \overbrace{\sum_{r \in \mathcal{R}} \sum_{k \in \mathcal{V}'} \zeta^{[r]} x_k^{[r]}}^{\text{reward}} - \overbrace{\sigma \sum_{r \in \tilde{\mathcal{R}}} \sum_{k \in \mathcal{V}'} f(x_k^{[r]}, \tilde{x}_k^{[r]})}^{\text{redistribution penalty}} \Big\}. \quad (1)$$

The first term denotes the total reward obtained by admitting the slice requests. Here, $x_k^{[r]}$ is a binary variable that equals 1 when the VNFs associated with the slice request $r \in \mathcal{R}$ are placed at EC $k \in \mathcal{V}'$, and is 0 otherwise. The second term represents the total penalty associated with slice redistributions. The redistribution penalty, which applies only to active requests, is a function of the current placement, $x_k^{[r]}$, and the previous placement, $\tilde{x}_k^{[r]}$, $\forall r \in \tilde{\mathcal{R}}$, where the set $\tilde{\mathcal{R}}$ denotes the set of active slices in the network. We note that this formulation enables any penalty function to be used.

Constraints: The following constraints need to be satisfied.

$$\sum_{r \in \mathcal{R}} d^{[r]} x_k^{[r]} \leq D_k, \forall k \in \mathcal{V}' \quad (2a)$$

$$\sum_{r \in \mathcal{R}} c^{[r]} x_k^{[r]} \leq C_k, \forall k \in \mathcal{V}' \quad (2b)$$

$$\sum_{r \in \mathcal{R}} b^{[r]} y_{i,j}^{[r]} \leq B_{i,j}, \forall (i,j) \in \mathcal{L} \quad (2c)$$

$$\sum_{k \in \mathcal{V}'} x_k^{[r]} \leq 1 \quad (2d)$$

$$x_k^{[r]}, y_{i,j}^{[r]} \in \{0,1\} \quad (2e)$$

$$\sum_{j \in \mathcal{V}} y_{i,j}^{[r]} - \sum_{j \in \mathcal{V}} y_{j,i}^{[r]} = \begin{cases} \sum_{k \in \mathcal{V}'} x_k^{[r]}, \text{ if } i = s^{[r]} \in \mathcal{V} & (3a) \\ -x_k^{[r]}, \text{ if } i = k \in \mathcal{V}' & (3b) \\ 0, \text{ otherwise.} & (3c) \end{cases}$$

Each EC has a certain amount of storage to meet the storage required by the VNFs of the slice requests it hosts. Consequently, the storage capacity constraint is given in (2a). Each slice request hosted by an EC also requires computing resources based on the associated VNFs. Therefore, the computing constraint is given in (2b). Once a slice request is admitted, the traffic through the slice needs to be forwarded to the selected EC. Doing so utilizes the bandwidth on the links that constitute the route from source edge node to the EC. The link capacity constraint is given in (2c), where $y_{i,j}^{[r]}$ is a binary variable that is 1 when link $(i,j) \in \mathcal{L}$ is utilized to route the traffic associated with the slice request $r \in \mathcal{R}$, and is 0 otherwise. Each slice request $r \in \mathcal{R}$ is served by at most one EC at a time, as denoted by (2d). The assignment variables for EC selection, $x_k^{[r]}$, and link utilization, $y_{i,j}^{[r]}$, are binary, which are presented in (2e). The traffic in the substrate-slice network must follow the flow conservation rule for the outgoing and incoming flows in it [16], as presented in (3). The first case (3a) applies to the source $s^{[r]} \in \mathcal{V}$ of the slice request $r \in \mathcal{R}$. It states that the total outgoing traffic from the source $s^{[r]}$ is equal to the total traffic received by all the ECs in the network. Consequently, if none of the ECs can be assigned to serve the request r, i.e., $\sum x_k^{[r]} = 0, \forall k \in \mathcal{V}'$, the outgoing traffic from the source $s^{[r]}$ is zero. The second case (3b) states that the traffic received at an EC k is either one or zero depending on whether the request r is served by the EC. Finally, the third case (3c) states that the difference between the outgoing traffic and incoming traffic at an intermediate edge node $i \in \mathcal{V}$ is zero in order to satisfy the flow conservation rule.

