Multi-Resource Allocation for Network Slicing under Service Level Agreements

Francesca Fossati, Stefano Moretti, Stefano Secci

Multi-Resource Allocation for Network Slicing under Service Level Agreements

Francesca Fossati*†, Stefano Moretti‡, Stefano Secci*
* Cnam, Cedric, 75003 Paris, France. Email: firstname.lastname@cnam.fr
†Sorbonne Université, CNRS LIP6, 75005 Paris, France, Email: francesca.fossati@sorbonne-universite.fr
‡ CNRS UMR7243, PSL, Université Paris-Dauphine, Paris, France. Email: stefano.moretti@lamsade.dauphine.fr

***Abstract*—Network slicing in 5G aims to provide an end-to-end partition of the physical network that is optimized for the service it has to supply. Each slice needs to fulfill a Service Level Agreement (SLA), that is a contract between the slice provider and the tenants on the quality of service and reliability, expressed for a diverse set of physical resources (spectrum, link capacity, computing power, etc). For the multi-resource allocation problem in network slicing, we provide two scheduling algorithms that take into account SLA requirements in terms of minimum and nominal resource quantity demands. We show that the algorithm that considers the availability rate of the service, in addition to providing the minimum capacity, has better performances in terms of time-fairness. For both scheduling algorithms we consider a user delaying policy able to take into account SLA priority and latency requirements.**

I. Introduction

Fifth-generation (5G) communication systems have the objective to achieve different and heterogeneous services or verticals belonging to different business domains ([1]), e.g., agriculture, transport, industry etc. In particular, the community agrees on the categorization of the services in three main classes depending on the latency, frequency, bitrate and reliability requirements: enhanced Mobile BroadBand (eMBB), Ultra Reliable Low Latency Communications (URLLC) and massive Machine Type Communications (mMTC).

To support the requirements of this type of services, monolithic telecommunication network architectures are expected to be replaced by more flexible virtualized network environments created on top of the physical infrastructure by the network operator. This end-to-end virtual partition of the physical network is known as network slicing. The network slice has to be logically isolated from other slices and it is composed of a heterogeneous set of resources optimized for the set of services it has to provide [2].

A 5G network needs to fulfill Service Level Agreements (SLAs) so that - due to the heterogeneity of the requirements of each tenant - every single slice respects a specific SLA. An SLA is a contract between the provider and the customer that specifies the technical conditions of a service provisioning, i.e, connection performance, availability, liability etc., and the price of the services [3], by means of measurable parameters or metrics [4].

In this paper we provide two multi-resource scheduling algorithms, able to slice the resources between tenants and to fulfill their SLA in the challenging case in which at least one resource is not able to fully cover the tenants demand. In particular we consider three metrics: the first one is the guarantee of the *minimum service*, i.e., the minimum amount of resource that has to be guaranteed to the tenants; the second one is the *nominal capacity*, i.e., the amount of resource required in normal conditions, while the third one is the *service availability*, i.e., the measure, in percentage or units of time, of the successful service access to the tenant. We firstly propose an algorithm considering only the minimum service and the nominal capacity requirements, which we then refine to consider also service availability and to provide an allocation fairly distributed on time.

The paper is organized as follows. Section II provides a background on multi-resource allocation, together with a state of the art on recent works considering resource allocation in network slicing. In section III we provide two algorithms to allocate resources respecting the SLA. Section IV provides a numerical analysis and comparison of the two algorithms and finally section V concludes the work.

II. Background

The allocation of multiple and heterogeneous computing and network resources could be solved by allocating resources individually; however, that would be at the expense of fairness and global system efficiency [10]. To care about these aspects, it is necessary to adopt a multi-resource approach [5]. Multi-resource allocation techniques made surface first in data-center computing, where the problem of jointly managing processor, memory and storage resources made first surface. The best known scheduling algorithm in this field is the one proposing a Dominant Resource Fair (DRF) allocation to decide the number of tasks of different types to execute [6].

