Joint UPF and Edge Applications Placement and Routing in 5G & Beyond

Endri Goshi†, Hasanin Harkous*, Shohreh Ahvar‡, Rastin Pries*, Fidan Mehmeti†, Wolfgang Kellerer†
†Technical University of Munich, *Nokia Germany, ‡Nokia France
E-Mail:†{endri.goshi, fidan.mehmeti, wolfgang.kellerer}@tum.de,
*{hasanin.harkous, shohreh.ahvar, rastin.pries}@nokia.com,

***Abstract*—The development of 5G networks has enabled support for a vast number of applications with stringent traffic requirements, both in terms of communication and computation. Furthermore, the proximity of the entities, such as edge servers and User Plane Functions (UPFs) that provide these resources is of paramount importance. However, with the ever-increasing demand from these applications, operators often find their resources insufficient to accommodate all requests. Some of these demands can be forwarded to external entities, not owned by the operator. This introduces a cost, reducing the operator's profit. Hence, to maximize operator's profit, it is important to place the demands optimally in internal or external edge nodes. To this end, we formulate a constrained optimization problem that captures this objective and the inter-play between different parameters, which turns out to be NP-hard. Therefore, we resort to proposing a heuristic algorithm which ranks the demands according to their value to the operator and amount of resources they need. Results show that our approach outperforms the benchmark algorithms, deviating from the optimal solution by only $\sim 3\%$ on average.**

***Index Terms*—5G core, UPF, Edge clouds, Routing.**

I. Introduction

The introduction and deployment of 5G networks has enabled an array of new applications that leverage the high-throughput, scalability and flexibility of such networks. Computationally-intensive and delay-sensitive applications, such as Augmented and eXtended Reality (AR/XR), rely on the availability of compute and network resources at users' proximity. Through technologies such as Network Function Virtualization (NFV), network resources can be leveraged by flexibly deploying User Plane Functions (UPFs) [1]. Moreover, Multi-Access Edge Computing (MEC) allows users to offload compute tasks to servers at the network edge.

Currently, MEC solutions consider a separate orchestration of the compute and network resources, where cloud operators and Telco Network Operators (TNOs) manage only the resources of their domain. The importance of the joint consideration of these types of resources is highlighted by concepts such as in-network computing and Compute and Network Convergence (CNC) [2]. Collaboration initiatives from AWS [3] and Verizon [4], aim to facilitate the deployment of Edge Applications (EAs) close to the TNO's UPFs. These approaches have two main shortcomings: i) EA orchestration burden falls on the EA Providers (EAPs), who must know their users' location to orchestrate compute resources efficiently; and ii) TNOs lack prior information about the deployment locations of specific EAs, resulting in inefficient orchestration of network resources.

Therefore, in this work, we propose a different approach where the TNOs establish agreements with EAPs to take over the orchestration of the EAs, in addition to the UPFs. This way, they can efficiently route user demands within the transport network, while establishing a new line of revenue through the deals made with the EAP. This approach is applicable also to private 5G network operators looking to expand the services they offer and efficiently utilize their edge infrastructure.

Research works investigating the joint allocation of compute and network resources, consider the problem from a network planning perspective. As such, they focus on optimally placing dimensioning the edge infrastructure and statically deploying the UPFs, predominately assuming a 1-to-1 association between Base Stations (BSs), Edge Nodes (ENs), and UPFs [5]–[8]. In contrast, in this work, we consider an already-deployed edge infrastructure and transport network. A holistic approach to the problem is taken, integrating aspects such as UPF and EA deployment, user demand placement and routing, in a single problem formulation. Modeling such a system with realistic assumptions presents challenging tasks, especially given the multiple dimensions of the problem.

To this end, we model a system with integrated compute and network infrastructure, and then formulate the joint UPF and EA placement and routing problem as a constrained optimization problem using Integer Linear Programming (ILP). The problem is NP-hard, and therefore, we design and implement a greedy heuristic called *RanGr*. Our evaluations based on extensive realistic simulations show that *RanGr* outperforms the benchmark algorithms and is able to find high-quality solutions within a very short time, making it a viable algorithm to be used by TNOs for the joint orchestration of their compute and network resources.

The remainder of the paper is organized as follows. A detailed explanation of the system model is provided in Section II, followed by the problem formulation in Section III. Section IV introduces *RanGr*, our proposed heuristic approach, while Section V details the evaluation scenarios and the obtained results. Section VI presents related work in the areas of edge infrastructure and UPF deployment. Lastly, Section VII discusses future work and concludes the paper.

This work was supported in part by the German Federal Ministry of Education and Research (BMBF) as part of the project "6G-ANNA" with project identification number 16KISK107.

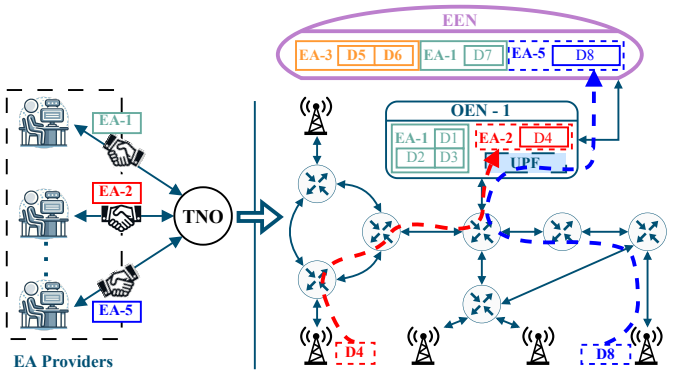


Fig. 1. System overview depicting the initialization of UPFs and EAs, and the placement of demands in the OENs and the EEN.

