Joint placement, routing and dimensioning at the network edge for energy minimization

Maxime Elkael, Andrea Araldo, Salvatore d'Oro, Hind Castel-Taleb, Massinissa Ait Aba, Badii Jouaber

Joint placement, routing and dimensioning at the network edge for energy minimization

Maxime Elkael*, Andrea Araldo*, Salvatore D'Oro‡, Hind Castel-Taleb*, Massinissa Ait Aba†, Badii Jouaber*

*Institut Polytechnique de Paris, Telecom Sud-Paris, SAMOVAR {name.surname}@telecom-sudparis.eu
†Davidson Consulting, France {name.surname}@davidson.com
‡Northeastern University, Boston, MA, U.S.A. {n.surname}@northeastern.edu

***Abstract*—Thanks to resource virtualization, Physical Network Operators (PNOs) can share their 5G network to multiple Mobile Virtual Network Operators (MVNOs) which can leverage the shared physical infrastructure to deploy their services up to the edge. This allows much more flexibility with respect to the previous generation of cellular networks: MVNO software components can be placed at different locations, can be allocated a certain amount of virtual resources (e.g., bandwidth, CPU cycles), and be reachable via different paths. To the best of our knowledge, strategies to minimize energy consumption while satisfying Service Level Agreements (SLAs) between the PNO and the MVNOs are still largely missing, particularly if it is required to take the non-linearity of delays into account. To fill this gap, we formulate the problem of joint placement of software components, routing of user requests and resource dimensioning. SLAs are represented in terms of latency and reliability constraints. Via Column Generation, we obtain exact solutions in real-sized networks. Our numerical results show that we can save up to 50% energy in networks with up to 30 nodes compared to the state-of-the-art algorithms, which are focused on placement or resource minimization.**

I. Introduction

Motivated by the increasing demand for low-latency and high-throughput services and applications, 5G and beyond cellular networks are transitioning from rigid hardware-based architectures to virtualized and cloud-based deployments. These new architectures decouple network functions from the hardware where they execute, thus abstracting network services from the infrastructure. This substantial shift is resulting in cellular networks that (i) are easier to maintain, monitor and reprogram; and (ii) can effectively deliver the necessary performance requested by advanced services such as Augmented and Virtual Reality (AR/VR), high-definition video streaming, autonomous driving and tactile gaming.

An important enabler of this new trend is network slicing. Via network slicing, a physical network operator (PNO) can virtualize and share its physical infrastructure among multiple mobile virtual network operators (MVNOs). Specifically, the PNO provides each MVNO with a *slice* of computing and networking resources that are sufficient to meet Service Level Agreements (SLAs) negotiated between the PNO and the MVNOs. On the other hand, it has been reported that the network Operational Expenditure (OPEX) represent 25% of the total costs of operators, of which 90% is energy bills [1].

This work was supported by Agence Nationale pour la Recherche through the AIDY-F2N project, grant number ANR-19-LCV2-0012 and is based upon work partially supported by the U.S. National Science Foundation under grant CNS-1925601

Hence, in this work, we answer the following questions: *How should a PNO jointly decide (i) placement of services, (ii) routing and (iii) allocation of computational resources to each slice? And, how to take those decisions to minimize energy consumption, while satisfying SLAs?* Despite their tight inter-dependency, decisions (i), (ii) and (iii) have never been optimized jointly due to high complexity. To the best of our knowledge we are the first to formalize and solve optimally a mathematical program which takes the three aforementioned decisions jointly. Leveraging appropriate inter-relations between computational resources and network paths and applying column generation (CG), we are able to find exact solutions for real-sized networks. Thanks to our joint approach, we get rid of restrictive assumptions used in previous work which either consider that all the resources to allocate are known a priori (*e.g.*, by allocating a fixed amount of CPU to match the a-priori known demand) [2]–[6], or formalize the routing problem as a multi-commodity flow problem and assume that the "destination node" is fixed and determined a-priori [7], [8].

We differentiate ourselves from the above works in the following aspects. First, the amount of resources to be allocated to each slice is for us an optimization variable, which gives us the opportunity to meet SLA requirements at a finer grain (e.g., allocating more CPUs resources to execute services faster). Second, we fully leverage the flexibility of virtualized networks and let the PNO decide the placement of software components, as well as which nodes to activate/deactivate to minimize energy consumption. Moreover, while latency constraints are usually expressed in terms of mean delay [2] [9], we model SLAs by considering a maximum latency threshold and the corresponding level of reliability, *i.e.*, the minimum fraction of time where the threshold must be met.

By jointly deciding placement, routing and computational resource allocation policies, and by accurately modeling SLAs, we provide each slice with the precise amount of resources needed to satisfy SLAs. This allows minimizing energy and reducing the risk of over- and under-provisioning.