The above optimization problem is a special case of the multi-constraint multiple knapsack problem, which is NP-hard. We refer the interested reader to [17] for details about the NP-hardness proof. Therefore, solving the optimization problem for a large number of slice requests and large networks is computationally prohibitive. In the next section, we propose a low-complexity, polynomial-time, greedy, heuristic approach called RESET.

IV. Proposed Approach: RESET

We solve the slice request admission and placement problem in three phases, namely, request selection phase, EC selection phase, and forwarding path selection phase. These address three fundamental questions: in which order should the slice requests be admitted, which EC should fulfil the computing and storage demands of a request, and which path should be used for forwarding traffic of the slice while satisfying its bandwidth requirements. Prior to these phases, we identify a fraction of the active slice requests for potential redistribution. We discuss the phases in detail below.

A. Identify Active Slices for Potential Redistribution

We first shortlist the active slice requests. The requests are ordered using a cost function, which we define next. The resources assigned to a pre-specified fraction $(1-\delta)$, where $0 \leq \delta \leq 1$, of the requests at the top of the ordered list are left untouched. The remaining requests are combined with the set of new requests, which means that their network slices can be redistributed or dropped. For ease of description, we shall refer to these slices as new slice requests as well. Here, δ is a system parameter. The larger the value of δ, the higher the reward but also the higher the redistribution penalty. We investigate the trade-offs associated with δ in Section V-A.

B. Request Selection

The request selection phase decides the order in which new requests are admitted to the network. Each request $r \in \mathcal{R}$ requires a bandwidth $b^{[r]}$, computing resources $c^{[r]}$, and storage resources $d^{[r]}$. We design the following weighted cost function that determines a cost, $\theta^{[r]}$, of r in terms of the resources it requires:

$$\theta^{[r]} = \alpha_{\text{req}} \frac{b^{[r]}}{b_{\max}} + \beta_{\text{req}} \frac{d^{[r]}}{d_{\max}} + \gamma_{\text{req}} \frac{c^{[r]}}{c_{\max}}, \tag{4}$$

where $\alpha_{\text{req}} \geq 0$, $\beta_{\text{req}} \geq 0$, and $\gamma_{\text{req}} \geq 0$ are predefined weights that accord different priorities to bandwidth, storage, and computing resource requirements, respectively, and $\alpha_{\text{req}} + \beta_{\text{req}} + \gamma_{\text{req}} = 1$. The constants $b_{\max}$, $d_{\max}$, and $c_{\max}$ denote the maximum bandwidth, storage, and computing requirements, respectively, which are calculated from the set of new slice requests.

The edge controller in the eMANO framework computes the ratio $\frac{\zeta^{[r]}}{\theta^{[r]}}$ for each new request r, where $\zeta^{[r]}$ is the reward. It sorts this ratio in the descending order. Then, starting from the request with the largest ratio, each request is sequentially checked to determine whether all its network resource and slice-specific QoS constraints, which are given in (2a)–(3c), can be met. If this is so, then the request is admitted. Else, it is dropped.

C. EC Selection

Each EC has computing and storage resources that it utilizes to place the VNFs associated with the slice requests it serves. Therefore, for a request r admitted from the sorted list generated by the request selection phase (cf. Section IV-B), the edge controller determines the *candidate* ECs, $\tilde{\mathcal{V}}' = f(\mathcal{V}', r)$, which can fulfil the required storage and computing resources. In case $\tilde{\mathcal{V}}' = \emptyset$, then the request is dropped. To select an EC k from $\tilde{\mathcal{V}}'$, we design a cost function ϕ_k as