II. System Model

An overview of the system considered in this work is given in Fig. 1. The edge infrastructure consists of two types of ENs: i) Operator Edge Node (OEN), and ii) External Edge Node (EEN). The former, represents edge servers owned by the TNO, and used to deploy UPFs, EAs, and place incoming demands that are served by the EAs. OENs are denoted by $e \in \mathcal{E}$ and there are $|\mathcal{E}|$ OENs comprising the edge infrastructure. Each OEN e is characterized by its processing and storage capacity, denoted by P_e and S_e, respectively, and by its operational cost C_e. On the other hand, the EEN, denoted by e^*, is an abstraction of edge servers not owned by the operator (e.g., provided by AWS or Google). The resources in the EEN, which are assumed to be abundant, are rented by operators for additional deployment of EAs and demands.

In addition, the edge network comprises $|\mathcal{J}|$ BSs denoted by $j \in \mathcal{J}$, $|\mathcal{K}|$ forwarding nodes denoted by $k \in \mathcal{K}$, and $|\mathcal{V}|$ links interconnecting the transport network, denoted by $v \in \mathcal{V}$. Each link v is characterized by its bandwidth capacity B_v, and its latency t^v[1]. We assume that the forwarding nodes can process the incoming traffic at line rate, as long as the link capacity is not exceeded. Thus, we do not model their capacity and processing latency. The edge network topology is modeled as a directed graph $G = (\mathcal{J} \cup \mathcal{K} \cup \mathcal{E} \cup e^*, \mathcal{V})$.

A. EAs and User Demands

In our system, agreements exist between EAPs and TNOs to manage the orchestration of $|\mathcal{I}|$ different EAs denoted by $i \in \mathcal{I}$. They represents services (e.g., AR/XR applications) that are deployed on ENs and serve the incoming user demands. EAs are characterized by their idle CPU utilization p_i (e.g., due to monitoring) and storage requirement s_i.

Demands are abstracted as Packet Data Unit (PDU) sessions. In this work, we consider the BS as the source of a demand since that is the entry point of the traffic into the transport network, and thus, demands can belong to the same user or to different ones, making the approach general enough. From each BS j there are $|\mathcal{L}_{j,i}|$ demands of EA type i that need to be served, denoted by $l \in \mathcal{L}_{j,i}$. Demands are uniquely identified by the tuple $\{j, i, l\}$, and are characterized by their CPU requirement $p_{j,i,l}$, bandwidth requirement $b_{j,i,l}$, and maximum delay budget $t_{j,i,l}$. Serving a demand (i.e., placing it on an OEN or EEN) generates revenue for the operator and is captured by the utility value, denoted as $U_{j,i,l}$. Conversely, the cost incurred for renting resources in the EEN to place a demand is captured by $C_{j,i,l}$.

B. UPFs and Routing

One of the new key aspects of 5G Core architecture is the softwarization of its entities, enabled by advancements in NFV and programmable data planes technologies [9]. UPFs in 5G are commonly deployed as Virtual Network Functions (VNFs) on general-purpose servers. On this basis, UPFs in our system are modeled as VNFs. Clusters of UPFs are dynamically initialized on OENs and can be *horizontally scaled* (i.e., new replicas are deployed to keep up with the input traffic), where $r \in \mathcal{R}$ denotes the number of UPFs in a cluster, and $R = |\mathcal{R}|$ is the maximum cluster size. For a UPF of scale r deployed on the OEN, p_r CPU resources have to be reserved, amounting to a total processing bandwidth of B_r.

One of prime tasks accomplished by the UPF is to anchor user traffic that leaves the core network, and forward it to its destination. In our system, the user traffic is represented by the demands, whose destinations are the EAs deployed on OENs or EEN. Thus, "accepting" a demand means finding an EN with enough resources to place the demand, and routing it through a viable path from the source BS to the anchoring UPF, and further to the destination EA. We assume that if a demand is to be placed on an OEN, it will be anchored on a UPF instance deployed on that same OEN (e.g., demand D4 shown in red in Fig. 1). Otherwise, if the demand is to be offloaded to the EEN (e.g., demand D4 shown in blue in Fig. 1), it will be anchored on a UPF deployed on a viable OEN and will then be routed to the EEN. To this end, for routing we consider a *link-path* formulation [10], aimed to find the best path from source BS to destination OEN/EEN. Therefore, a set of $|\mathcal{N}|$ paths from any BS to any OEN and the EEN is calculated, each denoted by $n \in \mathcal{N}$. This formulation allows for a more granular representation of routes and captures constraints and objectives related to path characteristics (e.g., latency, bandwidth) more effectively.

