Routing and Slot Allocation in 5G Hard Slicing

Nicolas Huin
Huawei Technologies
nicolas.huin@huawei.com

Jérémie Leguay
Huawei Technologies
jeremie.leguay@huawei.com

Sébastien Martin
Huawei Technologies
sebastien.martin@huawei.com

Paolo Medagliani
Huawei Technologies
paolo.medagliani@huawei.com

Shengming Cai
Huawei Technologies
caishengming@huawei.com

ABSTRACT

5G networks will enable the creation of network slices to serve very different user requirements. Flex Ethernet (FlexE) is a standard technology that provides strict isolation between slices, also called hard slicing, by allocating capacity slots of physical links to slices. The resulting resource allocation problem is called Routing and Slot Allocation problem (RSA). We first prove that this problem is NP-hard and cannot be approximated. Then, we develop two matheuristics to efficiently solve the problem, by leveraging on a combination of Column Generation and Gauss Seidel procedures. The numerical evaluation, carried out by comparing the two matheuristics against a greedy algorithm over a realistic IP-RAN networks, shows an optimality gap smaller than 7%, while reducing the reservation cost by 4% compared to the greedy algorithm.

1 INTRODUCTION

The deployment of next generation 5G networks is paving the road for custom and personalized network services. In particular, due to the improvement in terms of end-to-end network capacity, latency and reliability, it is now possible to envision the decomposition of the physical network into several virtual sub-networks with very different requirements. Each sub-network, also called a *slice*, is independent from each other, and operated by different players, often referred to as *tenants*. The partitioning of network resources aims at guaranteeing that the requirements of tenants are met in all slices.

The importance of network slicing relies on the fact that these virtual networks can be designed to guarantee different Quality of Service (QoS) requirements. In 5G networks [5], three main use cases are commonly identified, namely enhanced Mobile Broad Band (eMBB), ultra Reliable and Low Latency Communications (uRLLC), and Massive IoT (mIoT), using the same physical infrastructure. The resources are provisioned inside each slice in such a way that the SLA (Service Level Agreement) requirements specified for each tenant can be met.

According to the isolation level, we categorize slicing technologies into *soft* and *hard* slicing. In soft slicing [1, 4], despite that QoS performance guarantees are pledged to slices, the traffic is actually multiplexed in a queuing system. A high load on a physical link may introduce an additional latency for all the slices that are routed through that link. And the traffic in one slice may impact the other slices in case of congestion. However, within hard slicing [8], each slice has dedicated resources at both physical and MAC layers. Performance misbehaviors of one slice can not have any influence on the other slices.

The main technology used to provide hard isolation is Flex Ethernet (FlexE) [10]. As mentioned in [1], it is a key enabler of 5G networks. The way FlexE can provide isolation between slices is through the reservation of resources at physical and MAC layers in a Time-Division Multiplexing Access (TDMA) fashion. The capacity of physical ports inside FlexE-enabled devices is allocated to each slice in the form of slots, i.e., multiples of a fundamental unit, normally expressed in Gigabits. Once a slot is allocated to a slice, it cannot be shared with another one. When a slice is created, FlexE slots must be reserved on physical links and user traffic must be steered through these slots. A network controller is typically taking routing and slot allocation decisions with the goal of minimizing unused resources.

In this paper, we present the *Routing and Slot Allocation (RSA)* problem for hard slicing with FlexE in 5G networks where the goal is to minimize the cost of resource reservations for a slice, under the constraint that all services in the slice are accepted. We show that this problem is NP-hard and it cannot be approximated with constant factors unless P = NP. We also present an efficient heuristic to quickly approximate the optimal solution.

The RSA problem is similar to problems such as the *multi-commodity network optimization problems with general step cost functions* [6], or the *energy-aware routing with discrete link rates problem* [2]. However, a few key differences exist. Firstly, the problem studied by [6] considers splittable flows unlike our problems where each service must be routed on a unique path. Secondly, even though [2] consider unsplittable flows, we cannot apply their method due to statistical multiplexing available in IP-RAN networks (see Section 2.3). To the best of our knowledge, we are the first to propose a column generation algorithm to solve this problem.

The structure of the paper is the following. We explain *hard slicing* in Section 2 and formally present the RSA problem in Section 2.3. We then propose an extended formulation of the problem in Section 3 and detail the column generation procedure. We then show in Section 4 two heuristics and compare, in Section 5, our heuristics on realistic 5G scenarios using IP-RAN network. Finally, we conclude this paper and discuss future works in Section 6.

2 HARD SLICING

Hard physical isolation between different slices can be acheived with Flex Ethernet (FlexE). This section presents how the technology works and the Routing and Slot Allocation problem (RSA).