$$\phi_k = \alpha_{\text{ec}} \frac{\sum_{(i,k)\in\mathcal{L}} B_{i,k}}{\sum_{(i,k)\in\mathcal{L}} B^{\text{res}}_{i,k}} + \beta_{\text{ec}} \frac{D_k}{D^{\text{res}}_k} + \gamma_{\text{ec}} \frac{C_k}{C^{\text{res}}_k}, \forall i \in \mathcal{V}, \tag{5}$$

where $\alpha_{\text{ec}} \geq 0$, $\beta_{\text{ec}} \geq 0$, and $\gamma_{\text{ec}} \geq 0$ are predefined constants that determine the relative importance of the bandwidth, storage, and computing resources, respectively, of the EC nodes, and $\alpha_{\text{ec}} + \beta_{\text{ec}} + \gamma_{\text{ec}} = 1$. Furthermore, $\sum_{(i,k)\in\mathcal{L}} B_{i,k}$ represents the total bandwidth of all incoming links to the candidate EC, $k \in \tilde{\mathcal{V}}'$. The constants $B^{\text{res}}_{i,k}$, D^{res}_k, and C^{res}_k denote the residual bandwidth of the link $(i,k) \in \mathcal{L}$, residual storage of EC k, and residual computing resources of EC k, respectively. Thus, an EC with lower residual resources is associated with a higher cost. The EC with the lowest cost is selected to place the VNFs associated with the slice request. This process starts with the admitted request at the top of the sorted list in the request selection phase and ends with the admitted request at the bottom of the list. As mentioned, active slices that can be redistributed are also considered. If the current assignment differs from the previous one, we determine the penalty as per (1), which depends on σ and the penalty function.

D. Forwarding Path Selection

Once an EC is selected for a given slice request, as described in Sections IV-B and IV-C, we search for a route from source edge node to the selected EC through which the traffic associated with the slice can be forwarded to satisfy the slice's bandwidth requirement. The forwarding path selection algorithm is presented in Algorithm 1. The eMANO first prunes all the links from the substrate network that do not satisfy the bandwidth requested by the slice request (refer to Steps 1 and 7). Hence, it obtains a residual substrate network. Then, Dijkstra's weighted shortest path algorithm is used to determine the forwarding path (refer to Step 3).

Algorithm 1 Forwarding path selection algorithm

Inputs: Graph, $\mathcal{G}(\mathcal{V}, \mathcal{L})$, with initial link capacity, $B_{i,j}$, and residual link capacity, $B^{\text{res}}_{i,j}$, $\forall (i,j) \in \mathcal{L}$;
Bandwidth demand $b^{[r]}$ and source $s^{[r]}$ of request r;
Selected EC: k

Output: Forwarding path from source $s^{[r]}$ to EC k

1: Residual Graph $\mathcal{G}'(\mathcal{V}, \mathcal{L}')$ = RESIDUAL_GRAPH($\mathcal{G}$, $b^{[r]}$)
2: link_weight$(i,j) = \frac{B_{i,j}}{B^{\text{res}}_{i,j}}$, $\forall (i,j) \in \mathcal{L}'$ ▷ link weight
3: path = Dijkstra_Shortest_Path($\mathcal{G}'$, $s^{[r]}$, k, link_weight)
4: **function** RESIDUAL_GRAPH($\mathcal{G}$, $b^{[r]}$)
5: **for** (i,j) in $\mathcal{G}.edges$ **do**
6: **if** $B^{\text{res}}_{i,j} < b^{[r]}$ **then** ▷ demand cannot be satisfied
7: **Prune** link (i,j) from $\mathcal{G}$ ▷ link is pruned
return Residual graph $\mathcal{G}(\mathcal{V}, \mathcal{L}')$