III. Problem Formulation

In this work, we focus on maximizing the utility derived from demands that are accepted and served on the OENs. We also consider the costs associated with keeping the OENs operational and the rental costs incurred from offloading demands to the EEN. The objective is to maximize the following function:

$$\max \left(\sum_{j,i,l} (U_{j,i,l} \cdot (1 - F_{j,i,l}) - C_{j,i,l} \cdot T_{j,i,l}) - \sum_{e} z_e \cdot C_e \right), \quad (1)$$

[1]Link latency is defined as the sum of the transmission and propagation latencies. We assume that the link latency between any OEN and the EEN is higher than between any two nodes in operator's transport network.

where $U_{j,i,l}$ represents the utility (profit) gain from accepting a demand, $F_{j,i,l} \in \{0,1\}$ denotes whether demand $\{j,i,l\}$ is rejected, $C_{j,i,l}$ denotes costs incurred if demand is placed on EEN, and $T_{j,i,l} \in \{0,1\}$ denotes whether the demand is offloaded to the EEN. $z_e \in \{0,1\}$ denotes whether OEN e is operational and C_e denotes the operational costs. The first term refers to the utility gained from accepting a demand, the second term refers to the cost paid to the EEN operator, and the last term captures the operating cost of OENs.

In addition, we formulate 17 constraints related to the finite resources of OENs, UPFs, network resources, EA deployment and demand anchoring and placement.

1) OEN capacity constraints: The CPU resources allocated for the deployment of UPFs, initialization of EAs, and placement of demands, should not exceed the OEN's total CPU capacity:

$$\sum_{r=1}^{|\mathcal{R}|} p_r w_{r,e} + \sum_{i=1}^{|\mathcal{I}|} p_i x_{i,e} + \sum_{j=1}^{|\mathcal{J}|} \sum_{i=1}^{|\mathcal{I}|} \sum_{l=1}^{|\mathcal{L}_{j,i}|} p_{j,i,l} d_{j,i,l,e} \leq P_e, \forall e \in \mathcal{E}, \tag{2}$$

where $w_{r,e} \in \{0,1\}$ denotes whether a UPF cluster with r replicas is initialized on OEN e, $x_{i,e} \in \{0,1\}$ denotes whether EA i is deployed on OEN e, and $d_{j,i,l,e} \in \{0,1\}$ denotes whether demand $\{j,i,l\}$ is placed on OEN e.

The storage resources taken up by the deployment of EAs should not exceed the OEN's total storage capacity:

$$\sum_{i=1}^{|\mathcal{I}|} s_i \cdot x_{i,e} \leq S_e, \quad \forall e \in \mathcal{E}. \tag{3}$$

2) EA deployment constraint: EAs can be deployed on an OEN only if the OEN is in operational mode (*ON*):

$$x_{i,e} \leq z_e, \quad \forall i \in \mathcal{I}, \forall e \in \mathcal{E}. \tag{4}$$

3) UPF deployment constraints: UPF clusters can be initialized on an OEN only if it is in operational mode (*ON*):

$$\sum_{r=1}^{|\mathcal{R}|} w_{r,e} \leq z_e, \quad \forall e \in \mathcal{E}. \tag{5}$$

On an OEN, there can only be one UPF cluster deployed:

$$\sum_{r=1}^{|\mathcal{R}|} w_{r,e} \leq 1, \quad \forall e \in \mathcal{E}. \tag{6}$$

4) Demand placement constraints: A demand can be placed on an OEN only if an EA of the same type is initialized on the OEN:

$$d_{j,i,l,e} \leq x_{i,e}, \quad \forall j \in \mathcal{J}, \forall i \in \mathcal{I}, \forall l \in \mathcal{L}_{j,i}, \forall e \in \mathcal{E}. \tag{7}$$

An EA should be initialized on an OEN only if there are demands of the same type that will be placed on the OEN:

$$\sum_{j=1}^{|\mathcal{J}|} \sum_{l=1}^{|\mathcal{L}_{j,i}|} d_{j,i,l,e} \geq x_{i,e}, \quad \forall i \in \mathcal{I}, \forall e \in \mathcal{E}. \tag{8}$$

There are three options for handling a demand: it can be rejected, offloaded to the EEN, or placed on exactly one OEN:

$$F_{j,i,l} + T_{j,i,l} + \sum_{e=1}^{|\mathcal{E}|} d_{j,i,l,e} = 1, \ \forall j \in \mathcal{J}, \forall i \in \mathcal{I}, \forall l \in \mathcal{L}_{j,i}. \tag{9}$$

5) Demand anchoring constraints: An accepted demand must be anchored on exactly one OEN:

$$F_{j,i,l} + \sum_{e=1}^{|\mathcal{E}|} w_{j,i,l,e} = 1, \quad \forall j \in \mathcal{J}, \forall i \in \mathcal{I}, \forall l \in \mathcal{L}_{j,i}, \tag{10}$$

where $w_{j,i,l,e} \in \{0,1\}$ denotes whether demand $\{j,i,l\}$ is anchored on the UPF cluster deployed on OEN e.

Demands will be anchored on OEN e if they are also placed on OEN e, or if they are placed on the EEN but OEN e is the last hop in the path:

$$w_{j,i,l,e} = d_{j,i,l,e} + \sum_{n=1}^{|\mathcal{N}|} \delta_{j,i,l}^{n,v}, \tag{11}$$
$$\forall j \in \mathcal{J}, \forall i \in \mathcal{I}, \forall l \in \mathcal{L}_{j,i}, \forall e \in \mathcal{E}, v = \{e, e^*\},$$

where $\delta_{j,i,l}^{n,v} \in \{0,1\}$ denotes whether link v is part of the path n that routes the demand $\{j,i,l\}$. As $v = \{e, e^*\}$, the constraint considers only the last hop from any OEN to EEN.