2.1 Flex Ethernet for hard slicing

As shown in Figure 1, the Optical Internetworking Forum (OIF) has designed the FlexE standard as an extension of the traditional IEEE 802.3 standard for wired Ethernet. In more details, FlexE is implemented at the layer 1.5 of the OSI stack, adding a *shim*

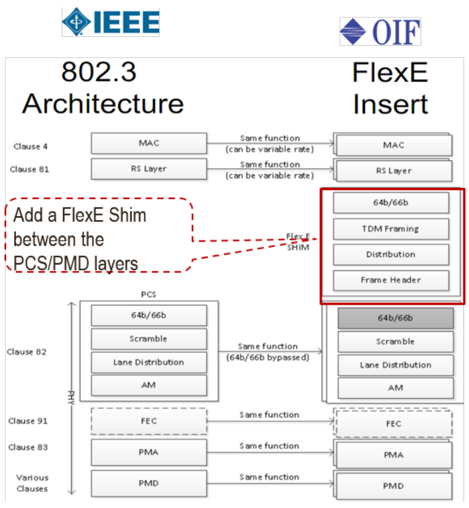

Figure 1: Extension of IEEE 802.3 to support Flex Ethernet.

layer which is in charge of allocating transmission slots to each slice with a fixed calendar, as presented in Figure 2. This rigid

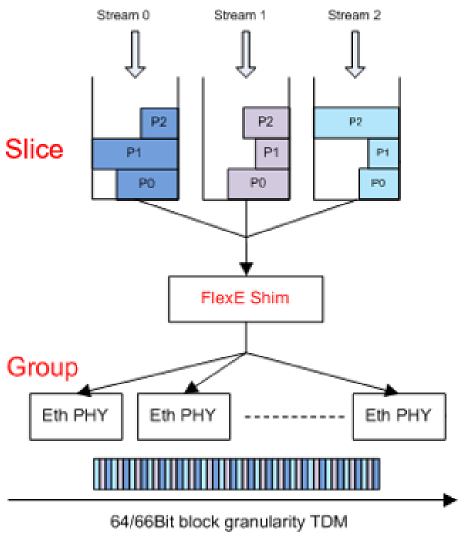

Figure 2: Role of the FlexE shim.

mapping forces the bandwidth reserved over a sub-interface to be expressed as a multiple of fundamental slot units. In the FlexE implementation we considered, the bandwidth is reserved in blocks of 5 Gb [10]. However, the first 5 slots allocated to each slice can be of 1 Gb, for a finer bandwidth reservation.

Data packets from the FlexE shim are then mixed on different PHY interfaces that carry all or part of the traffic coming from one or more sub-interfaces. The PHY interfaces are then multiplexing in a TDMA fashion, according to 64/66-bit block data line encoding. This multiplexing operation follows a rigid calendar, which is shared between the transmitter and the receiver to let the latter decodes the data when received.

2.2 Slot allocation policy

FlexE follows three bandwidth reservation rules: (i) it is necessary to activate enough slots on a link to cover all the services of the slice routed through that link; (ii) if there is enough activated slots over a link to accommodate a new service, it is not necessary to activate a new one; (iii) the slot activation sequence comes with a given order. For instance, referring to Figure 3, Service 1 of 7 Gb and Service 2 of 3 Gb need to use the same FlexE link. For Service 1, it is necessary to activate the first 5 1-Gb slots and 1 5-Gb slot, for a total of 10 Gb. This means that there are 3 Gb that are activated and not used by Service 1 and that can be used "for free" by Service 2. In particular, according to the FlexE standard, it is not possible to activate a 5-Gb slot before having activated all the 1-Gb slots. Thus, only the following link configurations are allowed: 1 Gb, 2 Gb, 3 Gb, 4 Gb, 5 Gb, 10 Gb, 15 Gb, 20 Gb and other multiples of 5 Gb slots.

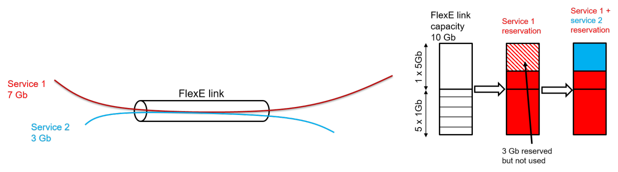
Service 1
7 Gb
Service 2
3 Gb
FlexE link
FlexE link capacity 10 Gb
1 x 5Gb
5 x 1Gb
Service 1 reservation
Service 1 + service 2 reservation
3 Gb reserved but not used