Algorithm 2 presents the proposed greedy heuristic algorithm for determining which slice requests to serve to maximize the total reward. In it, Steps 1 and 2 include a fraction of the active slice requests, which can be redistributed, into the set of new requests. Step 3 sorts the requests in the descending order based on the cost function in (4). Step 6 checks for the EC with the lowest cost using (5). Based on the path selection algorithm (refer to Algorithm 1), the request is admitted and the network capacity is updated. If a candidate EC or a forwarding path is not found, then the request is dropped. Finally, Steps 16–19 check for active slices that were redistributed and computes the penalty term. The computation time complexity of RESET is:

$$\underbrace{[O(|\mathcal{R}| + |\mathcal{R}|\log|\mathcal{R}|)]}_{\text{request selection}} + \Big[O(|\mathcal{R}|) \times [\underbrace{O(\mathcal{V}^{'}|\mathcal{L}|+|\mathcal{V}^{'}|)}_{\text{EC selection}} + \underbrace{O(|\mathcal{L}|+|\mathcal{V}|(|\mathcal{L}| + |\mathcal{V}|\log|\mathcal{V}|))}_{\text{path selection}}]\Big].$$

Algorithm 2 Greedy heuristic algorithm for reward maximization

Inputs: **Network:** $\mathcal{G}(\mathcal{V}, \mathcal{L})$ with $B_{i,j}$, $B_{i,j}^{\text{res}}$, $\forall (i,j) \in \mathcal{L}$; Set of ECs $\mathcal{V}^{'}$; Set of active requests $\tilde{\mathcal{R}}$ with $\tilde{x}_k^{[r]}$ $\forall r \in \tilde{\mathcal{R}}$; **Request:** Set of received requests $\mathcal{R}$;

Output: Assign slice requests to maximize total reward

$\tilde{\mathcal{R}}^{'} \leftarrow f(\tilde{\mathcal{R}}, \delta)$ ▷ active slices that can be redistributed
$\mathcal{R} \leftarrow \mathcal{R} \cup \tilde{\mathcal{R}}^{'}$
$\mathcal{R}^{'} \leftarrow \mathcal{R}.SortDecreasing()$ using $\frac{\zeta^{[r]}}{\theta^{[r]}}$
for request r in $\mathcal{R}^{'}$ **do** ▷ from sorted list
  $k \leftarrow$ EC with minimum cost using (5)
  if k **then** ▷ candidate EC found
    $path \leftarrow$ forwarding path using Algorithm 1
    if $path$ **then** ▷ forwarding path found
      $x_k^{[r]} = 1$ ▷ request is assigned
      total_reward ← total_reward + $\zeta^{[r]}$
      UPDATE_NETWORK($\mathcal{G}$, $path$, r, k)
    else
      Drop request r ▷ no forwarding path found
  else
    Drop request r ▷ no candidate EC found
for $r \in \tilde{\mathcal{R}}^{'}$ **do** ▷ check for active slice redistribution
  if check_redistribution($x_k^{[r]}$, $\tilde{x}_k^{[r]}$) **then**
    total_reward ← total_reward - $\sigma f(x_k^{[r]}, \tilde{x}_k^{[r]})$
    redistribution ← redistribution + 1
function UPDATE_NETWORK($\mathcal{G}$, $path$, r, k)
  for (i,j) in $path$ **do**
    $B_{i,j}^{\text{res}} \leftarrow (B_{i,j}^{\text{res}} - b^{[r]})$ ▷ link capacity updated
  $D_k \leftarrow (D_k - d^{[r]})$ ▷ storage resource updated
  $C_k \leftarrow (C_k - c^{[r]})$ ▷ computing resource updated