Other models going beyond DRF were also proposed, as mentioned hereafter. These models can be extended to any problem where a set of agents has a demand on different resource types. In network slicing, agents are tenants or verticals, demanding for a partition of network and computing infrastructure. While in [6] the demand is for a task to be executed and the allocation the number of tasks to run, in network slicing the demand is the quantity of resource needed by the tenant and the allocation is a portion of the demand assigned to the tenant, and that for multiple resources.

Formally we can model a network slice allocation problem as a pair (R, D) where R is a vector containing the avail-

able quantity of each resource and D is the demand matrix containing the quantity of each resource demanded by each tenant. Let $N = \{1, ..., n\}$ be the set of tenants and let $M = \{1, ..., m\}$ be the set of available resources, then R is a m-dimensional vector and D a $n \times m$ matrix. When each resource is sufficient to fully satisfy the users, each user receives exactly what asked for, but if it exists at least one resource $j \in M$ such that $\sum_{i=1}^{n} d_{ij} > r_j$, i.e., a resource is not enough to satisfy all tenants demands, the problem becomes challenging. Being $x = (x_1, ..., x_n)$, with $0 \leq x_i \leq 1 \; \forall i \in N$, the vector of the percentage of resources allocated to each tenant, then the allocation matrix A corresponding to x is $\begin{bmatrix} d_{11} \cdot x_1 & ... & d_{1m} \cdot x_1 \\ ... & ... & ... \\ d_{n1} \cdot x_n & ... & d_{nm} \cdot x_n \end{bmatrix}$. The allocation has to satisfy (i) non-negativity, i.e., each user should receive at least zero, (ii) demands boundedness, i.e., each user cannot receive more than its demand and (iii) efficiency, i.e., allocations belong to the admissible region $\mathcal{F}$ s.t. $\sum_{i \in N} a_{ij} = \sum_{i \in N} x_i d_{ij} \leq r_j$, $\forall j \in M$. The described DRF rule can be considered as a generalization of the MMF allocation rule because it aims to provide a MMF allocation across users dominant shares; it has additional desirable properties [6]. The DRF allocation is the solution of the following problem:

$$\begin{aligned} \text{maximize} \quad & x \\ \text{subject to} \quad & ds_i x_i = ds_j x_j, \quad \forall i, j \in N \\ & 0 \leq x_i \leq 1, \forall i \in N \end{aligned} \tag{1}$$

and $x \in \mathcal{F}$, where $ds_i = \max_j \{\frac{d_{ij}}{rj}\}$ is user i dominant share.

Other well known allocation rules are the asset fairness [6], aiming at equalizing the resource allocated to each users, the Nash product that maximizes the product of the percentage of resource to allocate [6], or the Bottleneck-Based Fairness (BBF) aiming to equalize the share received on the bottleneck resource [8] and its variant, the bottleneck max fairness [9]. [5] surveys fair multi-resource allocation rules proposed until 2018. [10] presents a framework to generalize single-resource allocation rules, while proposing two new multi-resource rules.

Recent works deal with resource allocation in network slicing. These works address the problem from different points of view. For example some of them are looking at the allocation of only one resource, such as the radio one ([11]-[15]), while others looks at the heterogeneous set of resources provided by the slice ([16]-[18]). Furthermore the objectives are various (e.g. fairness [11], [16], maximization of service provider revenue [18], etc.) and the approaches are different: centralized, i.e., it exists a resource or slice provider that takes the decision on the resource to allocate ([11], [12] [16]-[18]), or distributed ([14], [15]).

Our work differs from the cited ones in that we propose dynamic algorithms taking into account SLA. We propose centralized fair multi-resource scheduling algorithms meant to be run at each time frame, investigating how to ensure fairness looking at past scheduling outcomes.

III. SLA-aware multi-resource scheduling

A Service Level Agreement (SLA) is a contract between the resource provider (in our case the slice provider) and the final user (in our case the tenant) and it can specify (i) the minimum guaranteed and nominal capacities for each given resource, (ii) the amount of time the service is guaranteed, (iii) penalties in case the service requirements are not met, (iv) latency or jitter, (v) the service assistance, etc. [3]. Between this specification the one strictly linked to the resource allocation is (i), while the ones related to a scheduling process are (ii)-(iv).