Figure 3: Scheme of FlexE link utilization

On top of the three rules mentioned above, there is another bandwidth reservation policy that must be followed in IP-RAN networks (IP networks for mobile radio networks in 4G or 5G). Figure 4 shows that in aggregation and core networks statistical multiplexing can be used to save resources. The main idea of statistical multiplexing is to assume all services crossing a link will not be active at the same time. Therefore, it is possible to reserve only a portion of the bandwidth required by the services. However, it is necessary to reserve enough bandwidth to ensure that (i) the scaled sum of the capacities of the services passes and (ii) each service alone can pass. The scaling factor applied in the aggregation and in the core network is different as it depends on the number of services using the network. This mechanism is referred to as *Convergence Ratio* (CR). The CR scaling factor can be applied only to services that explicitly support it. For example, if two services, requesting for 4 Gb each, are routed on a link with a convergence ratio of 2, 4 Gb must be allocated as to allow each service to be routed alone.

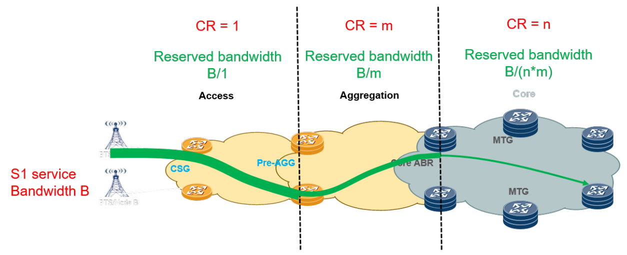

Figure 4: Convergence ratio (CR) in IP-RAN networks.

2.3 Routing and Slot Allocation problem

Let $G = (V, E)$ be the graph representing the network, where V is the set of nodes associated with the routers and E is the set of links between the routers. For each link $e \in E$ we consider a positive cost C_e per unit of bandwidth used, a capacity b_e and, a latency λ_e. In particular, b_e can be expressed as a multiple of a basic unit, referred to as slot, whose size is defined by the FlexE standard. For each link, it is possible to define a set of valid slot configurations S^e that enumerates the possible slot activations. To each link configuration $s \in S^e$ corresponds a bandwidth utilization ξ_{es}.

A slice consists of a set of demands K to be allocated in the network. Each demand $k \in K$ is characterized by a source node $s^k \in V$, a destination node $t^k \in V$, a bandwidth requirement D_k and a latency bound Λ_k. As we are considering an IP-RAN network, statistical multiplexing applies in some parts of the network for the subset $K_{\mathsf{C}} \subseteq K$ of the demands. For each link e, we define a CR factor μ^e and the amount of bandwidth used by a demand $k \in K_{\mathsf{C}}$ is given by $D_k^e = \mu^e D_k$. However, the bandwidth allocation of a link with statistical multiplexing must ensure that each demand in K_{C} can be routed alongside the demand without convergence ratio. Thus, the bandwidth usage of a link e to allocate the set of demands K_e, it is given by

$$u(e, K_e) = \sum_{k \in K_e \cap K_{\mathsf{NC}}} D_k + \max\left(\sum_{k \in K_e \cap K_{\mathsf{C}}} D_k^e, \max_{k \in K_e \cap K_{\mathsf{C}}} D_k \right) \quad (1)$$

where $K_{\mathsf{NC}} \subseteq K$ is the set of demands that are not requesting for statistical multiplexing.

The Routing and Slot Allocation problem (RSA) consists in *computing a feasible path for all demands within the slice, while respecting the link capacities and delay constraints and minimizing the cost of resource reservation in the network.*

3 COLUMN GENERATION MODEL

In this section, we first formulate the problem via an Integer Linear Program (ILP). As this model requires an exponential number of variables, we propose a pricing procedure, based on Column Generation (CG) techniques to dynamically add the necessary variables.

3.1 Problem formulation

For each demand $k \in K$, we denote by P^k the set of all possible paths between source s^k and destination t^k. The number of paths for each demand can be exponential. For each demand k and each path $p \in P^k$, a binary variable x_{kp} is equal to 1 if the path p is used by demand k, 0 otherwise. For this extended model we also consider the slot configuration variables y_{es} for each $e \in E$ and $s \in S^e$ defined in the previous section.