A demand can be anchored on an OEN, if there is a UPF cluster deployed on that OEN:

$$w_{j,i,l,e} \leq \sum_{r=1}^{|\mathcal{R}|} w_{r,e}, \quad \forall j \in \mathcal{J}, \forall i \in \mathcal{I}, \forall l \in \mathcal{L}_{j,i}, \forall e \in \mathcal{E}. \tag{12}$$

UPF clusters are initialized on an OEN only if there are demands anchored on that OEN:

$$\sum_{j=1}^{|\mathcal{J}|} \sum_{i=1}^{|\mathcal{I}|} \sum_{l=1}^{|\mathcal{L}_{j,i}|} w_{j,i,l,e} \geq \sum_{r=1}^{|\mathcal{R}|} w_{r,e}, \quad \forall e \in \mathcal{E}. \tag{13}$$

6) Routing constraints: Accepted demands can be routed through exactly one path:

$$\sum_{n=1}^{|\mathcal{N}|} \theta_{j,i,l}^{n} = d_{j,i,l,e} + T_{j,i,l}, \tag{14}$$
$$\forall j \in \mathcal{J}, \forall i \in \mathcal{I}, \forall l \in \mathcal{L}_{j,i}, \forall e \in \mathcal{E},$$

where $\theta_{j,i,l}^{n} \in \{0,1\}$ denotes whether path n is used to route demand $\{j,i,l\}$.

If a demand is routed through a path, then all the links in that path are used to transport the demand:

$$\delta_{j,i,l}^{n,v} = \theta_{j,i,l}^{n}, \ \forall j \in \mathcal{J}, \forall i \in \mathcal{I}, \forall l \in \mathcal{L}_{j,i}, \forall n \in \mathcal{N}, \forall v \in \mathcal{V}. \tag{15}$$

The latency incurred in the selected path, expressed as a function of the number of links in the path and the links' latency, cannot exceed the demand's maximum delay budget:

$$\sum_{v=1}^{|\mathcal{V}|} \delta_{j,i,l}^{n,v} \cdot t^v \leq t_{j,i,l}, \forall j \in \mathcal{J}, \forall i \in \mathcal{I}, \forall l \in \mathcal{L}_{j,i}, \forall n \in \mathcal{N}. \tag{16}$$

7) Network resources: The total volume of traffic that is anchored on an OEN cannot exceed the total capacity of OEN's UPF cluster:

$$\sum_{j=1}^{|\mathcal{J}|} \sum_{i=1}^{|\mathcal{I}|} \sum_{l=1}^{|\mathcal{L}_{j,i}|} w_{j,i,l,e} \cdot b_{j,i,l} \leq \sum_{r=1}^{|\mathcal{R}|} w_{r,e} \cdot B_r, \quad \forall e \in \mathcal{E}. \tag{17}$$

The total volume of traffic transported through a link cannot exceed the link's bandwidth:

$$\sum_{j=1}^{|\mathcal{J}|}\sum_{i=1}^{|\mathcal{I}|}\sum_{l=1}^{|\mathcal{L}_{j,i}|}\sum_{n=1}^{|\mathcal{N}|}\delta_{j,i,l}^{n,v}\cdot b_{j,i,l}\leq B_v,\quad \forall v\in\mathcal{V}. \tag{18}$$

To summarize, the constrained optimization problem related to our system, and in the focus of this paper, is:

$$\max\left(\sum_{j,i,l}(U_{j,i,l}\cdot(1-F_{j,i,l})-C_{j,i,l}\cdot T_{j,i,l})-\sum_{e}z_e\cdot C_e\right)$$
$$\text{s.t.}\qquad (2)-(18).$$

The optimization problem (1)-(18) is an ILP since all the decision variables are binary, and the relation between them in all the constraints are linear. Moreover, the problem of demand placement in an EN with limiting CPU resources is NP-hard. Due to space limitations, we refer the readers to the technical report [11] for the proof of the NP-hardness of our problem by reducing the Multiple Knapsacks Problem to it.

IV. RanGr

Since our problem is NP-hard, finding the optimal solution with a solver is computationally very expensive. Therefore, to overcome this inherent limitation of any variation of the Knapsack Problem (KP), we design and implement our own ranked greedy algorithm called *RanGr*. As will be shown in Section V, *RanGr* finds near-optimal solutions and very fast.

Algorithm 1 provides a detailed overview of *RanGr*'s execution. For enhanced clarity, we have separated the definitions of the `Main` and `anchorAndPlaceDemands` procedures. Initially, the algorithm calculates paths from each BS to every OEN and EEN, for the given network topology. The function called in line 2 does not calculate only the shortest path, but all the paths between two nodes that are shorter than a predefined *cutoff* value. This threshold on the maximum number of hops in the path is used as a stopping condition for the path calculation, and is based on the fact that routing demands via longer paths may violate their latency constraint. In analogy with the ILP's link-path formulation, this is the same function that calculates the list of paths which are then given as input to the ILP $(1)-(18)$, with the only difference being that in *RanGr* the paths are also sorted based on the number of hops.

RanGr is designed as a *primal* greedy heuristic, meaning that the demands are ranked and sorted in descending order of their rank (lines 3-5) before executing the anchoring and placement procedure in line 6. More details on the ranking formula and the anchoring and placement algorithm are given in the following subsections. In lines 7-13 a *"reallocation"* process takes place. First, a list of the rejected demands is created, and then *RanGr* evaluates if the OENs have become profitable. Therefore, if the profit from demands anchored or placed on the OEN is not greater than its operational costs, the algorithm proceeds to undo the anchoring/placement of the demands from that OEN, and appends them to the list of rejected demands. Lastly, the list is sorted based on the rank of the demands, and the anchoring and placement algorithm is executed again for this subset of the initial demands. This step marks the end of *RanGr*'s execution, and the final UPF and EA deployment schemes, demands' anchoring and placement schemes and the selected routes are obtained.