In this section we propose algorithms considering the common way SLA management appears in network slicing specifications, i.e., including a minimum resource amount or capacity and a success rate in serving the demand, i.e., the availability. Furthermore an algorithm supporting user delaying in the service queue is proposed in order to satisfy latency requirements.

In the following, we (a) model the problem (III-A), (b) establish a users delaying policy (III-B), (c) define how to allocate the resources, under the constraint of guaranteeing a minimum share of resource (III-C), and (d) propose two scheduling algorithms (III-D,III-E).

A. Problem statement

Given a time frame t, the resource allocation problem is a tuple $(D_t, D_t^m, \gamma_t, \nu_t, R_t)$ where D_t is the demand matrix, D_t^m is the matrix containing the minimum amounts of resource to allocate to each tenant, R_t is the available resource, γ_t is a vector containing the priority index of each tenant, ν_t is a vector containing the availability rate of each tenant, i.e., it contains the percentage of time the tenant was served, with at least the minimum resource. The priority index γ_t is linked to the latency of the service: if the service requires a low latency, its priority is high and the value of γ_t is low; if not, the priority is lower and the value of γ_t is higher. For instance URLLC services are characterized by higher priority indexes compared to eMBB and mMTC services.

Key assumptions are as follows: (i) the demand processing time is discrete, i.e., the matrices D_t and D_t^m collect the information about the users demand in the time frame $t-1$ and they are treated in the same moment when the time frame t starts; (ii) the priority index γ_t depends only on the service latency, while in general it can be linked also to the tenant importance, measurable by how much a tenant is willing to pay for a service. Note that in the following when we avoid the subscript t in the notation we are considering a generic instant of time t.

B. User delaying policy

When the slice provider is not able to satisfy the minimum allocation of each tenant ($\sum_{i=1}^{n} d_{ij}^m > r_j$, for at least one resource j), it is necessary to introduce a process to eliminate tenants and put them in hold for the next time slot. The order on which tenants have to be held needs to take into account the tenant priority index. Different user delaying policies are

$$\gamma = [1\ 2\ 1\ 1\ 2\ 2]$$

$i =$

2	6	5	1	4	3

(a) Considering the priority index

$$\gamma = [1\ 2\ 1\ 1\ 2\ 2] \qquad \nu = [.8\ .7\ .2\ 0\ 1\ .2]$$

$i =$

5	2	6	1	3	4

(b) Considering the priority index and the availability rate

Fig. 1: Order of users delaying

possible. Here we propose one that takes into account only the priority index, and one that considers both the user priority index and the current availability rate of each tenant.

The two policies are depicted in Figure 1. In the first one only the priority index is used as decision variable. The order of users to remove is established ordering the users from the lower to the higher priority ones, and if tenants belong to the same class of service (i.e. they have the same index) the choice is done randomly. In Fig. 1a the vector is giving the position index of the users and it is clear that firstly tenants with priority 2 are eliminated, and then are the ones with priority 1. The second policy is based on the idea that we should firstly consider the priority index and then look at the value of ν. Higher values of ν correspond to higher percentages of time in which tenants are served in the past. It follows that, inside the same class of service we should eliminate users from the one with highest value of availability rate to the one with a lower value in order to enforce fairness within the same class of service. In Fig. 1b, for example, the first tenant of the list is the 5th one because it has priority 2 and it was served 100% of the times, while the last is the 4th that has high priority and it was never served until the considered instant of time.

C. Multi-resource allocation with minimum demand

To solve the multi-resource allocation we chose to use the DRF rule described in Section II, because of its good properties in terms of fairness. However, to provide an allocation that guarantees a quantity of resource not inferior to the minimum demand we need to modify the capacity constraint of the optimization problem[1]. In this case we can bound the percentage of resource to allocate to each tenant to be bigger than the minimal ones calculated as $x_i^m = max_j(x_{ij}^m) = max_j(\frac{d_{ij}^m}{d_{ij}})$.