The following ILP *FlexE-CG* is a valid formulation for the FlexE problem.

$$\min \sum_{e \in E} C_e \sum_{s \in S^e} \xi_{es} y_{es} \quad (2a)$$

$$\text{s.t} \sum_{k \in K_{\mathsf{NC}}} \sum_{p \in P^k : e \in p} D_k x_{kp} + \sum_{k \in K_{\mathsf{C}}} \sum_{p \in P^k : e \in p} \mu^e D_k x_{pk} \leq \sum_{s \in S^e} \xi_{es} y_{es} \quad \forall e \in E \quad (2b)$$

$$\sum_{k' \in K_{\mathsf{NC}}} \sum_{p \in P^{k'} : e \in p} D_{k'} x_{pk'} + \sum_{p \in P^k : e \in p} D_k x_{kp} \leq \sum_{s \in S^e} \xi_{es} y_{es} \quad \forall e \in E, k \in K_{\mathsf{C}} \quad (2c)$$

$$\sum_{p \in P^k} x_{pk} \geq 1 \quad \forall k \in K \quad (2d)$$

$$\sum_{s \in S} y_{es} \leq 1 \quad \forall e \in E \quad (2e)$$

$$x_{pk} \in \{0, 1\} \quad \forall k \in K, p \in P^k \quad (2f)$$

$$y_{es} \in \{0, 1\} \quad \forall e \in E, s \in S^e \quad (2g)$$

The inequalities (2b) are the traditional capacity constraints on each link e where the amount of traffic for demands in K_{NC} and demands scaled with the convergence ratio in K_{C} must be smaller or equal than the size of the activated slot ξ_{es}. Inequalities (2c) ensure that each demand in K_{C} can be routed on its own. Inequalities (2d) ensure that at least one path is assigned to one demand. Inequalities (2e) guarantee that only one slot configuration is activate on each link. Remark that, as we aim at minimizing costs which are positive, it is useless to take two paths for each demand. In order to help the pricing procedure, we do not consider strict equality in (2e). The inequalities (2f) and (2g) are the integrality constraints.

Pricing procedure. Since the model *FlexE-CG* has an exponential number of variables, it is necessary to propose a pricing procedure to generate only the necessary columns (i.e., to activate variables) inside the CG algorithm. Indeed, the pricing procedure is a sub module of the CG algorithm allowing to generate only the necessary columns that improve the linear relaxation of the *FlexE-CG* model and allow to reach the optimal relaxed solution. The pricing procedure consists in solving a sub problem to define if there exists a column such that the associated constraint in the dual formulation is violated([3]).

At each step of the column generation algorithm, we obtain the optimal dual values $\delta^* \in \mathbb{R}_+^E$, $\pi^* \in \mathbb{R}_+^{E \cdot K_{\mathsf{C}}}$, $\gamma^* \in \mathbb{R}_+^K$, $\theta^* \in \mathbb{R}_+^E$ associated with the inequalities (2b), (2c), (2d), (2e), respectively. Thus, for a given demand $k \in K$, the separation of a violated dual constraint is equivalent to finding a path p such that

$$-\sum_{e \in p} D_k^e \delta_e - \sum_{e \in p} D_k \pi_e^k + \gamma_k > 0 \quad (3)$$

if $k \in K_{\mathsf{C}}$, and the following if $k \in K_{\mathsf{NC}}$:

$$-\sum_{e \in p} D_k \delta_e - \sum_{e \in p} \sum_{k' \in K_{\mathsf{C}}} D_k \pi_e^{k'} + \gamma_k > 0 \quad (4)$$

For each demand $k \in K_{\mathsf{C}}$ (resp. $k \in K_{\mathsf{NC}}$), the constrained shortest path where the cost on each link is $e \in E$ by $D_k^e \delta_e + D_k \pi_e^k$ (resp. $D_k \delta_e + \sum_{k' \in K_{\mathsf{C}}} D_k \pi_e^{k'}$) solves the pricing procedure. If solved optimally, it guarantees that a path is found if it exists. If the cost of the shortest path is strictly smaller than γ_k, then we add the column (variable) associated with this path and this demand to the problem. If for all demands, no columns are added, the column generation procedure terminates.

Note that additive end-to-end QoS constraints, such as delay, jitter or packet loss (taking the logarithm), can be integrated in the path computation procedure. In our heuristic algorithm, we use well-known algorithms such as LARAC [9] or GEN-LARAC [11] to solve the constrained shortest path problem.

3.2 Column generation algorithm

As mentioned in Section 3.1, the *FlexE-CG* formulation contains an exponential number of variables and is adapted to a column generation algorithm to solve its relaxation. Figure 5 depicts the whole procedure where the fractional solution is then fixed to integer using a rounding algorithm. Column generation relies on a pricing problem to generate variables on-the-fly instead of enumerating them in the master problem. We combine it with a constraint generation procedure for constraints (2c) to avoid any stability issue and improve convergence speed.