The rest of the paper is organized as follows. Section II surveys related work, while in Section III we lay out our model and assumptions. In Section IV, we formulate the problem of joint edge service placement, routing and computational resource dimensioning to minimize power consumption as a Non-Linear Mixed Integer Program (NLMIP). Then, in Section V we transform the NLMIP problem into a more tractable

Mixed Integer Linear Program (MILP) and describe our exact algorithm. In Section VI, we evaluate our solution and compare it with other approaches in the literature. Section VII concludes the paper with final remarks. We only give here sketch of the proofs. Full proofs can be found in our extended draft [10].

II. Related Work

Edge service placement is a widely studied problem in the literature. Most works in this field do not take the path between the service and its clients into account, and only a small fraction of those consider both energy and latency when placing edge services latency [6], [11]–[14]. Another set of works, to which the present article belongs, focuses on flow-based models that take paths into account. To our knowledge, a large part of these works solves the so-called Virtual Network Embedding (VNE) problem [3], [15]. In VNE, a virtual graph has to be placed on the physical network. Each of the nodes has a certain amount of available computing resources (*e.g.,* CPU), while edges have a certain amount of bandwidth, which are both reserved upon placing services. Various aspects such as SLA [16] or energy [5] have been studied. The most common objective is to minimize resource consumption (measured in terms of CPU and bandwidth) [15], [17]. Various techniques have been studied such as metaheuristics [18], [19], Reinforcement Learning [3], [4], [20] or integer linear programming [15], [21].

The work on VNE closest to ours is [22], where the authors consider both energy cost and latency constraints. However, as most works considering latency in VNE [2], [16], latency is calculated only based on paths and not on the reserved CPU. In fact, to our knowledge, VNE works do not consider dynamic scaling of CPU to meet the SLA.

Another work which seeks to place services and paths jointly, but does not belong to the VNE literature, is [23]. Similar to us, it models the problem as a flow problem. However, again, SLA constraints are not considered. To our knowledge, the only other work considering both propagation and processing delays while optimizing the energy is [24]. However, in this work, the processing delays are modeled as constants, which does not take their stochastic nature into account, while it is this precise aspect which makes it hard to meet 5G's stringent SLAs. The problem of allocating resources while considering paths is also investigated by other works in the context of classic multicommodity-flow (MCF) problems [25] or OSPF weight optimization [26], [27]. In general, MCF is formulated as a resource minimization problem [28]. Another common objective is load balancing [8], [29], but it is of less interest for energy as it leads to using many nodes to distribute load.

Overall, while the aspects we treat have received significant attention, our literature review shows that jointly allocating paths and resources while minimizing energy and satisfying SLAs is largely unexplored. Moreover, we also notice that our review does not suggest a good off-the-shelf candidate to perform a fair numerical comparison. Indeed, Salaht *et. al.* [30] pointed out that "the service placement problem has been highly discussed in the literature [. . .] Based on different application descriptions, network assumptions, and expected outcomes, these solutions are generally difficult to compare with each other."

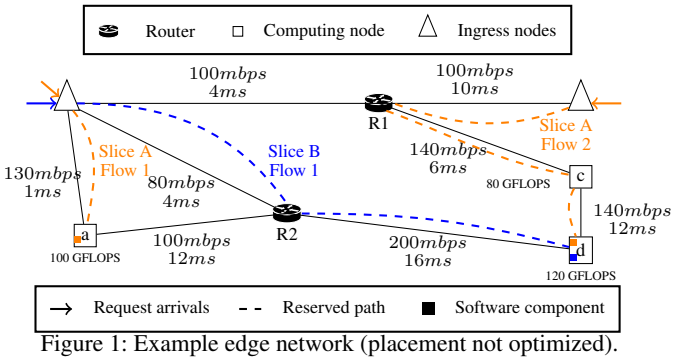

Figure 1: Example edge network (placement not optimized).

III. System model and assumptions

We consider the system depicted in Figure 1. The network is owned by a Physical Network Operator (PNO) and is modeled as a graph $G(\mathcal{V}, \mathcal{E})$. Nodes $\mathcal{V}$ can be of three types: ingress nodes, computing nodes and routers. Ingress nodes are entry points of the network (e.g., base stations (BS)) where traffic from a set of Mobile Virtual Network Operators (MVNOs) arrives. Computing nodes can host edge and cloud services to perform computation tasks (e.g., computer vision, video transcoding), and routers steer traffic throughout the network.

Each node $v \in \mathcal{V}$ of the graph is equipped with a capacity CPU_v of CPU resources. Computing nodes can host services and the PNO can allocate portion of such CPU resources to execute such services. Router and ingress nodes do not host any service and only execute networking functionalities, thus they can be modeled as nodes with 0 CPU capacity, *i.e.*, $CPU_v = 0$.