Algorithm 1: RanGr

Input: *D* — List of input demands; *BSList* — List of BSs in the topology; *OENList* — List of OENs and their parameters; *EEN* — EEN and its parameters
Output: Demands' anchoring and placement scheme

```
1  def Main():
2      getAndSortPaths(BSList, OENList, EEN)
3      for d in D:
4          d_rank ← rankDemand(d_U, d_C, d_CPU)
5      D_sorted ← sort(D, key ← d_rank)
6      anchorAndPlaceDemands(D_sorted, OENList, EEN)
7      D^R ← getRejectedDemands()
8      for OEN in OENList:
9          if OEN_Profit ≤ OEN_ON_cost:
10             D^R_OEN ← undoAnchorAndPlacement()
11             D^R.append(D^R_OEN)
12     D^R_sorted ← sort(D^R, key ← d_rank)
13     anchorAndPlaceDemands(D^R_sorted)
14 def anchorAndPlaceDemands(D_sorted, OENList, EEN):
15     initEmptyList (anchoredDemands)
16     for d in D_sorted:
17         d_anchored ← False
18         for OEN in OENList:
19             OEN_RP, OEN_RA ← rankOEN(OEN)
20         rankedOENs ← sort(OENList, key ← OEN_RP)
21         topOEN ← rankedOENs[0]
22         if canPlaceInEdge(topOEN):
23             anchorInOEN(d, topOEN)
24             appendAnchoredd
25             d_edge ← topOEN
26             d_anchored ← True
27         else:
28             rankedOENs ← sort(OENList, key ← OEN_RA)
29             for OEN in rankedOENs:
30                 if canAnchorInOEN(OEN):
31                     anchorInOEN(d, OEN)
32                     d_anchored ← True
33                     break
34         if d_anchored is False:
35             rejectDemand(d)
36     for d in anchoredDemands:
37         if canPlaceInEdge(d_edge):
38             placeInEdge(d, d_edge)
39         elif canPlaceInEdge(EEN):
40             placeInEdge(d, EEN)
41         else:
42             rejectDemand(d)
```

A. Demand Ranking

In order to obtain high-quality solutions that maximize the operator's profit, it is important to define a ranking criteria that efficiently captures the "attractiveness" of anchoring/placing a demand. In the classic KP, the *efficiency* of an item is defined as the *profit-to-weight ratio*, where weight is the only constraint that determines whether an item can be placed into the knapsack. In our problem, although there are many constraints on the resources at hand, most of them serve the purpose of identifying the subset of OENs where the demand can be anchored/placed on (e.g., the latency and bandwidth constraints). Therefore, CPU and storage represent the only resources counterpart to the weight in the classic KP. As the

latter is related to the EA that will serve the demands and not directly to the demand itself, for the sake of simplicity, it is omitted from the rank calculation. Summarizing, we have the following:

Definition 1. *The rank/efficiency of a demand is defined as*

$$d_{rank} = \left(\frac{d_U}{d_{CPU}}, -(d_U - d_C) \right), \tag{19}$$

where d_U, d_C and d_{CPU} represent the utility, offloading cost, and CPU requirement of the demand, respectively.

To prevent any unintentional bias caused by varying ranges among the values, all data are normalized. The rank of the demand is defined as a tuple, where the ratio of utility to CPU requirement takes precedence. In cases when this ratio is identical for more than one demand, the difference between its utility and cost (i.e., the profit margin) is the deciding factor. Following this ranking methodology, we compile a list where demands that are more *profitable* when placed on OENs, and that have a lower profit margin when offloaded to the EEN, are ranked higher.

At a first glance, it may seem that the chosen ranking formula is too simple for such a complex problem. Nonetheless, we experimented with other formulas that take into account the relationship between utility, cost, and CPU resources in a single equation rather than a tuple, but the quality of the solution did not improve, and in some cases it even degraded.

B. Anchoring and Placement Procedure

After the demands are ranked and sorted, in line 6, the anchoring and placement procedure (lines 14-42) is executed. This procedure implements a best-fit algorithm, which identifies the most suitable EN for each demand being evaluated. The objective of maximizing the operator's profit is translated in this procedure to: i) maximize the number of accepted (i.e., anchored) demands, ii) maximize the number of highly-profitable demands placed on OENs, and iii) minimize the number of OENs in operation mode. Anchoring demands on an OEN means that UPF instances need to be initialized on the OENs, and CPU resources have to be allocated to these UPFs. At the same time, OENs' CPU resources constrain the placement of demands onto these OENs. For this reason, the anchoring and placement procedures are separated from each other and executed sequentially.

RanGr tries first to anchor as many demands as possible (lines 16-35). For each demand being evaluated, it ranks the OENs by calculating:

- OEN_{R_A} – the remaining CPU resources at the OEN after anchoring the demand, defined as:

$$OEN_{R_A} = OEN_{CPU'} - UPF_{upscaleCPU}, \tag{20}$$

  where $OEN_{CPU'}$ is the amount of available CPU resources, and $UPF_{upscaleCPU}$ represents the CPU resources to be allocated to the UPF in case that anchoring the demand would lead to a UPF upscale.