The algorithm works as follows: first, we warm-start the *FlexE-CG* model with a solution found using a greedy algorithm (see

Algorithm 2 for more details). Then, we proceed with the following steps:

1) The column generation alternates between solving the master problem and the pricing problems:
   a) We solve a reduced *FlexE-CG*, i.e., *FlexE-CG* with a subset of paths, using a linear solver.
   b) Using the dual values of *FlexE-CG*, for each demand, we look for constrained paths that violate (3) or (4), using the LARAC algorithm. If we find any, we add them to *FlexE-CG* and go back to step 1a).
2) We then search for any violated multiplexing constraint (2c). If none is violated, we have an optimal solution z^*_{LP} for the relaxation, otherwise we go back to step 1.

Algorithm 1 Randomized rounding

Input: A network $G = (V, E)$, link capacity b_e, $\forall e \in E$, set of demands K, set of paths $\bar{P} = \bigcup_{k \in K} \bar{P}_k$, vector $x \in \mathbb{R}^{|P|}$ of value for each path
Output: Set of paths P

```
1:  $p^*_k \leftarrow \emptyset, \forall k \in K$
2:  $K_e \leftarrow \emptyset \quad \forall e \in E$
3:  for demand $k \in K$ do
4:      while $\bar{P}_k \neq \emptyset$ do
5:          $\tilde{p} \leftarrow$ path drawn at random from $\bar{P}_k$ with $Pr(p \text{ is selected}) = \frac{x_p}{\sum_{p \in \bar{P}_k} x_p} \quad \forall p \in \bar{P}_k$
6:          if $\forall$link $e \in \tilde{p} : u(e, K_e \cup \{k\}) \leq b_e$ then
7:              $p^*_k \leftarrow \tilde{p}$
8:              for $\forall$link $e \in \tilde{p}$ do
9:                  $K_e \leftarrow K_e \cup \{k\}$
10:             break
11:         else
12:             $x_p \leftarrow 0$
13:             $P^k \leftarrow P^k \setminus \tilde{p}$
14: $K_{\mathrm{REJ}} \leftarrow \{k \in K : p^*_k = \emptyset\}$    ▷ Get set of demands not routed
15: return $\bigcup_{k \in K}\{p^*_k\} \cup Greedy(G, K_{\mathrm{REJ}}, K_e, b)$
```

Randomized Rounding. Since the column generation procedure only provides a relaxed solution for *FlexE-CG*, we need to derive an integral solution from it. We propose a randomized rounding algorithm, shown in Algorithm 1. For each demand, we randomly (with uniform distribution) choose a path amongst all the paths generated during the column generation procedure. The probability of choosing a path p is given by

$$P(k \text{ is routed on } p) = x^*_{pk}$$

where x^*_{pk} is the value of x_{pk} in the optimal solution of the relaxation of *FlexE-CG*. We check that the selected path can be routed on the current network configuration. If this is the case, we update the link-slot allocation and move to the next demand. Otherwise, we remove the path from the set of possible paths and pick a new one at random. If there is no more path in the pool, we add the demand to the list of *rejected demands*. Once all demands are considered and if the list of *rejected demands* is not empty, we try to find a solution for the rejected demands with the greedy algorithm.

Parallelization. As depicted below in Figure 5, the master problem and the pricing problems are solved iteratively but columns in the pricing can be generated in parallel. In the rounding step, we run in parallel several randomized rounding routines to ensure that the final solution will be integer and feasible. Finally, the best solution among those provided in the rounding step is selected.

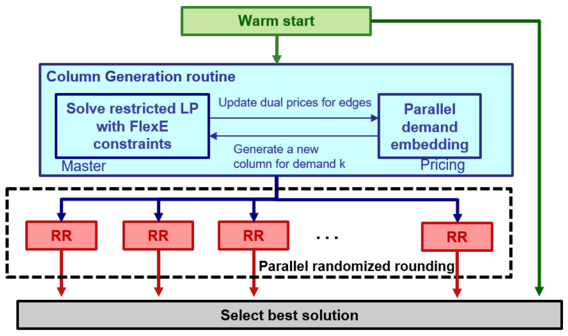
Warm start
Column Generation routine
Solve restricted LP with FlexE constraints
Update dual prices for edges
Parallel demand embedding
Generate a new column for demand k
Master
Pricing
RR
RR
RR
...
RR
Parallel randomized rounding
Select best solution

Figure 5: Algorithmic framework to solve *FlexE-CG*.

4 HEURISTICS

In this section, we present two heuristics we designed to solve the RSA problem. The first one is a simple greedy algorithm that we use as benchmark. The second one is an adaptation of a procedure from the literature [7] to solve a network planning problem with splittable flows and no considerations on statistical multiplexing.