Nodes are connected via links, which are represented by the edges $\mathcal{E} = \mathcal{V} \times \mathcal{V}$ of the graph. For each link $(v, u) \in \mathcal{E}$ connecting nodes v and u, we assume a maximum bandwidth capacity $BW_{v,u}$. Each link $(v, u) \in E$ is characterized by its propagation delay $D_{v,u}$. Let us define a path $P \subseteq E$ as a sequence of links connecting distinct nodes of the graph. The propagation delay on a path P is

$$D_{prop}(P) = \sum_{(v,u)\in P} D_{v,u}. \tag{1}$$

We consider a set $\mathcal{M}$ of MVNOs willing to leverage the physical infrastructure to deploy and offer a variety of services (e.g., AR/VR, video streaming and autonomous driving) to their customers. Services offered by MVNOs have different performance requirements.

To ensure isolation, the PNO allocates a dedicated slice $r \in \mathcal{R}$ to each MVNO, where $\mathcal{R}$ being the set of all slices. If the same MVNO is allocated multiple slices, we will model them as virtually separate MVNOs. For simplicity, we assume each MVNO only runs one software component. We define software components as virtualized programs which are run on the computational nodes and can serve requests for the users of the slice they belong to.

For any given network slice r, the corresponding MVNO specifies a SLA as follows:

- A maximum tolerable latency value D_r (in ms).

- A reliability level SL_r indicating the minimum fraction of requests that should be satisfied within D_r. For example, if $D_r = 2ms$, $SL_r = 0.95$, the MVNO requires 95% of its subscribers' requests to be served within $2ms$.
- OPS_r, the number of floating point operations to perform by software component of slice r. OPS_r is a random variable and we assume the MVNO communicates the parameters describing its probability distribution.
- $\mathcal{B}_r \subset \mathcal{V}$: base stations from which service requests arrive.

Since each slice request r can include base stations located at geographically different locations, we decompose the slice into a set $\mathcal{K}_r$ of $|\mathcal{B}_r|$ flows, one for each base station. Figure 1 shows an example with two slices. Slice A has two flows, Slice B has one flow. A flow $k \in \mathcal{K}_r$ is defined by the following elements:

- The ingress/source node, $s_k \in V$
- The inter-arrival time distribution AT_k of user requests.
- The set $T_k \subset V$ of candidate computing nodes that can produce feasible solutions for flow k and can thus host services for k. Let $\mathcal{P}_k$ be the set of candidate paths with starting node s_k and any destination $t_k \in T_k$. These are all the loopless paths which have a propagation delay lower that D_r.
- The amount of bandwidth BW_k^d requested.

Let us denote with $\mathcal{K} = \bigcup_r \mathcal{K}_r$ the set of all flows requested by MVNOs. We assume that each flow $k \in \mathcal{K}_r$ is assigned a single SLA corresponding to slice r and defined as the tuple (D_r, SL_r). Since flows $k \in \mathcal{K}_r$ belong to the same slice r, the SLA of r applies to all $\mathcal{K}_r$. Therefore, we set $D_k = D_r$, $SL_k = SL_r$ and $OPS_k = OPS_r$ for all $k \in \mathcal{K}_r$.

The PNO aims to (i) serve all flows $k \in \mathcal{K}$ while satisfying the SLA requirements (which we consider to be constraints); and (ii) use the least amount of energy (which is the objective function). This is done by jointly placing the software components, allocating CPU to them, and deciding which path to use for routing requests from the BS to the software component.

We now introduce our assumptions.

Assumption 1. *Each flow k is routed between its source s_k and its chosen destination $t_k \in T_k$ on a single path.*

This assumption is justified by considering that using multiple routes for a single flow can introduce jitter, which may be undesirable for services with tight latency constraints such as the ones we consider in this paper.

Assumption 2. *We assume that OPS_k is exponentially distributed. Decision variable cpu_k is the amount of CPU that the PNO decides to allocate to the software component of flow k, expressed in floating point operations per second (FLOPS).*

The time necessary to process a service request of flow k (e.g., the time spent by the software component of that flow to process the instructions in the request) is exponentially distributed with mean $E(OPS_k)/cpu_k$.

The next proposition naturally follows.

Proposition 1. *The service time of the request is exponentially distributed with mean $\mathbb{E}[OPS_k]/cpu_k$. Therefore, the software component of each flow k can be modeled as a G/M/1 queue where AT_k is the arrival process and $\mu_k = cpu_k/\mathbb{E}[OPS_k]$ is the mean service rate.*

Assumption 3. *Queuing delay on links is negligible, and the total delay for flow k using path $P \in \mathcal{P}_k$ to reach destination node t_k is*

$$D_{tot}^k(P, \mu_k) = 2 \cdot D_{prop}(P) + D_{wait}(\mu_k, AT_k), \qquad (2)$$

where $D_{wait}(\mu_k, AT_k)$ is a random variable representing the waiting time of any request at the software component Prop. 1) and $D_{prop}(P)$ is the propagation delay (1).

This assumption, common in recent works [2], [3], [15], is reasonable since we consider that the PNO reserves to each flow the amount of bandwidth demanded by the respective MVNO on the entire route. We assume MVNOs request a sufficient amount of bandwidth, so that large queuing times on such links are unlikely to happen.