V. Performance Evaluation

We evaluate the performance of RESET and compare it with the optimal solution to ascertain its efficacy. Table I presents the parameters and their values used for evaluating the performance. We consider the AttMpls topology from the Internet topology Zoo [18] to create the edge nodes and links between them. We also consider the scale-free topology [19] to see the performance of RESET compared to the optimal solution. The results with the scale-free topology, which are not shown due to space constraints, are qualitatively similar. The ECs are selected as follows. The edge nodes are sorted in the descending order of their degrees. A fraction η of the nodes with the largest degrees in this sorted list are designated as ECs. As a real-dataset of network slicing is difficult to get, we set the values of network resources (bandwidth, computing, and storage) and slice-specific bandwidth requirements based on a careful study of the literature [12], [15], [20]. These values are generated randomly from the range specified in Table I. Furthermore, the slice-specific computing and storage demands are generated based on the associated VNFs [15]. As mentioned in Section III, the UPF requirements of different slices can be different. To generate the different UPF requirements, we categorize the UPFs associated with a slice request into two sets F1 and F2, as shown in Table I. The UPF in F1 is always selected for each slice request. For example, the video optimizer (VO) UPF is always present in all eMBB slice requests. Whereas, in F2, we select one to two UPFs randomly using the `random.sample()` method available in Python. The reward value and active time of a slice request are given in Table I. The values of the predefined constants for the cost functions for request selection and EC selection, and the redistribution penalty are also given in Table I. We use the following penalty function for performance evaluation:

$$f(x_k^{[r]}, \tilde{x}_k^{[r]}) = \left(x_k^{[r]} - \tilde{x}_k^{[r]}\right)^2. \tag{6}$$

Thus, the penalty increases as the number of slice redistributions increases.

We take an average of 20 runs to present the results with a 95% confidence interval [21]. The slice requests are generated using a Poisson process, whose arrival rates we vary. Altogether, we generate 500 slice requests in a single run. The new slice requests are considered for admission once every 10 seconds. The simulation results are reported in two phases: a) optimal vs. RESET in Section V-A; and b) benchmark schemes vs. RESET in Section V-B. We consider the following performance metrics to benchmark the performance of RESET: total reward, percentage of redistributions, percentage of admitted requests, and running time. The percentage of admitted requests is a metric of interest to service providers who wish to accommodate as many slice requests as possible in the network.

A. Results: Optimal vs. RESET

To get the optimal solution, we use the IBM-CPLEX solver [22]. The simulation is conducted in a machine with the Intel Xeon CPU and 128 GB RAM. Given the exponential complexity of the optimal solution, we consider only $\eta = 10\%$ of the total number of nodes in the network to be ECs. We show results when up to 5% and 10% of the active slices can be redistributed.

1) Reward: Figure 2 compares the total reward obtained using the optimal solution and RESET. We show results for two values of the penalty factor σ: 0 and 0.5. For both values of σ, we see that RESET is competitive with the optimal solution; its total reward is within 15–25% of that of the optimal solution. The total reward decreases as the arrival rate increases. This is because the number of time slots decreases as the arrival rate increases since each run considers a total of 500 slice requests. The total reward of RESET ($\delta = 10\%$) with zero penalty is slightly more than that of RESET ($\delta = 5\%$). This is intuitive since more redistributions can occur as δ increases.

TABLE I: Simulation settings

Network		Request											
Parameter	**Value**	**Type**	**Band. (Mbps)**	**Reward**	**Active time**	**UPF**		**Constants**			**VNF**	**CPU**	**Storage (GB)**
						F1	**F2**	α	β	γ	IDS	2	10
Topology	AttMpls	eMBB	30–100	6–10	20–100	VO	IDS, FR, NAT, TM	0.2	0.4	0.4	FR	2	5
Percentage ECs (η)	10% and 20%	uRLLC	5–15	8–10	5–20	IDS	VO, FR, NAT, TM	0.33	0.33	0.33	NAT	1	2
Link bandwidth	1–2 Gbps	mMTC	0.5–1.5	1	1–5	TM	VO, IDS, FR, NAT	0.4	0.4	0.2	TM	1	2
EC CPUs	72–100	Percentage of active slice redistribution (δ)				5% and 10%					VO	2	20
EC storage	2–3 TB	Redistribution penalty (σ)				0 and 0.5							
Predefined Constant	α: 0.33; β: 0.33; γ: 0.33												

NAT: network address translation

VO: video optimizer; TM: traffic monitoring; IDS: intrusion detection; FR: firewall