It follows that the problem to solve is:

$$\begin{aligned} \text{maximize} \quad & x \\ \text{subject to} \quad & x \in \mathcal{F}, \\ & ds_i x_i = ds_j x_j, \quad \forall i,j \in N \\ & x_i^m \leq x_i \leq 1, \forall i \in N \end{aligned} \tag{2}$$

[1]Other rules can be used or proposed; it remains important that they are adapted to guarantee the minimum allocation to each tenant.

Algorithm 1 Allocation considering minimum capacity requirements (MIN-CAP)

Input: R, D, D^m, N, M, γ
Output: A

$o \leftarrow$ ordered vector of users using γ
$N^* \leftarrow N$
$P \leftarrow \emptyset$
$count \leftarrow 1$
while it $\exists$ at least one $j \in M$ s.t. $\sum_{i \in N^*} d_{ij}^m > r_j$ **do**
  $i^* \leftarrow o(count)$
  $N^* \leftarrow N^*_{-i^*}$
  $P \leftarrow P_{+i^*}$
  for k=card(P):1 **do**
    if $\sum_{i \in N^*} d_{ij} + d_{kj} \leq r_j, \forall j \in M$ and $k \in P$ **then**
      $N^* \leftarrow N^*_{+k}$
      $P \leftarrow P_{-k}$
    end if
  end for
  $count \leftarrow count + 1$
end while
if it $\exists$ at least one j s.t. $\sum_{i \in N^*} d_{ij} > r_j$ **then**
  $a_i \leftarrow$ solution of (2) $\forall i \in N^*$
else
  $a_i \leftarrow d_i \ \forall i \in N^*$
end if
$a_i \leftarrow zeros(m) \ \forall i \notin N^*$

As already mentioned, it is possible that the optimization problem has no solution when there are no enough resources to satisfy tenants minimal demands. For this reason we introduced the user delaying policy. In the following, we combine the proposed resource allocation and the delaying policy so that the two scheduling algorithms are able to satisfy SLA constraints.

D. Baseline algorithm: minimum capacity (MIN-CAP)

We propose a baseline algorithm called 'MIN-CAP', that uses the first re-order of the users, i.e., the one considering only the priority index.

The allocation resulting from (2) is calculated after having checked that the minimum demands for each tenant can be satisfied. In the case this is not possible, the tenants are, one at time, delayed using the proposed order. Each time a user is delayed the algorithm checks if there is one or more than one user already delayed that can be re-introduced because its own minimal demand can be satisfied. Obviously the order used for the re-introduction check follows the reverse order of the tenants delaying.

The pseudo-code shows the algorithm used at time slot t. The notation is lightened avoiding the subscript t.

E. Refined algorithm: considering service availability guarantees (REF-MIN-CAP)

The MIN-CAP algorithm does not take into account SLA requirements on the service availability. For example, if a

Algorithm 2 Refined algorithm (REF-MIN-CAP)

for t = 0:T **do**
  Input: $R_t, D_t, D_t^m, N, M, \gamma_t, \nu_t$
  Output: A_t

  We avoid from here the subscript t

  $o \leftarrow$ ordered vector of users using γ and ν
  $N^* \leftarrow N$
  $P \leftarrow \emptyset$
  $count \leftarrow 1$
  while it $\exists$ at least one $j \in M$ s.t. $\sum_{i \in N^*} d_{ij}^m > r_j$ **do**
    $i^* \leftarrow o(count)$
    $N^* \leftarrow N^*_{-i^*}$
    $P \leftarrow P_{+i^*}$
    for k=card(P):1 **do**
      if $\sum_{i \in N^*} d_{ij} + d_{kj} \leq r_j, \forall j \in M$ and $k \in P$ **then**
        $N^* \leftarrow N^*_{+k}$
        $P \leftarrow P_{-k}$
      end if
    end for
    $count \leftarrow count + 1$
  end while
  update of ν
  if it $\exists$ at least one j s.t. $\sum_{i \in N^*} d_{ij} > r_j$ **then**
    $a_i \leftarrow$ solution of (2) $\forall i \in N^*$
  else
    $a_i \leftarrow d_i \ \forall i \in N^*$
  end if
  $a_i \leftarrow zeros(m) \ \forall i \notin N^*$
  $t = t + 1$
end for

tenant is left in a standby state at scheduling time slot t in order to guarantee the minimum level of service to the other tenants, it shall likely be served in the time slot $t + 1$, or not too late. Thus, we want an algorithm that is time-fair, i.e., when the number of time slots T is big enough, the waiting time for each tenant is similar and kept small.