4.1 Greedy algorithm

Algorithm 2 Greedy algorithm

Input: A network $G = (V, E)$, link capacity b_e, $\forall e \in E$, set of demands K to route, set of demands K_e on each link e
Output: Set of paths P

```
1:  $P \leftarrow \emptyset$
2:  $K_e \leftarrow \emptyset, \forall e \in E$
3:  for demand $k \in K$ do
4:      $E^k \leftarrow \{e : u(e, K_e \cup \{k\}) \leq b_e\}$
5:      $w^k(e) = \begin{cases} 1 \text{ if } A(u(e, K_e)) \geq u(e, K_e \cup \{k\}) \\ 1 + C_e \text{ otherwise} \end{cases} \quad \forall e \in E^k$
6:      Build weighted graph $G^k = (V, E^k, w^k)$
7:      Find shortest path $p$ from $s^k$ to $t^k$ in $G^k$
8:      $P \leftarrow P \cup \{p\}$
9:      for link $e \in p$ do
10:         $K_e \leftarrow K_e \cup \{k\}$
    return $P$
```

Algorithm 2 is a greedy algorithm that selects a path, for each demand, by solving a constrained shortest path problem and update the slot allocation accordingly.

For each $k \in K$, we build a weighted graph $G^k = (V, E^k, w^k)$ and search for a constrained shortest path from s^k to t^k on G^k using the LARAC algorithm [9].

The weights w^k_e are chosen in order to favor paths that do not need a bigger slot allocation to route k and is given by

$$w^k(e) = \begin{cases} 1 \text{ if } A(u(e, K_e)) \geq u(e, K_e \cup \{k\}) \\ 1 + C_e \text{ otherwise.} \end{cases}$$

where K_e is the set of demands on e and $A(x)$ returns the minimum bandwidth allocation needed to route x units of bandwidth.

We also filter out links that do not have enough capacity to route demand k. Once a path p is found, we update the sets K_e for each link on p and move on the next demand.

4.2 Gauss-Seidel algorithm

Algorithm 3 Gauss-Seidel algorithm

Input: A network $G = (V, E)$, link capacity $b_e \forall e \in E$, set of demands K, set of paths $P = \{p_k : \forall k \in K\}$
Output: Set of paths P

```
1: $E_{\mathsf{CAND}} \leftarrow E$
2: while $E_{\mathsf{CAND}} \neq \emptyset$ do
3:     $K_e \leftarrow \{k : e \in p_k\} \quad \forall e \in E$
4:     $\tilde{e} \leftarrow \arg\max_{e \in E_{\mathsf{CAND}}} C_e \times (S(u(e, K_e)) - u(e, K_e))$
5:     $E_{\mathsf{CAND}} \leftarrow E_{\mathsf{CAND}} \setminus \tilde{e}$
6:     $(P_{\mathsf{OLD}}, K_{\mathsf{OLD}}, b_{\mathsf{OLD}}) \leftarrow (P, K_{\tilde{e}}, b_{\tilde{e}})$
7:     $b_{\tilde{e}} \leftarrow \lfloor S(u(\tilde{e}, K_{\tilde{e}})) \rfloor$
8:     for demand $k \in K_{\mathsf{OLD}}$ do
9:         for link $e \in p_k$ do
10:            $K_e \leftarrow K_e \setminus k$
11:        $p_k \leftarrow \emptyset$
12:    $P_{\mathsf{NEW}} \leftarrow P \cup Greedy(G, K_{\mathsf{OLD}}, K_e, b)$
13:    if $Cost(P_{\mathsf{NEW}}) < Cost(P_{\mathsf{OLD}})$ then
14:        $P \leftarrow P_{\mathsf{NEW}}$
15:    else
16:        $P \leftarrow P_{\mathsf{OLD}}$
17:        Restore $(P_{\mathsf{OLD}}, K_{\mathsf{OLD}}, b_{\mathsf{OLD}})$ as current solution
18: return P
```

Finally, we present a Gauss-Seidel procedure in Algorithm 3 that aims at improving any existing solution, similar to the *link-rerouting* algorithm in [7]. It is a local search heuristic which tries to reduce the number of active slots of each link by rerouting demands on new paths.

More precisely, for a valid solution, the algorithm chooses the link $\tilde{e}$ with the most *free* bandwidth on it, weighted by its cost, i.e,

$$\arg \max_{e \in E_{\mathsf{CAND}}} C_e \times (A(u(e, K_e)) - u(e, K_e))$$

where E_{CAND} is the set of links not yet considered for removal. We remove all demands using e from the network and reduce the number of slot on e by one, e.g., if e was a FlexE link with a reservation of 15G, we reduce it to 10G. We then greedily route the removed demands on the new network configuration. If we obtain a lower cost with the new routing, we use it as our new best solution. Otherwise, we restore the link to its previous slot configuration, restore the removed demands on their previous paths. We continue until all links have been considered.