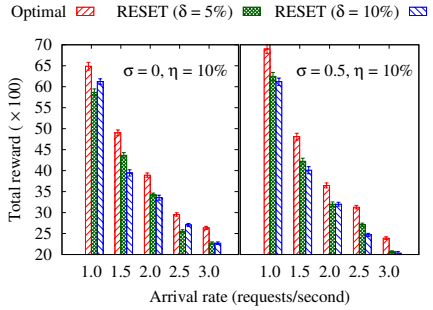

Fig. 2: Total reward

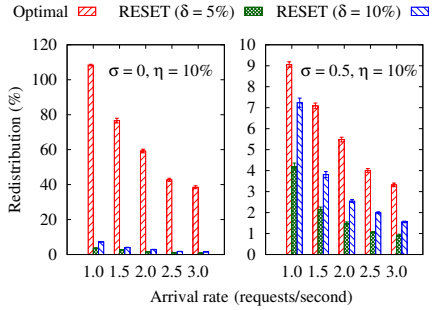

Fig. 3: Percentage of slice redistributions

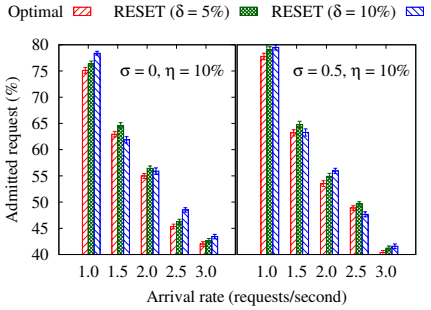

Fig. 4: Percentage of admitted slices

However, the reverse is true when $\sigma = 0.5$. Now, the total reward of RESET ($\delta = 5\%$) exceeds that of RESET ($\delta = 10\%$).

2) Percentage of Slice Redistributions: Figure 3 shows the percentage of slice redistributions for two values of the penalty factor σ. We see that the percentage of slices that are redistributed in the optimal solution is very high when $\sigma = 0$. For small arrival rates, more than 100% of the slices can be redistributed. This occurs because an active slice can be redistributed multiple times during the time it is served by the network. However, the percentage of redistributions decreases by an order of magnitude when σ is increased to 0.5. In contrast, the percentage of redistributions is markedly lower in RESET. This is because no more than a δ fraction of the active slices can be redistributed. However, as we saw in Figure 2, even with a low slice redistribution percentage, RESET achieves a reward that is competitive with the optimal solution. The percentages of slice redistributions increase as δ increases from 5% to 10% as more active slices are redistributed. Lastly, we observe that the percentage of redistribution decreases as the arrival rate increases. This occurs because it takes less time for 500 requests to arrive as the arrival rate increases.

3) Admitted Requests: Figure 4 shows the percentage of the newly admitted slice requests. Interestingly, we see that the percentage of admitted requests is almost equal for both schemes.

In summary, we see that RESET yields a reward competitive with the optimal solution, but with far fewer redistributions. Furthermore, we note that the computation time of RESET is smaller by two orders of magnitude than that of the optimal solution, as the optimization problem in (1) is NP-hard.

B. Extension to Large Scenarios

We now study the performance of RESET in large deployment scenarios. For this, we conduct the simulations with more ECs in the network. Since the existing edge slicing schemes [10], [12] consider only single-hop routing, we cannot compare RESET with them. Instead, we compare RESET with two other benchmark schemes, namely, Reward and FCFS. In Reward, the requests are prioritized based on their reward values without considering their bandwidth, computing, and storage demands. On the other hand, in FCFS, the requests are admitted in the order they are received. Thus, the request selection phase (cf. Section IV-B) of RESET is different from that of Reward and FCFS. However, the EC selection (cf. Section IV-C) and path selection (cf. Section IV-D) phases are the same for all three schemes. Furthermore, we set the redistribution percentage of active slices, δ, as 5% in all schemes.