- OEN_{R_P} – the remaining CPU resources in the OEN after anchoring the demand and assuming it will also be placed there, defined as:

$$OEN_{R_P} = OEN_{CPU''} - UPF_{upscaleCPU} - EA_{CPU} - d_{CPU}, \tag{21}$$

  where $OEN_{CPU''}$ is the amount of available CPU resources assuming that previous demands anchored on this OEN are also placed there, EA_{CPU} is the CPU requirement of the EA that would serve the demand (considered only in the first occurrence of the EA in the given OEN), and d_{CPU} is the demand's CPU requirement.

Since the demands can be placed either on the same OEN where they are anchored, or on the EEN, we can think of the anchoring procedure as a non-binding placement procedure. After OENs are ranked, they are sorted by the value of $\text{OEN}_{\text{R}_\text{P}}$, in decreasing order. The viability of the top-ranked OEN to place the demand is then evaluated. If placing the demand would not violate any of the constraints, *RanGr* proceeds to anchor the demand on the OEN and save the placement scheme for later. If placing would not be possible, then the OENs are again sorted in decreasing order, but this time by the value of $\text{OEN}_{\text{R}_\text{A}}$. The algorithm iterates over the list of sorted OENs for anchoring, and once it finds a suitable OEN, it anchors the demand and breaks the loop. In cases where a suitable OEN could not be found, the demand is rejected.

Lastly, *RanGr* executes the placement procedure (lines 36-42) by iterating through the demands that were successfully anchored. Here, *RanGr* evaluates again if it would be possible to place the demand on the same OEN where it is anchored on. The reason why this check is performed again is due to scenarios where as *RanGr* evaluates more demands, CPU resources end up being used to initialize more UPFs, and the assumed placement schemes for some of the demands may not be valid anymore. If the placement scheme is still valid, *RanGr* proceeds to place the demands; otherwise, it checks if the demand can be offloaded to the EEN (while still being anchored on the original OEN). If that is not the case (e.g., the latency requirement cannot be satisfied), the demand is rejected.

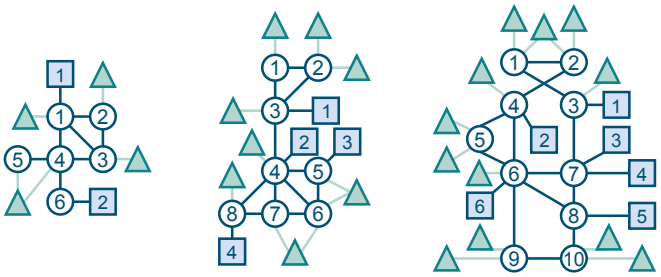
Regarding its computational complexity, *RanGr* exhibits a worst-time complexity of $O(|D|\log|D| + |D| \times |E|\log|E|)$. For the complete analysis on how this value is obtained, we refer the readers to the technical report [11].

V. Performance Evaluation

We implement *RanGr* in Python and compare its performance with the ILP solved with Gurobi [12], and two benchmark algorithms. The evaluation setup consists of a Virtual Machine (VM), to which we allocate 20 CPU cores and 64 GB of RAM. In the following, we discuss in detail the evaluation scenarios and the obtained results.

A. Benchmarks

With reference to algorithm evaluations performed in the literature for similar problems [5], [6], [13]–[15], we have implemented two benchmark algorithms to compare



(a) Small (b) Medium (c) Large

Fig. 2. Edge network topologies considered for the evaluation. The configuration of the number of BSs (triangles), switches (circles), and OENs (squares) for each topology is: a) $(4, 6, 2)$, b) $(8, 8, 4)$, and c) $(12, 10, 6)$. The same amount of resources is allocated to each OEN and link in each topology.

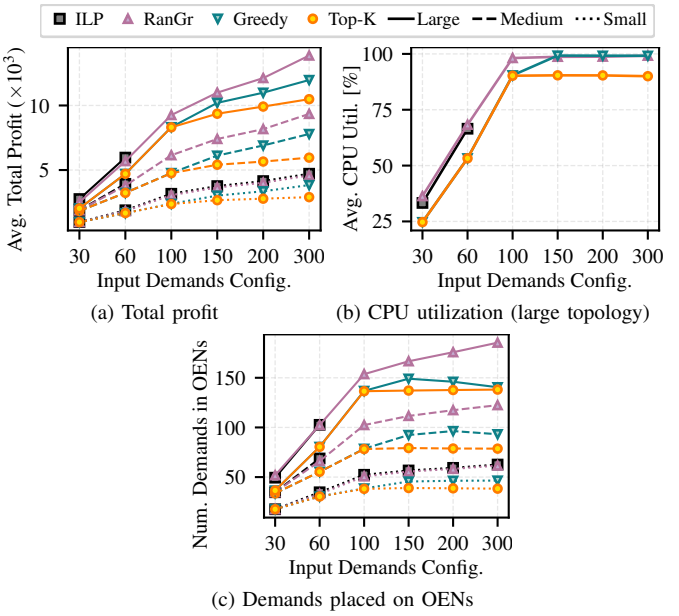
TABLE I
PARAMETERS USED FOR GENERATING INPUT DEMANDS

Demand parameter	Value
EA type	$[1, 2, 3, 4, 5]$
CPU req.	$[750, 1000, 1250, 1500]$ mCPU
Bandwidth req.	$[30, 40, 50, 60]$ Mbps
Delay budget	$[3, 4, 5]$ ms
Cost of EEN offloading	$[40, 50, 60, 70]$ units
Utility	$[44 - 91]$ units

to *RanGr*. These are: i) Greedy, and ii) Top-K. Instead of *RanGr*'s ranking mechanism, these algorithms rank the demands based on their utility values. Moreover, *RanGr* dynamically initializes OENs and UPF clusters. Greedy and Top-K determine these aspects in the beginning, based on the CPU and bandwidth requirements of the input demands. Lastly, *RanGr* employs a best-fit strategy when selecting on which OEN to place the demand, while the benchmark algorithms utilize a first-fit approach, where demands are placed on the first viable OEN.