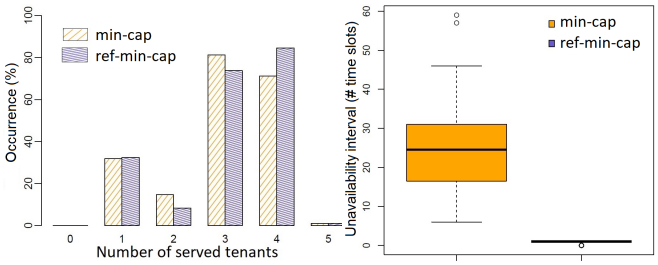
With the refined REF-MIN-CAP algorithm, in order to provide time-fair allocations we take into account the availability rate ν and we use the same algorithm, changing only the user delaying policy. In particular we use the second policy considering both ν and γ.

IV. Numerical evaluation

We provide two cases for the numerical analysis. In the first one we compare the two algorithms in the case in which the priority index of each user is the same. This means that for the first algorithm the users delaying policy is random, while for the second one it depends only on the availability rate. We consider 200 time slots and a slicing problem with 3 resources meant to represent live memory, vCPU, link capacity, and 5 slices in the scheduling queue; the resource amounts are respectively fixed to 2000 GB, 150 vCPU and 50 Gbps.

API Name	Memory (GB)	vCPUs	Gbps	Instance Type
m4.10xlarge	160.00	40.00	10.00	General purpose
m4.16xlarge	256.00	64.00	25.00	General purpose
c5.9xlarge	72.00	36.00	10.00	Compute optimized
c5.18xlarge	144.00	72.00	25.00	Compute optimized
c4.8xlarge	60.00	36.00	10.00	Compute optimized
r4.8xlarge	244.00	32.00	10.00	Memory optimized
r4.16xlarge	488.00	64.00	25.00	Memory optimized
x1.16xlarge	976.00	64.00	10.00	Memory optimized
x1.32xlarge	1952.00	128.00	25.00	Memory optimized
x1e.16xlarge	1952.00	64.00	10.00	Memory optimized
x1e.32xlarge	3904.00	128.00	25.00	Memory optimized
p3.8xlarge	244.00	32.00	10.00	Accelerated comput.
p3.16xlarge	488.00	64.00	25.00	Accelerated comput.
p2.8xlarge	488.00	32.00	10.00	Accelerated comput.
p2.16xlarge	732.00	64.00	25.00	Accelerated comput.
g3.8xlarge	244.00	32.00	10.00	Accelerated comput.
g3.16xlarge	488.00	64.00	25.00	Accelerated comput.
f1.16xlarge	976.00	64.00	25.00	Accelerated comput.
h1.8xlarge	128.00	32.00	10.00	Storage optimized
h1.16xlarge	256.00	64.00	25.00	Storage optimized
d2.8xlarge	244.00	36.00	10.00	Storage optimized
i3.8xlarge	244.00	32.00	10.00	Storage optimized
i3.16xlarge	488.00	64.00	25.00	Storage optimized

TABLE I: Amazon EC2 istances

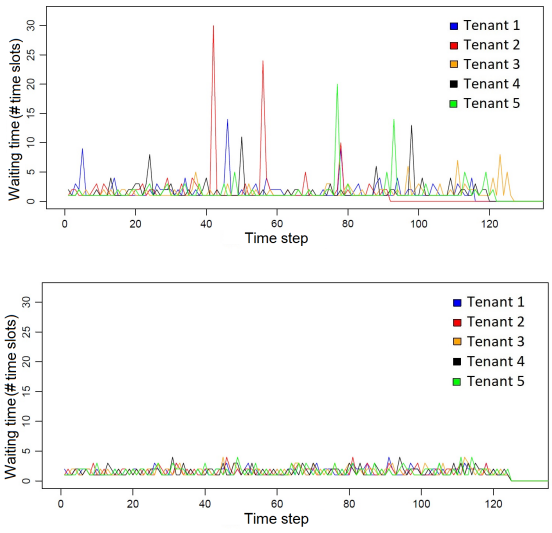


(a) Average number of served clients (b) unavailability gap (best served - worst served)