1) Reward: Figure 5 compares the total reward of the three schemes for two values of σ and two values of η. With more ECs in the network, the reward of all three schemes increases as more number of requests are served. We see that RESET achieves a higher reward than Reward and FCFS. This is because RESET admits the requests based on the ratio of the reward and the resource cost in serving the request. On the other hand, Reward admits a request based on its reward value irrespective of the network resources it demands. This leads to a shortage of network resources over time and a lower reward. In contrast, FCFS admits the requests sequentially in the order they arrive until it runs out of either bandwidth, storage, or computing resources. Its reward is lower as it does not take the reward value into account while admitting slices.

2) Percentage of Slice Redistributions: Figure 6 compares the percentage of slice redistributions of the three schemes for different values of η. We see that the percentage of redistributions with FCFS is lower than that of the RESET and Reward. FCFS always admits the slice requests in the order they are

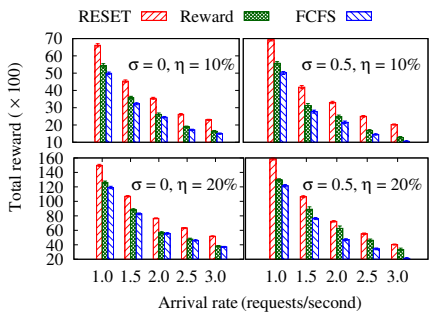

Fig. 5: Total reward

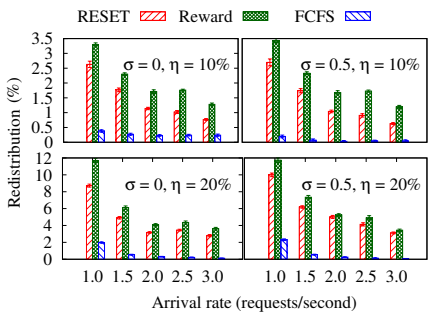

Fig. 6: Percentage of slice redistributions

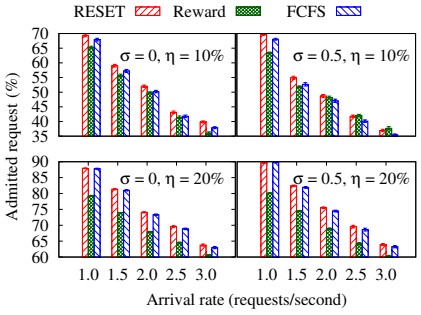

Fig. 7: Percentage of admitted slices

received. Therefore, the active slices that were redistributed are considered before the new requests. This, in turn, reduces the percentage of slice redistributions. On the other hand, RESET admits the redistributed active slices based on their reward value and resource cost. Therefore, a redistributed active slice may not be assigned to the EC to which it was assigned earlier. This, in turn, increases the percentage of slice redistributions with RESET. Similarly, Reward also entails a higher percentage of slice redistributions than FCFS as it admits slice requests based on their reward values.

3) Admitted Requests: Figure 7 compares the percentage of admitted requests of the three schemes for different values of η. We see that the percentage of admitted requests of RESET and FCFS is comparable while that of Reward is lower. This is because Reward does not consider the resource requirements of the slice requests while admitting them. This leads to a shortage of network resources over time.

VI. Conclusion

We studied the problem of edge network slicing in 5G with the goal of maximizing the total reward obtained from admitting new slice requests while also controlling the fraction of active slices that get redistributed. We proposed a greedy, polynomial-time, heuristic approach called RESET to solve the NP-hard problem. RESET consists of three phases – request selection, EC selection, and forwarding path selection. In the request selection phase, we designed a cost function to decide the order in which new slice requests are admitted. The EC selection phase determines the EC that fulfils the computing and storage demands of a slice request. The forwarding path selection phase determines the forwarding path to route the traffic associated with the admitted slice requests in the networks. We numerically observed that RESET achieved a performance close to the optimal solution but entailed far fewer slice redistributions. It also achieved a higher reward than the benchmark schemes. As a future extension of this work, we plan to study the cross-domain network slicing problem, in which the slicing occurs over the 5G RAN, edge, and core networks.