B. Evaluation Scenarios

We compare the performance of *RanGr* for three different topologies of different sizes: i) small, ii) medium, and iii) large, as illustrated in Fig. 2. Each OEN is assumed to have 32 CPU cores (32000 mCPU) and 250 GB of storage, while the links have a capacity of 10 Gbps. Moreover, we assume a switching on cost of 200 units for OENs, 1 ms latency for the transport links, and 1.5 ms latency for the links connecting OENs to EEN. The sets of input demands are generated as a percentage of the total OENs' CPU resources that they require. Thus, sets with demands that in total have CPU requirements amounting to $[30, 60, 100, 150, 200, 300]\%$ of the capacity of OENs are generated. For each scenario, we generate 100 different inputs, where the parameters of a demand are picked randomly from a pre-defined set, and each demand is assigned randomly to a BS. The demand parameters are summarized in Table I. Due to the high execution time, ILP results are available only for the small topology, and for the medium and large topologies, results are obtained only for the lightly-loaded scenarios.



(a) Total profit (b) CPU utilization (large topology)

(c) Demands placed on OENs

Fig. 3. Performance of the algorithms for different configurations of input demands, in terms of a) total profit, b) OEN CPU utilization, and c) number of demands placed on OENs. Results for the small, medium and large topologies are shown in dotted, dashed, and solid lines, respectively.

C. Results

In this subsection, we discuss the results of our evaluation with respect to the total profit, CPU utilization of the OENs, and the number of demands placed in OENs . Due to space limitations, for the comparison of the execution time of each algorithm we refer the readers to the technical report [11].

1) Profit: Fig. 3a shows the average total operator's profit for different topologies and demands. *RanGr* exhibits a near-optimal performance consistently, deviating only $\sim 3\%$ from the optimal solution found by solving the ILP with Gurobi. Greedy and Top-K, on the other hand, perform good only for the lightly-loaded scenarios, and as the number of demands increases, their performance degrades, deviating on average $\sim$ 19% and $\sim$ 33% from the optimal solution. In highly-loaded scenarios, *RanGr* consistently outperforms Greedy, with the latter deviating by on average $\sim$ 18% and $\sim$ 10% from *RanGr*'s solution, for the medium and large topologies, respectively. The total profit obtained by using Top-K is on average $\sim$ 29% lower than *RanGr* for the medium topology, and $\sim$ 17% for the large topology. The superior performance of *RanGr* can be attributed to its ranking mechanism, and the best-fit strategy taken for anchoring and placing the demands in the edge infrastructure.

2) CPU Utilization and Demands in OENs: Fig. 3b shows the average CPU utilization (considering all OENs) for the large topology. *RanGr* again performs very good, reaching $\sim 98\%$ CPU utilization as the demands increase. However, it is important to also evaluate how these resources are distributed. Fig. 3c, shows the number of demands placed on the OENs for all the topologies. In the case of *RanGr*, this number increases consistently as the input size increases, proving that the ranking operation can efficiently capture the value

of the demands. In terms of CPU utilization, Top-K could only reach up to $\sim 90\%$, for all the overloaded scenarios. By investigating the average number of demands it has placed on OENs, we see that it is almost static for all the highly-loaded scenarios. The reason is that Top-K considers only K demands for placement, leading to inefficient CPU utilization. Lastly, Greedy also reaches $\sim 98\%$ CPU utilization, but in Fig. 3c we see that it performs bad in allocating these resources to the demands. Specifically, this is a result of the inefficient ranking mechanism, which allocates *"heavy"* demands first, saturating the available resources as a result.

VI. Related Work

The edge server placement problem has been investigated for different objectives and with different considerations [6], [8], [16]–[18]. From a profit perspective, the authors in [16] minimize the number of MEC locations and model the relation between BSs and MECs, while the authors in [6] additionally consider UPF deployment when dimensioning the edge infrastructure and formulate an objective that maximizes the operator's profit. The authors in [8] optimize the placement of edge servers in a BS topology, while minimizing the access delay. Energy-aware [17] and load-aware [18] deployment procedures are also investigated. A joint approach to the edge server and service placement problem is given in [7]. However, these works assume a network planning approach, trying to optimally dimension the edge infrastructure for any future services that might be deployed. Here, we formulate the problem for an already-deployed edge infrastructure, and consider aspects related to service and UPF placement, as well as demand routing.

Leyva-Pupo *et al.* formulate the problems of minimizing capital and operating expenditures when deploying UPFs [19] and UPF chains [13] in the 5G infrastructure. In another research work, the authors take a more holistic approach for jointly dimensioning the edge infrastructure and deploying UPFs. A multi-objective framework is presented in [20] which solves the multi-objective optimization problem of minimizing the number of edge nodes, UPFs, and relocations.