5 NUMERICAL RESULTS

In this section, we present numerical results to compare the algorithms on an IP-RAN scenario. The compact formulation (not presented in this paper) and the *FlexE-CG* model have been solved using CPLEX 12.7 and all algorithms have been executed on a server with 4 Intel(R) Xeon(R) CPU E5-4627 v2 @ 3.30GHz and 504GB of RAM.

5.1 IP-RAN scenario

We generate instances of an IP-RAN network with multiple domains connected to a mesh network. Each domain is composed of a set of nodes connected in single or dual-homing (access network) to a ring with probabilistic shortcuts (aggregation network). Services in slices can exists between nodes in the access networks or between a node in an access network and a node in the core network. The bandwidth requirement of services is randomly chosen between 50 Mb and 1 Gb. We consider two types of scenarios: hard slicing, denoted FlexE, and soft slicing, denoted VLAN (Virtual LAN), a candidate technology for this scenario. For VLAN, we assume that the granularity of each slot is 1 Mb, which is negligible compared to the size of the smallest demand. All algorithms are executed with a time limit of one hour. Results are averaged over 5 trials. A summary of the parameters used in the experiments is shown in Table 1.

Topology type	Instance name	# Nodes	# Edges	# Demands
Small	$\mathtt{VLAN|FlexE}_{50}$	50	60	60
Middle	$\mathtt{VLAN|FlexE}_{1250}$	1250	1600	300
Large	$\mathtt{VLAN|FlexE}_{5000}$	5000	6000	600

Table 1: Number of nodes, edges and demands for each instance type

Lower bounds: Figure 6 shows the lower bounds obtained with the compact formulation and the optimal solution z^*_{LP} of the relaxation of *FlexE-CG*. The compact formulation can provide the optimal solution for small instances; we can thus evaluate the quality of the bounds computed by *FlexE-CG*. On small VLAN scenarios (i.e., with 50 demands), the bounds provided by *FlexE-CG* are close to the optimal (less than 3%); however they are larger for the hard slicing scenarios (around 22%) as the bigger granularity of FlexE worsens the relaxation of the objective function.

On middle size instances (i.e., with 1250 demands), the compact formulation cannot be solved to optimality within the one-hour limit. Moreover, the bounds computed is much smaller than the ones computed by *FlexE-CG*. Thus, we use the bounds of *FlexE-CG* to evaluate the solutions of our algorithms on middle and large instances.

Solution quality: In Figure 7, we compare the solutions of the greedy and *FlexE-CG* algorithms, improved by the Gauss-Seidel algorithm, in terms of gap to the best lower bound, i.e., the compact solution for small instances and the *FlexE-CG* bounds for middle and large instances. The gap is computed as $(z_{\mathsf{SOL}} - \mathsf{LB})/\mathsf{LB}$, where z_{SOL} is the value of the solution and LB is the best known lower bound of the instance. Solutions of the compact formulation provided by CPLEX are not shown as CPLEX cannot return a valid solution in one hour.

First, we can see that the greedy provides good solutions, whose gap is 10.5% in the worst case. *FlexE-CG* can further improve the solution provided by the greedy algorithm and, on average, the gap is reduced by 3.8%. Moreover, *FlexE-CG* solutions are close to the optimal on soft slicing scenarios, with a gap smaller than 1.4%. The gap of hard slicing scenarios is larger, up to 6.2%, on middle size instances. However, the gap to optimality might be smaller as the bounds for hard slicing are not as tight as the ones for soft slicing.

Computational time: Finally, in Figure 8, we compare the computational time of the algorithms. The compact formulation is quite slow to be solved compared to the other algorithms. While it takes up to 36s, on average to solve small instances, *FlexE-CG* finds a solution in less than 2 s and the greedy algorithms takes

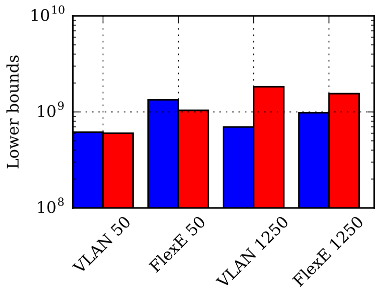

Figure 6: Average lower bounds obtained with the compact formulation (optimal for small networks) and *FlexE-CG* (1h timeout).