Assumption 4 (Constant Energy for Network processing)**.** *The energy consumed for routing is constant regardless of the BW.*

This assumption has been demonstrated experimentally for routers [31] and for dedicated network cards [32] [33].

We are now ready to derive the energy model of the system, which will then be used to formulate the optimization problem.

First of all, our model is based on the fact that it has empirically shown that the energy consumption of a server is an affine function of the cpu utilization [34]. We note this affine power function for node v as

$$\text{pow}(v) = c(v) \cdot l(v) + e(v),$$

where $e(v)$ is the power consumed when node v is idle (the y-intercept of $\text{pow}(v)$), $c(v)$ is the slope, and $l(v) = \mathbb{E}[\text{load}(v)]$ is the average utilization of the node (which will be computed later).[1] We assume $c(v)$ and $e(v)$ are known for each type of node, hence in the remaining we focus on calculating the load.

Recall each software component can be modeled as a G/M/1 queue (see **Proposition 1**). Hence, the average throughput of the queue modeling software component of flow k is:

$$\mathbb{E}[\text{throughput}(k)] = \mu_k \cdot (1 - \mathbb{P}_{idle}^k) = \mathbb{E}[AT_k]^{-1}$$

because in G/M/1 queues we have $1 - \mathbb{P}_{idle}^k = \frac{\mathbb{E}[AT_k]^{-1}}{\mu_k}$ [35] (as long as the queue is stable *e.g.* $\mathbb{E}[AT_k]^{-1} \leq \mu_k$). From **Assumption 2**, we derive the average amount of cpu consumed by the software component:

$$cpu_{avg} = \mathbb{E}[\text{throughput}(k)] \cdot \mathbb{E}[OPS_k] \qquad (3)$$

from which we can derive expected the percentage of CPU utilization $l(k, v)$ induced by a single software component:

$$l(k, v) = \frac{cpu_{avg}}{CPU_v} = \frac{\mathbb{E}[AT_k]^{-1} \cdot \mathbb{E}[OPS_k]}{CPU_v} \qquad (4)$$

Finally, since the queues are independent, the expected load on a node v hosting a set $\mathcal{S}$ of software components is:

$$l(v) = \mathbb{E}[\text{load}(v)] = \sum_{k \in \mathcal{S}} l(k, v) \qquad (5)$$

[1] We note the power is easily converted into an energy measured in watt-hour

IV. Problem formulation

The problem of placing all slices' flows while minimizing energy consumption and respecting SLAs has decision variables $\mathbf{x} = \{a(v), l(v) \quad \forall v \in V, \quad f_k(P), \mu_k(P), cpu_k(P) \quad \forall k \in 1 \dots K, \quad P \in \mathcal{P}_k\}$. $f_k(P)$ variables indicate whether path P is used by commodity k, hence it decides for the routing, while $\mu_k(P)$ and $cpu_k(P)$ decide respectively for the service rate and amount of CPU of commodity k, *e.g.* they determine dimensionning. They also decide placement as if they are equal to 0 it implies the software component is not placed on the node.

Note that the quantities cpu_k and μ_k have been replaced with their counterparts, *i.e.*, $cpu_k(P), \mu_k(P)$. Indeed, those quantities depend on the path P we select to route flow k, as if path P is shorter (in terms of delay), we can afford to have a lower service rate $\mu_k(P)$ by giving less $cpu_k(P)$ to the software component of flow k. Moreover, $cpu_k(P)$ and $l(k, v)$ can be calculated from $\mu_k(P)$ using (4) and Prop. 1. We also denote by $\delta_{u,v}(P)$, $\phi_v(P)$, $\omega_v(P)$ the indicator functions which respectively indicate if (u, v) is in path P, if v is the last node of P and if v is part of path P. $\delta_{u,v}(P)$, $\phi_v(P)$, $\omega_v(P)$ are input parameters of the problem.

$$\min_{\mathbf{x}} \quad \sum_{v \in V} a(v) \cdot e(v) + \sum_{v \in V} l(v) \cdot c(v) \tag{6}$$

$$\text{s.t.} \quad \sum_{1 \le k \le K} \sum_{P \in \mathcal{P}_k} f_k(P) \cdot BW_k^d \cdot \delta_{u,v}(P) \le BW_{u,v} \quad \forall (u, v) \in \mathcal{E} \tag{7}$$

$$\sum_{P \in \mathcal{P}_k} f_k(P) = 1 \qquad \forall k \in 1 \dots K \tag{8}$$

$$\sum_{1 \le k \le K} \sum_{\forall P \in \mathcal{P}_k} f_k(P) \cdot \phi_v(P) \cdot cpu_k(P) \le CPU_v \qquad \forall v \in \mathcal{V} \tag{9}$$