References

[1] "5G programmable infrastructure converging disaggregated network and compute resources," 5GPPP, Tech. Rep., Jan. 2018.

[2] "Verticals uRLLC use cases and requirements," NGMN Alliance, Tech. Rep., Feb. 2020.

[3] Rost et al., "Network slicing to enable scalability and flexibility in 5G mobile networks," *IEEE Commun. Mag.*, vol. 55, no. 5, pp. 72–79, May 2017.

[4] A. Ksentini and N. Nikaein, "Toward enforcing network slicing on RAN: Flexibility and resources abstraction," *IEEE Commun. Mag.*, vol. 55, no. 6, pp. 102–108, Jun. 2017.

[5] I. Afolabi, T. Taleb, K. Samdanis, A. Ksentini, and H. Flinck, "Network slicing and softwarization: A survey on principles, enabling technologies, and solutions," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 3, pp. 2429–2453, thirdquarter, 2018.

[6] "MEC in 5G networks," ETSI, Tech. Rep., 2018.

[7] L. Peterson, T. Anderson, S. Katti, N. McKeown, G. Parulkar, J. Rexford, M. Satyanarayanan, O. Sunay, and A. Vahdat, "Democratizing the network edge," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 49, no. 2, pp. 31–36, May 2019.

[8] Cisco, "Cisco converged 5G xhaul transport," 2018.

[9] ETSI, "Network functions virtualisation (NFV); management and orchestration," 2014.

[10] G. Castellano, F. Esposito, and F. Risso, "A distributed orchestration algorithm for edge computing resources with guarantees," in *Proc. IEEE INFOCOM*, Apr. 2019, pp. 2548–2556.

[11] Q. Liu and T. Han, "DIRECT: Distributed cross-domain resource orchestration in cellular edge computing," in *Proc. ACM Mobihoc*, Jul. 2019, pp. 181–190.

[12] S. D'Oro, L. Bonati, F. Restuccia, M. Polese, M. Zorzi, and T. Melodia, "Sl-edge: Network slicing at the edge," in *Proc. ACM Mobihoc*, Oct. 2020, pp. 1–10.

[13] F. Zhang, G. Liu, X. Fu, and R. Yahyapour, "A Survey on virtual machine migration: Challenges, techniques, and open issues," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 2, pp. 1206–1243, Secondquarter 2018.

[14] 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 *Proc. CNSM*, Nov. 2020, pp. 1–9.

[15] J. Pedreno-Manresa, P. S. Khodashenas, M. S. Siddiqui, and P. Pavon-Marino, "On the need of joint bandwidth and NFV resource orchestration: A realistic 5G access network use case," *IEEE Wireless Commun. Lett.*, vol. 22, no. 1, pp. 145–148, Jan. 2018.

[16] R. Ahuja, T. Magnati, and J. Orlin, *Network Flows: Theory, Algorithms, and Applications*. Prentice-Hall, 1993.

[17] S. Martello and P. Toth, *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley & Sons, 1990.

[18] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The internet topology zoo," *IEEE J. Sel. Areas Commun.*, vol. 29, no. 9, pp. 1765–1775, Oct. 2011.

[19] A. L. Barabási and R. Albert, "Emergence of scaling in random networks," *Science*, vol. 286, pp. 509–512, 1999.

[20] D. Harutyunyan, R. Fedrizzi, N. Shahriar, R. Boutaba, and R. Riggio, "Orchestrating end-to-end slices in 5G networks," in *Proc. CNSM*, Oct. 2019, pp. 1–9.

[21] A. Hackshaw, "Statistical formulae for calculating some 95% confidence intervals," in *A Concise Guide to Clinical Trials*. John Wiley & Sons, Ltd, 2009, pp. 205–207.

[22] "CPLEX CP optimizer," https://www.ibm.com/in-en/analytics/cplex-cp-optimizer, Mar. 2020.