Service placement and scheduling for edge applications is investigated in [14], while the problem of jointly placing services in edge servers located in BSs, and routing user requests in the access network is investigated in [21]. Lastly, the authors in [15] propose a model to jointly slice the network and compute resources available at the edge, while minimizing the overall user-experienced latency.

Therefore, to the best of our knowledge, this is the first work that models and formulates the problem of jointly orchestrating the deployment of UPFs, Edge Applications (EAs), and demands on the edge infrastructure, with the objective of maximizing TNO's profit.

VII. Conclusion

In this paper, we considered the problem of jointly placing UPFs and edge applications as well as routing the demands, encompassing the entire cellular network. The goal was to maximize the operator's profit. The different traffic requirements of the tasks are taken into account, including the tolerable maximum latency of each demand, as well as the limitations across different types of resources in the network. The resulting optimization problem is NP-hard, and we resort to using a heuristic, *RanGr*. Results showed that the performance of *RanGr* is near-optimal and considerably better than the benchmark. As future work, multi-objective formulations that consider the energy consumption of ENs and scenarios with dynamic input demands will be considered.

References

[1] B. B. et al., "Technology pillars in the architecture of future 5G mobile networks: NFV, MEC and SDN," *Computer Standards & Interfaces*, vol. 54, 2017.

[2] Q. Duan, S. Wang, and N. Ansari, "Convergence of networking and cloud/edge computing: Status, challenges, and opportunities," *IEEE Network*, vol. 34, no. 6, 2020.

[3] AWS, "Wavelength," 2024. [Online]. Available: https://aws.amazon.com/wavelength

[4] Verizon, "Edge Discovery Service," 2024. [Online]. Available: https://www.verizon.com/business/5g-edge-portal

[5] Y. Li, X. Ma, M. Xu, A. Zhou, Q. Sun, N. Zhang, and S. Wang, "Joint placement of UPF and edge server for 6G network," *IEEE Internet of Things Journal*, vol. 8, no. 22, 2021.

[6] Y. Li, A. Zhou, X. Ma, and S. Wang, "Profit-aware edge server placement," *IEEE Internet of Things Journal*, vol. 9, no. 1, 2022.

[7] X. Zhang, Z. Li, C. Lai, and J. Zhang, "Joint edge server placement and service placement in mobile-edge computing," *IEEE Internet of Things Journal*, vol. 9, no. 13, 2022.

[8] S. Wang, Y. Zhao, J. Xu, J. Yuan, and C.-H. Hsu, "Edge server placement in mobile edge computing," *Journal of Parallel and Distributed Computing*, vol. 127, 2019.

[9] R. MacDavid, C. Cascone, P. Lin, B. Padmanabhan, A. ThakuR, L. Peterson, J. Rexford, and O. Sunay, "A P4-based 5G user plane function," in *Proc. of ACM SOSR*, 2021.

[10] M. Pióro and D. Medhi, *Routing, Flow, and Capacity Design in Communication and Computer Networks*. Morgan Kaufmann, 2004.

[11] E. Goshi, H. Harkous, S. Ahvar, R. Pries, F. Mehmeti, and W. Kellerer, "Joint UPF and edge applications placement and routing in 5G & beyond," 2024. [Online]. Available: https://arxiv.org/abs/2407.07669

[12] Gurobi Optimization, LLC, "Gurobi Optimizer Reference Manual," 2024. [Online]. Available: https://www.gurobi.com

[13] I. Leyva-Pupo and C. Cervelló-Pastor, "Efficient solutions to the placement and chaining problem of User Plane Functions in 5G networks," *Journal of Network and Computer Applications*, vol. 197, 2022.

[14] V. Farhadi, F. Mehmeti, T. He, T. L. Porta, H. Khamfroush, S. Wang, and K. S. Chan, "Service placement and request scheduling for data-intensive applications in edge clouds," in *Proc. of IEEE INFOCOM*, 2019.

[15] B. Xiang, J. Elias, F. Martignon, and E. Di Nitto, "Joint network slicing and mobile edge computing in 5G networks," in *Proc. of IEEE ICC*, 2019.

[16] S. Lee, S. Lee, and M.-K. Shin, "Low cost mec server placement and association in 5G networks," in *Proc. of ICTC*, 2019.

[17] Y. Li and S. Wang, "An energy-aware edge server placement algorithm in mobile edge computing," in *Proc. of IEEE EDGE*, 2018.

[18] X. Xu, Y. Xue, L. Qi, X. Zhang, S. Wan, W. Dou, and V. Chang, "Load-aware edge server placement for mobile edge computing in 5G networks," in *Proc. of ICSOC*, 2019.

[19] I. Leyva-Pupo, C. Cervelló-Pastor, and A. Llorens-Carrodeguas, "Optimal placement of user plane functions in 5G networks," in *Proc. of IFIP WWIC*, 2019.

[20] I. Leyva-Pupo, A. Santoyo-González, and C. Cervelló-Pastor, "A framework for the joint placement of edge service infrastructure and user plane functions for 5G," *Sensors*, vol. 19, no. 18, 2019.

[21] K. Poularakis, J. Llorca, A. M. Tulino, I. Taylor, and L. Tassiulas, "Service placement and request routing in MEC networks with storage, computation, and communication constraints," *IEEE/ACM Transactions on Networking*, vol. 28, no. 3, 2020.