$$a(v) \ge \frac{1}{K} \sum_{1 \le k \le K} \sum_{P \in \mathcal{P}_k} f_k(P) \cdot \omega_v(P) \qquad \forall v \in \mathcal{V} \tag{10}$$

$$l(v) = \sum_{1 \le k \le K} \sum_{\forall P \in \mathcal{P}_k} f_k(P) \cdot \phi_v(P) \cdot l(k, v) \qquad \forall v \in \mathcal{V} \tag{11}$$

$$\mu_k(P) \in \left\{ x | x \in \mathbb{R}, \mathbb{P}(D_{\text{tot}}^k(P, x) \le D_k) \ge SL_k \right\} \qquad \forall k \in 1 \dots K, P \in \mathcal{P}_k \tag{12}$$

$$f_k(P) \in \{0, 1\} \qquad \forall f_k(P) \tag{13}$$

$$a(v) \in \{0, 1\} \qquad \forall v \in \mathcal{V} \tag{14}$$

$$l(v) \ge 0 \qquad \forall v \in \mathcal{V} \tag{15}$$

The objective (6) of the PNO is to minimize energy consumption, by turning off nodes (setting $a(v) = 0$) and reducing their dynamic energy $l(v)$.Constraints (7) prevents from using more bandwidth than what is available. Constraints (8) ensure we select exactly one path per flow k among those in the candidate set $\mathcal{P}_k$. Constraints (9) enforce CPU capacities. Constraints (10) ensure that any node that is used by a path of software component is turned on.

Constraints (11) correspond to (5). Finally, Constraints (12) are non-linear and enforce that given a chosen path and a request arrival distribution, the service rate $\mu_k(P)$ is high enough to accommodate the SLA constraints.

In order to potentially accommodate software components of many flows in single nodes (which allows to turn more nodes off and save energy), Constraints (9) suggest to have cpu_k as small as possible for any flow k. However, we cannot reduce cpu_k too much, otherwise $D_{\text{wait}}(\mu_k, AT_k)$, and thus $D_{\text{tot}}^k(P, \mu_k)$ (Eq. (2)), would excessively increase and violate the SLA (12). Therefore, $\forall k \in 1 \dots K, P \in P_k$, we can replace

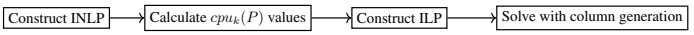

Figure 2: Block diagram of the exact solution algorithm

(12) with the optimal value of cpu_k

$$\mu_k(P) = \min_x \left\{ x | x \in \mathbb{R}, \mathbb{P}(D_{\text{tot}}^k(P, x) \le D_k) \ge SL_k \right\} \tag{16}$$

V. Solution

A. Simplification to a MILP

The goal of this section is to demonstrate that we can simplify the non linear problem(6)-(15) and cast it as a MILP while maintaining optimality. Theorem 1 is instrumental to demonstrate our claim.

Theorem 1. *If the requests from a flow k have general independent inter-arrival times, given a path $P \in \mathcal{P}_k$, the optimal service rates* (16) *can be computed in polynomial time.*

Sketch of Proof. Equation (16) suggests that we need to compute the minimum valid amount of CPU to satisfy SLA constraints. This minimum CPU value is a strictly increasing function of the propagation delay of the path considered (intuitively, the longer the path, the faster we need to process flow requests into software components to compensate for higher path latency). Finding the smallest satisfactory service rate then comes down to solving a root-finding problem with a unique root (the uniqueness is a property of the delay distribution of G/M/1 queues), which can be pre-computed using a bisection algorithm in polynomial time. □

Thanks to Theorem 1, we can pre-compute all values of $cpu_k(P)$ and take them as input of the following MILP, thus removing the non-linear constraints (12) in which the decision variables are $\mathbf{x}' = \{a(v), l(v) \quad \forall v \in V, \quad f_k(P) \quad \forall k \in 1 \dots K, \quad P \in \mathcal{P}_k\}$.

$$\begin{aligned} \min_{\mathbf{x}'} \quad & \sum_{v \in V} a(v) \cdot e(v) + \sum_{v \in V} l(v) \\ \text{s.t.} \quad & (7), (8), (9), (10), (11) \end{aligned} \tag{17}$$

B. Making the MILP more tractable

An issue with this formulation is that once we relax it into a linear program (LP), both variables $a(v)$ and $f_k(P)$ become fractional, and we would have to branch [36] on both variables to find an integer solution. Our early experiments have shown this makes even small problems unsolvable in a reasonable amount of time. Therefore, we introduce a the following set of constraints which ensures that we only have to branch on $f_k(P)$

$$a(v) \ge f_k(P) \cdot \omega_v(P), \forall v \in V, k = 1 \dots K, P \in \mathcal{P}_k. \tag{18}$$

These constraints state that if any binary variable $f_k(P)$, tied to path P, which uses node v, is equal to 1, variable $a(v)$ has to be 1. Otherwise if no path uses v, $a(v)$ will be set to its lowest possible value (*e.g.* 0) due to the objective function.