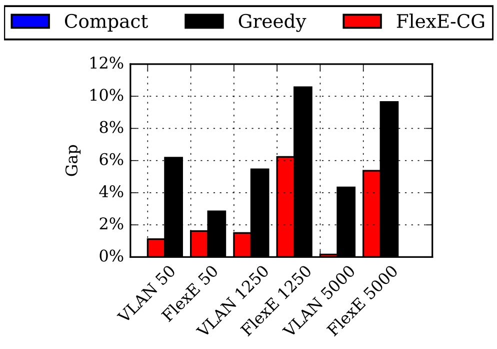

Figure 7: Average gap of each solutions for the greedy and *FlexE-CG*.

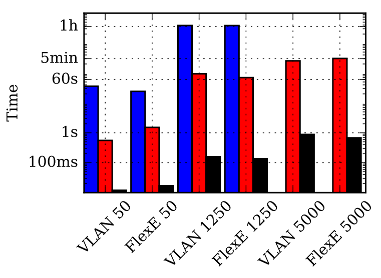

Figure 8: Average computation times of each algorithms.

less than 20 ms. As previously mentioned, the compact formulation exceeds the time budget on middle instances. The greedy remains efficient as it takes less than one second even for large instances. *FlexE-CG*, instead, is considerably slower than greedy for middle and large scale networks, but it provides for better results.

Given the different performance in terms of optimality gap and execution time of the two approaches, they could be used in parallel to efficiently solve the RSA problem. The greedy algorithm can be used to quickly accept demands in an online fashion, while *FlexE-CG* can be used to periodically reconfigure the network and minimize the total resource reservation cost.

6 CONCLUSION

In this paper, we presented the Routing and Slot Allocation problem for 5G hard slicing. We modeled the problem using mathematical programming and proposed an extended formulation, solved using column generation. We analyzed its strength against a basic integer linear formulation. Based on this extended formulation, we derived a matheuristic, referred to as *FlexE-CG*, that we benchmarked against a greedy algorithm. We also strengthened our matheuristic through an adaptation of the Gauss-Seidel procedure allowing to improve the performances of the two heuristics. We showed that the extended formulation can provide good dual bounds in a reasonable amount of time compared the the compact formulation. The derived heuristic manages to obtain an optimality gap smaller than 7%, while improving the cost value of the solutions provided by the greedy up to 4%. In future works, we will propose valid inequalities to reduce the computational time of our matheuristic and increase the dual bound. Furthermore, we will investigate on others matheuristics and exact method based on our extended formulation.

REFERENCES

[1] 5G Service-Guaranteed Network Slicing White Paper. Huawei whitepaper, February 2017.

[2] Mohamad Khattar Awad, Yousef Rafique, and Rym A. M'Hallah. Energy-aware routing for software-defined networks with discrete link rates: A benders decomposition-based heuristic approach. *Sustainable Computing: Informatics and Systems*, 13:31 – 41, 2017.

[3] V. Chvatal. *Linear Programming*. Freeman, USA, 1983.

[4] A. Destounis, G. Paschos, S. Paris, J. Leguay, L. Gkatzikis, S. Vassilaras, M. Leconte, and P. Medagliani. Slice-based column generation for network slicing. In *IEEE INFOCOM 2018 - Poster*, April 2018.

[5] X. Foukas, G. Patounas, A. Elmokashfi, and M. K. Marina. Network slicing in 5g: Survey and challenges. *IEEE Communications Magazine*, May 2017.

[6] V. Gabrel, A. Knippel, and M. Minoux. Exact solution of multicommodity network optimization problems with general step cost functions. *Operations Research Letters*, 25(1):15 – 23, 1999.

[7] Virginie Gabrel, Arnaud Knippel, and Michel Minoux. A comparison of heuristics for the discrete cost multicommodity network optimization problem. *Journal of Heuristics*, 9(5):429–445, Nov 2003.

[8] Liang Geng, Jie Dong, Stewart Bryant, Kiran Makhijani, Alex Galis, Xavier de Foy, and Slawomir Kuklinski. Network Slicing Architecture. Internet-Draft draft-geng-netslices-architecture-02, Internet Engineering Task Force, July 2017. Work in Progress.

[9] A. Juttner, B. Szviatovski, I. Mecs, and Z. Rajko. Lagrange relaxation based method for the qos routing problem. In *Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No.01CH37213)*, volume 2, pages 859–868, April 2001.

[10] OIF. Flex Ethernet 2.0 Implementation Agreement, June 2018.

[11] Ying Xiao, Krishnaiyan Thulasiraman, and Guoliang Xue. Gen-larac: A generalized approach to the constrained shortest path problem under multiple additive constraints. In *Algorithms and Computation*, 2005.