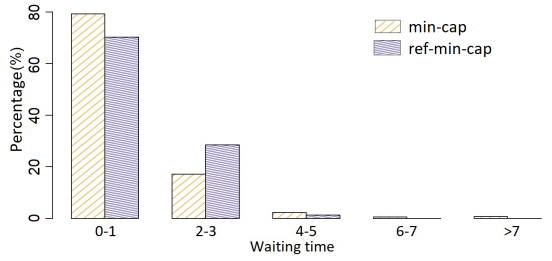
Fig. 2: Number of served client and unavailability gap

We randomly generate the slice demands using a subset of Amazon EC2 instances [19] (Table I) so that the congestion levels (fraction of the global demand not allocated due to resource scarcity) is heterogeneous. The minimum demand associated to each tenant is the minimum template available for each 'Instance Type': for example if the tenant demand instance type is 'compute optimized', then its minimum demand is 72 GB, 36 vCPUs, 10 Gbps (c5.9xlarge). We repeat the simulation 100 times.

We are interested in evaluating the performance of the two proposed algorithms. Figure 2 shows the average number of clients that are served at each time slot and the boxplots of the gap between the number of times the best served and the worst served tenant are served. From Figure 2a we can notice that there are no big differences in the number of served client between the two algorithms. The REF-MIN-CAP one is slightly better because it increases the number of times 4 tenants are served. The major differences between the two algorithms are shown in Figure 2b. It is clear that for REF-MIN-CAP after 200 time slots the number of time tenants are not served is the same (the gap is between 0 and 2 time slots) while with MIN-CAP the best served client is served a higher number of times, with a median value around 25.



(b) Waiting time of one repetition using REF-MIN-CAP



(c) Waiting time histogram

Fig. 3: Waiting time analysis.

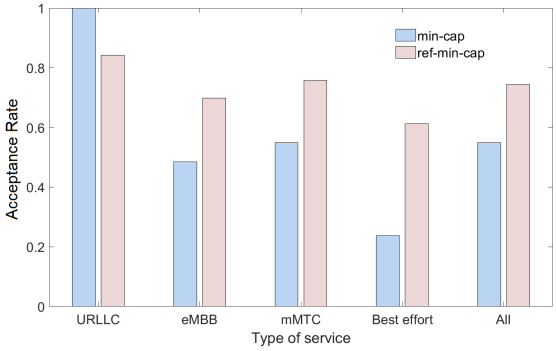


Fig. 4: Service availability for different slices.

Figure 3 shows the results of the waiting time analysis. Figure 3a and 3b show the waiting time of one simulation repetition on the 200 time slots. We can notice that MIN-CAP does not prevent the waiting time from growing excessively (in our case it can reach 30 time slots). This is due to the absence of the availability index that considers past service times. This index, that is present in REF-MIN-CAP, avoids an excessive growth of the waiting time and in particular, in our case, the waiting time does not exceed 5 time slots (Figure 3b). Figure 3c confirms that the waiting time of REF-MIN-CAP is bounded at 5, while for MIN-CAP, even if with a low probability, it can take higher values.

In the second numerical case we want to compare the two

Service type	Instance type	γ
URLLC	Accelerated computing	1
eMBB	Compute optimized Memory optimized	2
mMTC	Storage optimized	3
Best effort	General purpose	4

TABLE II: Adopted mapping of EC2 templates to 5G slices.