C. Column Generation-based solution

We first solve the LP relaxation of ILP eq. (17) (augmented with constraints 18, as it will later be embedded in a branch-and-bound procedure to solve the full problem. It could be

tempting to solve LP using generic techniques (*e.g.*, the simplex algorithm). However, at worst we have one variable $f_k(P)$ per loopless path P between each source and destination of each flow k. It is well known [37] that the number of paths in a graph is exponential in its number of nodes, making it impossible to enumerate those variables and solve the LP relaxation explicitly in reasonable time. Instead, we resort to *column generation (CG)* [36, §10.2]. By using CG we only consider a sequence of Reduced Master Problems (*RMP*s): $RMP^{(0)}$, $RMP^{(1)}$, etc. Each *RMP* comprises only a subset of the variables of the full LP and "excludes" all the others, implicitly forcing them to 0. In our case, we start with $RMP^{(0)}$ in which we exclude all variables $f_k(P), \forall k \in \{1, \dots, K\}, P \in \mathcal{P}_k$. Then, at every iteration i, we select one variable $f_k(P^{(i)})$ among the ones excluded in $RMP^{(i)}$ and we add it to $RMP^{(i+1)}$, *i.e.*, variable $f_k(P^*_{(i)})$ can take non-zero values in $RMP^{(i+1)}$. At the i-th iteration of CG, variable $f_k(P^*_{(i)})$ we choose to add is the one with the smallest "reduced cost". The reduced cost of each variable $f_k(P)$ measures how much the objective function would change if $f_k(P)$ was to be increased of an infinitesimal amount. Intuitively, if the reduced cost is positive, increasing the variable will increase the objective function, and if it is negative, it will decrease it. Hence by always adding variables with negative reduced cost at each step, optimality is guaranteed [36].

In the usual LP setting that uses the simplex method (and not CG), all variables are included in the problem and computing reduced costs of any single variable for each intermediate solution is straightforward (see [36], §9) and done in constant time. However, doing so for exponentially many variables like our problem requires is intractable. CG allows to skip this long computation and just find the variable with minimum reduced cost over the set of all excluded variables. Such a problem is called "pricing" and is usually solved by first proving that it is equivalent to another problem, much easier to solve. A key effort of this work is to find such an equivalent—yet easier—problem, which is summarized in the following theorem.

Theorem 2. *The pricing problem is equivalent to a bi-objective shortest path problem (BOSPP) with forbidden paths, which can be solved efficiently (e.g. [38]).*

Sketch of Proof. The first step of our proof is to decompose the reduced cost of $f_k(P)$ into the sum of 3 terms. The former is constant regardless of k and P, the second depends on dual variables associated with edges of P, and the third depends on the CPU consumed if P was to be used. Since the higher the propagation delay $D_{\text{prop}}(P)$ on a path P, the higher the CPU to be consumed to compensate for high $D_{\text{prop}}(P)$, it follows that minimizing this third term can be done by using the path with shortest delay. On the other hand, minimizing the second objective (which depends on dual variables) can be done by assigning the dual variables associated to edges as weights on such edges and finding the shortest path. Since these two objectives can be conflicting, we solve the pricing problem by computing the whole Pareto frontier for this problem and then finding the path with the smallest reduced cost among them. Finally, we prove that Equations (18) do not modify the reduced costs of the excluded variables. This is however not the case for the included variables. For this reason, computing the bi-objective shortest paths while forbidding all path associated with variables that have already been included ensures the Pareto-frontier contains the path with minimum reduced cost, *e.g.* the solution to the pricing problem. □

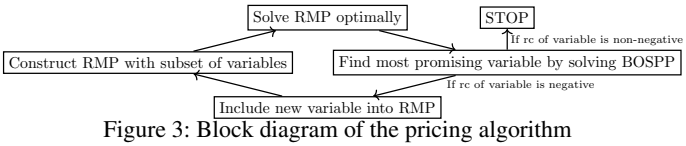

Figure 3: Block diagram of the pricing algorithm

Graph	SLAs	Min BW	Max BW	Min OPS_k	Max OPS_k	Min D_k	Max D_k
Abilene	[0.87,0.9,0.95]	10	30	150	200	600	1000
Agis	[0.87,0.9,0.95]	20	40	150	200	1400	1800
INS IXC	[0.87,0.9,0.95]	20	40	150	200	1400	1800

Table I: Random generation parameters.

Now that we have a solution to the LP relaxation (summarized up in Fig. 3), we can embed it in a branch-and-bound procedure. We choose to use the same branching rule as in [7].

VI. Performance Evaluation and Comparison

We compare our approach Energy Minimization Optimization (**EMO**) to two state-of-the-art strategies. The first one is the MCF CG algorithm from [7]. Since MCF CG minimizes resource utilization, we call it Resource Minimization Optimization (**RMO**) here. Note that we modify the original pricing problem of [7] to take SLA and CPU capacity constraints into account.The second compared approach is **UEPSO** [18], a VNE algorithm. Since VNE consists in placing a virtual graph onto a physical graph, we transform an instance of our problem into a VNE instance by transforming the set of flows into a virtual requests graph made of disjoint edges (one per flow). Since VNE assumes fixed CPU demands, we set the CPU demands equal to the lowest possible value, obtained by assuming that the path chosen has 0 propagation delay. Hence, UEPSO is evaluated in idealized conditions. We consider three graphs from the TopologyZoo dataset [39], namely Abilene, Agis and INS IXC Services, with 11, 24 and 30 nodes, respectively. We randomly generate 30 slices per graph. Each slice has a single flow. Note that after respectively 27, 21 and 11 slices SLA and capacity constraints become infeasible. Simulation details are summarized in Table I. We implement our solution in Julia[2], using the HiGHS solver for the LP solution (branching and CG are done in plain Julia). The algorithms run for a max. of 1 hour.

Figures 5 and 6 show that resource and energy minimization are contradicting objectives. In order to save energy, we may need to consume more resources. Indeed, EMO saves up to 50% of energy, when load is high (many slices), via using more resources than RMO. This counter-intuitive result can be explained by the fact that RMO tends to use shortest paths, which yields to a lower propagation delay and hence a lower CPU consumption. It also tends to select paths with fewer hops, hence consuming less bandwidth. On the other hand, minimizing energy consumption yields to taking slightly longer paths when possible, which might generate more resource consumption, but enable sharing of active nodes: the selected paths will share as many nodes as possible, so as to turn as

[2]the code is available at https://github.com/melkael/mcf-energy

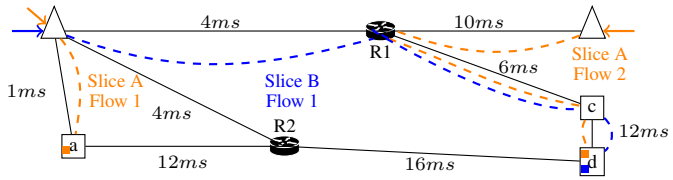

Figure 4: Ex. more energy efficient placement (compared to Fig. 1): Flow 1 of Slice B now does not activate R2.

many nodes off and consume less energy. An example of such a case is depicted in Figure 4, in which compared to Figure 1, Flow 1 of Slice B uses a longer path which enables to turn router R2 off. Those benefits come at the cost of runtime, as EMO is computationally harder to solve than RMO (Fig. 7). We relate this difficulty to the aforementioned difference between the two solutions: it is much harder to find the best way to share paths than to find a solution which, at its core, uses a large amount of the shortest paths. For example, on Agis, EMO times out for 11 slices, while the other results indicate the problem is feasible for up to 21 slices. On the other hand, for INS IXC Services, RMO obtains feasible solutions near-instantaneously while EMO often reaches the time-limit. Finally, we note that for all instances and metrics, UEPSO is much worse than both approaches. This is unsurprising since it only decides placement of software components and routing. This confirms the importance of jointly deciding placement, routing and resource dimensioning.

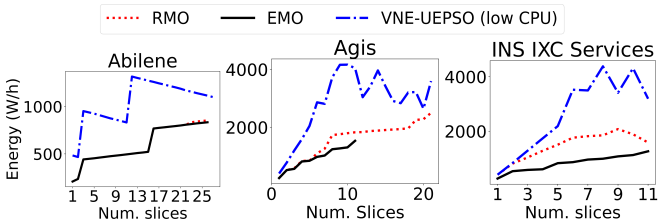

Figure 5: Energy consumption

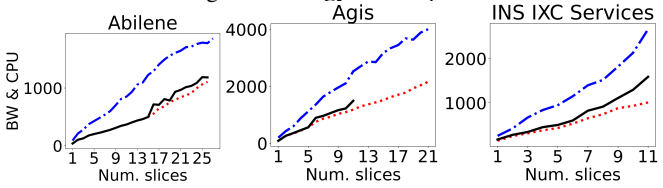

Figure 6: Resources consumption

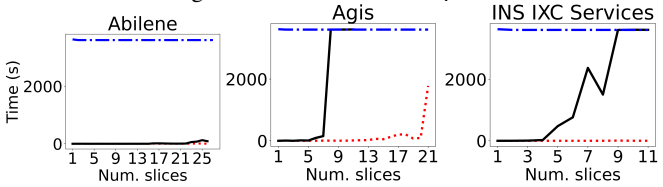

Figure 7: Runtime

VII. Conclusion

In this study, we introduced a novel model for addressing edge service placement under stringent performance demands. Our approach considers processing and propagation delays, offering infrastructure owners the flexibility to adjust CPU allocation per service to meet strict SLAs. We developed a precise solution that minimizes energy consumption while ensuring service requirements are met, crucial for effective 5G network slicing. Our findings highlight the potential for significant energy savings, up to 50% compared to a classic resource minimization approach, by strategically managing CPU allocation and service placement. Future work aims to enhance the solution's speed by incorporating strengthened inequalities and heuristic methods based on selected paths.