algorithms when users have different priorities linked to the latency required by the service. Following what recommended in [20], the importance of the latency requirement is high for URLLC services (implying not only very low propagation delay but also very low coding and processing time), medium for eMBB services and low for mMTC services. Moreover, mMTC service are expected to call for in-network storage and reformatting of exchanged IoT or machine generated data. Finally, eMBB services are expected to call for an amount of computing resources proportional to the bit-rate, which is meant to be an important one, in the order of the Gbps. Given these qualitative requirements, at first instance, we consider four levels of priority: three characterizing the three classes of services proposed for the 5G, and one characterizing the best effort class. Given the lack of slice templates in current 5G specifications, we propose to derive and differentiate them using Amazon template instance types in Table I. According to the service requirement assumptions above, in Table II we associate the 'accelerated computing' template to URLLC, the 'storage optimized' one to mMTC, the 'compute and memory optimized' one to eMBB and the 'general purpose' one to the best effort class; the value of γ is an arbitrary one, it just indicates the priority order.

Hence we randomly generate the slice demands using the differentiated subset of Amazon EC2 instances [19] (Table I), including the instance type for class differentiation as per Table II, with an heterogeneous level of congestion, and the minimum demand considered for each tenant set as the minimum template available for each 'Instance Type'. We repeat the simulation 100 times.

We plot in Fig. 4 the availability performance, i.e., the percentage of time a tenant is served when it submits a request. We do not differentiate among the case where the demand is a new demand, and the one where a demand comes from a tenant that is waiting to be served for already some time-slots. We can clearly see that the best-served tenants are the ones requiring an URLLC service; however, the REF-MIN-CAP algorithm shows more balance in the availability performance also for low priority classes. Non differentiating the service types (columns All), the REF-MIN-CAP brings a better global availability with respect to MIN-CAP.

We then consider in Fig. 5 the waiting time of the tenants, i.e., the time passing from the submission of the demand and the time in which the service is provided. We do not plot the outlier values of the boxplot, but we summarize the information about them in Table III. One can clearly observe that the second algorithm has better performance because, with a probability of at least 75%, the tenants are served when they submit the demand, independently of the type of service they

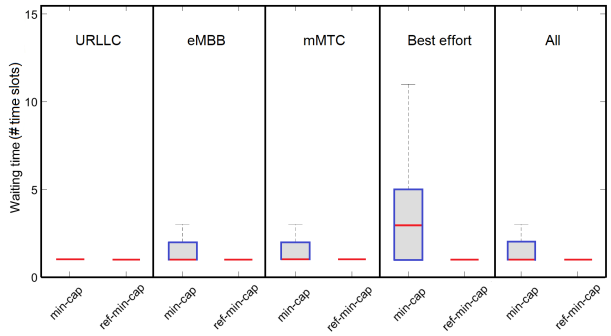


Fig. 5: Boxplot distribution for the waiting time.

Service Type	MIN-CAP		REF-MIN-CAP	
	Probability	Range	Probability	Range
URLLC	10^-4	2	0.18	2
eMBB	0.1	$[4, 56]$	0.2	$[2, 45]$
mMTC	0.09	$[4, 15]$	0.19	$[2, 9]$
Best effort	0.06	$[12, 46]$	0.2	$[2, 16]$

TABLE III: Boxplots outliers

require. On the other hand, the first algorithm differentiates the tenants, serving with a probability of at least 75% the URLLC tenants in 1 time-slot, and the eMBB and the mMTC tenants in 1 or 2 time-slot and the best effort in maximum 5 time-slots. Nonetheless, we need to consider that there is a not negligible probability that the service time gets high. Analyzing Table III we notice that for URLLC services, with the REF-MIN-CAP, it is possible for tenants to wait 2 time slots before being served, while with the first algorithm the probability that tenants are delayed is negligible. For all the other types of service, tenants can wait a long time before being served, but introducing the considerations about the availability rate in the delay policy, we can reduce the waiting time. In particular both the lower bound and the upper bound of the time-slot range decrease using REF-MIN-CAP. This shows how the second algorithm tries to enhance the global availability for the tenants, while providing a scheduling algorithm that is "time-fair".

We can conclude that REF-MIN-CAP differs from MIN-CAP in taking into account the tenants service history, by

- avoiding an excessive increase of the waiting time;
- improving the overall availability of the system.

When taking into account services belonging to different classes, i.e., with different priorities, considering the availability rate slightly penalizes tenants with high priority, while it can improve the satisfaction of the other tenants, decreasing the waiting time. This behavior can certainly be marginally modified toward more specific requirements adequately tuning algorithm parameters.

V. Conclusions

We presented two centralized scheduling algorithms (MIN-CAP and REF-MIN-CAP) to allocate resources in network slicing systems. The algorithms take into account the latency requirement for different services and the SLA requirement in terms of minimal demand that has to be allocated to each tenant. The first baseline algorithm considers only the tenant priority, while the second one considers also the availability rate with the aim to produce time-fair allocations. The simulations show that the second algorithm adequately counterbalances service availability with time-fairness.

The proposed algorithms, and in particular the second one, represent starting points to customize network slicing allocation performance toward more specific SLA requirements. For example, it may be interesting to introduce the notion of demand expiration time, i.e., a deadline within which the service must be provided.

Acknowledgement

This work was partially funded by the ANR MAESTRO-5G (https://maestro5g.roc.cnam.fr) project (ANR-18-CE25-0012).

References

[1] NGMN. "5G white paper". *Next generation mobile networks*, 2014.
[2] 5G Americas, "Network Slicing for 5G and Beyond". *White Paper*, 2016.
[3] D. Verma, "Service level agreements on IP networks", *Proceedings of the IEEE* 92: 1382-1388, 2004.
[4] M. A. Habibi, et al, "The Structure of Service Level Agreement of Slice-based 5G Network". *arXiv preprint arXiv:1806.10426*, 2018.
[5] P. Poullie, T. Bocek, B. Stiller."A survey of the state-of-the-art in fair multi-resource allocations for data centers." *IEEE Transactions on Network and Service Management*, 15.1: 169-183, 2018.
[6] A. Ghodsi, et al., "Dominant resource fairness: fair allocation of multiple resource types." *USENIX NSDI 2011*.
[7] O. Wlodzimierz, et al., "Fair optimization and networks: A survey." *J. of Applied Mathematics*, 2014.
[8] Y. Etsion, T. Ben-Nun and D. G. Feitelson, " A global scheduling framework for virtualization environments." *IEEE Int. Symposium on Parallel and Distributed Processing*, 2009
[9] T. Bonald, J. Roberts, " Multi-resource fairness: Objectives, algorithms and performance." *ACM SIGMETRICS Performance Evaluation Review* 43.1, 2015.
[10] F. Fossati, S. Moretti, F. Perny, S. Secci, " Multi-resource allocation for network slicing." HAL technical report, hal-02008115, 2019.
[11] P. Caballero, et al.,"Multi-tenant radio access network slicing: Statistical multiplexing of spatial loads." *IEEE/ACM Transactions on Networking (TON)* 25.5: 3044-3058, 2017.
[12] M. Jiang, M. Condoluci, T. Mahmoodi, "Network slicing in 5G: An auction-based model." *IEEE ICC 2017*.
[13] P. Caballero, et al., "Network slicing games: Enabling customization in multi-tenant networks." *IEEE/ACM Transactions on Networking* 27.2: 662 - 675, 2019.
[14] Y. Xiao, et al., "Distributed Resource Allocation for Network Slicing Over Licensed and Unlicensed Bands." *IEEE Journal on Selected Areas in Communications* 36.10: 2260-2274, 2018.
[15] H. Halabian, "Distributed Resource Allocation Optimization in 5G Virtualized Networks." *IEEE Journal on Selected Areas in Communications* 37.3: 627-642, 2019.
[16] M. Leconte, et al, "A resource allocation framework for network slicing." *IEEE INFOCOM 2018*.
[17] W. Guan, et al., "A service-oriented deployment policy of end-to-end network slicing based on complex network theory." *IEEE Access* 6: 19691-19701, 2018
[18] G. Wang, et al., "Resource Allocation for Network Slices in 5G with Network Resource Pricing." *IEEE GLOBECOM 2017*.
[19] Amazon EC2 instances comparison: https://www.ec2instances.info.
[20] M. Series, "IMT VisionFramework and overall objectives of the future development of IMT for 2020 and beyond." *Recommendation ITU: 2083-0*, 2015.