References

[1] GSMA, "5G Energy efficiency: Green is the new black."
[2] Esteves *et al.*, "Heuristic for edge-enabled network slicing optimization using the "power of two choices"," in *CNSM 2020*.
[3] Elkael *et al.*, "Monkey Business: Reinforcement learning meets neighborhood search for Virtual Network Embedding," *Computer Networks, 2022*.
[4] Haeri *et al.*, "VNE via Monte Carlo tree search," *Trans. on Cyb. 2017*.
[5] Botero *et al.*, "Energy efficient VNE," *ComLetters 2012*.
[6] Li and Wang, "An energy-aware edge server placement algorithm in mobile edge computing," in *EDGE 2018*.
[7] Barnhart *et al.*, "Using branch-and-price-and-cut to solve origin-destination integer MCF problems," *Operations Research 2000*.
[8] Duhamel and Mahey, "Multicommodity flow problems with a bounded number of paths: A flow deviation approach," *Networks 2007*.
[9] Huang *et al.*, "Dimensioning resources of network slices for energy-performance trade-off," in *ISCC 2022*.
[10] "Extended draft," *https://www.researchgate.net/publication/370489892*.
[11] Sharghivand *et al.*, "An edge computing matching framework with guaranteed quality of service," *Trans. on Cloud Computing 2020*.
[12] Yadav *et al.*, "Energy-latency tradeoff for dynamic computation offloading in vehicular fog computing," *IEEE Trans. on Vehicular Tech. 2020*.
[13] Badri *et al.*, "Energy-aware application placement in mobile edge computing: A stochastic optimization approach," *Trans. Par. & Dis. Sys. 2019*.
[14] Zhang *et al.*, "Cost efficient scheduling for delay-sensitive tasks in edge computing system," in *2018 SCC*.
[15] Chowdhury *et al.*, "Vineyard: Virtual network embedding algorithms with coordinated node and link mapping," *ToN 2011*.
[16] Han *et al.*, "A new VNE framework based on QoS satisfaction and network reconfiguration for fiber-wireless access network," in *2016 ICC*.
[17] Fischer *et al.*, "Virtual network embedding: A survey," *IEEE CST 2013*.
[18] Zhang *et al.*, "A unified enhanced particle swarm optimization-based virtual network embedding algorithm," *Inter. Journal of Com. Sys. 2013*.
[19] ——, "VNE based on modified genetic algorithm," *P2P Net. 2019*.
[20] Elkael *et al.*, "Improved monte carlo tree search for VNE," in *LCN 2021*.
[21] A. Aba *et al.*, "A two-stage algorithm for the VNE," in *LCN 2021*.
[22] Hejja and Hesselbach, "Online power aware coordinated virtual network embedding with 5G delay constraint," *JNCA 2018*.
[23] Fang *et al.*, "Optimising data placement and traffic routing for energy saving in backbone networks," *Trans. on Emerging Telco. Tech. 2014*.
[24] Chen *et al.*, "Optimal network slicing for service-oriented networks with flexible routing and guaranteed e2e latency," *TNSM 2021*.
[25] Ahuja *et al.*, "Network flows," 1988.
[26] Fortz and Thorup, "Internet traffic engineering by optimizing OSPF weights," in *INFOCOM 2000*.
[27] Dang *et al.*, "Monte Carlo Search Algorithms for Network Traffic Engineering," in *ECML PKDD 2021*.
[28] Kennington and Shalaby, "An effective subgradient procedure for minimal cost multicommodity flow problems," *Management Science 1977*.
[29] Raghavan and Tompson, "Randomized rounding: a technique for provably good algorithms and algorithmic proofs," *Combinatorica 1987*.
[30] Salaht *et al.*, "An overview of service placement problem in fog and edge computing," *ACM CSUR 2020*.
[31] A. Adelin *et al.*, "On the impact of monitoring router energy consumption for greening the internet," in *2010 ICGC*.
[32] Sivaraman *et al.*, "Profiling per-packet and per-byte energy consumption in the netfpga gigabit router," in *2011 INFOCOM WKSHPS*.
[33] Lu *et al.*, "Serverswitch: a programmable and high performance platform for data center networks." in *NSDI 2011*.
[34] Fan *et al.*, "Power provisioning for a warehouse-sized computer," *SIGARCH 2007*.
[35] Asmussen, *Applied probability and queues*. Springer, 2003, vol. 2.
[36] Dantzig *et al.*, *"Linear programming: Theory and extensions"*, 2003.
[37] Berge, *The theory of graphs*. Courier Corporation, 2001.
[38] Elkael *et al.*, "An exact algorithm to solve multiobjective, multi-constrained shortest path problems with forbidden paths."
[39] Knight *et al.*, "The internet topology zoo," *JSAC 2011*.