kaNSaaS: Combining Deep Learning and Optimization for Practical Overbooking of Network Slices

Sergi Alcalá-Marín
sergi.alcala@imdea.org
IMDEA Networks Institute & UC3M
Madrid, Spain

Antonio Bazco-Nogueras
antonio.bazco@imdea.org
IMDEA Networks Institute
Madrid, Spain

Albert Banchs
albert.banchs@imdea.org
IMDEA Networks Institute & UC3M
Madrid, Spain

Marco Fiore
marco.fiore@imdea.org
IMDEA Networks Institute
Madrid, Spain

ABSTRACT

Cloud-native mobile networks pave the road for Network Slicing as a Service (NSaaS), where slice overbooking is a promising management strategy to maximize the revenues from admitted slices by exploiting the fact they are unlikely to fully utilize their reserved resources concurrently. While seminal works have shown the potential of overbooking for NSaaS in simplistic cases, its realization is challenging in practical scenarios with realistic slice demands, where its actual performance remains to be tested. In this paper, we propose kaNSaaS, a complete solution for NSaaS management with slice overbooking that combines deep learning and classical optimization to jointly solve the key tasks of admission control and resource allocation. Experiments with large-scale measurement data of actual tenant demands show that kaNSaaS increases the network operator profits by 300% with respect to NSaaS management strategies that do not employ overbooking, while outperforming by more than 20% state-of-the-art overbooking-based approaches.

CCS CONCEPTS

• **Networks → Network resources allocation**; **Traffic engineering algorithms**.

KEYWORDS

Network Slicing, 5G, Forecasting, Optimization, Overbooking

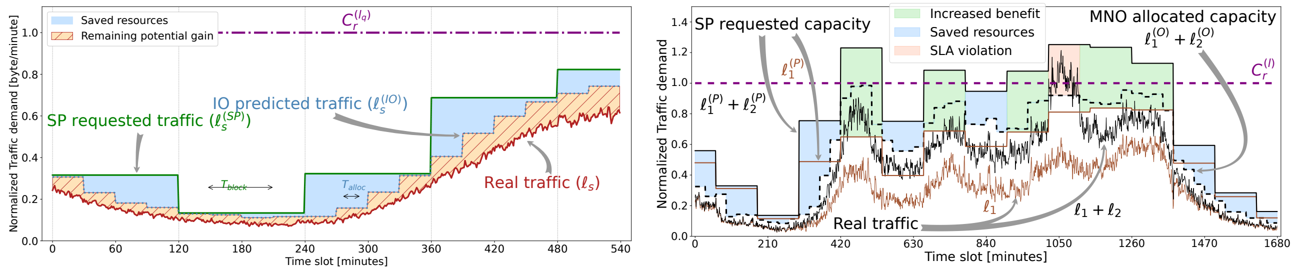
ACM Reference Format:
Sergi Alcalá-Marín, Antonio Bazco-Nogueras, Albert Banchs, and Marco Fiore. 2023. kaNSaaS: Combining Deep Learning and Optimization for Practical Overbooking of Network Slices. In *The Twenty-fourth International Symposium on Theory, Algorithmic Foundations, and Protocol Design for Mobile Networks and Mobile Computing (MobiHoc '23), October 23–26, 2023, Washington, DC, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3565287.3610265

1 INTRODUCTION

Softwarization has marked the evolution of mobile network infrastructures over the past decade, and Mobile Network Operators (MNOs) are today experimenting with proofs-of-concept and early deployments of cloud-native network technologies, supported by major cloud service providers [5, 32]. The dramatic increase in flexibility granted by production-grade cloud-native mobile network architectures will finally open new and long-envisioned business opportunities for MNOs. One of the most promising is network slicing, which abstracts a single physical infrastructure into multiple logical instances, or slices [25]. Each network slice is dedicated to specific traffic flows (*e.g.*, the video streaming demand generated by mobile clients of a given platform) and is configured so as to provide strong guarantees that the Service Level Agreement (SLA) for such traffic is met (*e.g.*, in terms of latency, throughput, or jitter).

Cloud-native NSaaS management. Cloud-native network architectures offer a natural support to network slicing operations [29]: they allow assigning dedicated resources (*e.g.*, spectrum, transport capacity, compute or memory resources, depending on the target network domain) to each slice [1], configuring dynamically the Virtual Network Functions (VNF) according to the SLA of each slice [24], and monitoring the fulfillment of such SLA [9]. As a result, the cloudification of networks implicitly paves the way to the realization of Network Slicing as a Service (NSaaS) models. Here, MNOs deliver slices to vertical customers, *i.e.*, Service Providers (SPs) who are able to configure their assigned slices up so as to best run their applications [38]. The NSaaS model ultimately creates a new marketplace that allows operators to maximize their revenue through an appropriate slice brokering [3].

Network slicing has drawn significant attention from the research community in the past years, and studies have tackled many challenges in the practical implementation of this paradigm. Among those, admission control and resource allocation are central tasks: a great portion of the potential advantage that NSaaS can bring to MNOs depends on correct choices on whether to accept a slice, and, if so, with what dedicated resources. As we will discuss in detail in Sec. 2, prior works have addressed both these problems, possibly in a joint fashion. Yet, the vast majority of the studies in the literature overlooks an important degree of freedom for the operator, *i.e.*, its flexibility in allocating resources that are not necessarily those specified by the SLA, as explained next.



(a) (b)

Figure 1: Overbooking in NSaaS, with notation. (a) Real traffic generated by one mobile service, requested slice capacity by the associated service provider (SP) at every slice brokering interval T_{SLA}, and actual capacity allocated by the Mobile Network Operator (MNO) thanks to a fast orchestration at periodicity T_{RA} and a fine-tuning of allocated resources closer to the actual service demand. We highlight the resource savings with respect to a blind allocation of the exact capacity requested by the SP, along with the remaining gain margin with respect to a perfect allocation matching the actual traffic. (b) Example of how overbooking improves the MNO NSaaS operation in a simple case with two slices. We show the real traffic and requested resources of the first slice (solid brown lines) and those of both slices (solid black lines). As per plot (a), the MNO can perform a faster and more accurate anticipatory allocation of resources to both slices (dotted line). This leads to resource savings (blue areas) with respect to allocating all the SP requests when those are below the capacity limit $C_r^{(l)}$. More importantly, it allows accepting both slices even if their aggregated requests exceed the MNO available capacity (green areas). Overbooking errors may however lead to SLA violations, when the actual demand of the accepted slices cannot be served (red area).

Overbooking network slices. Cloud-native technologies allow the MNO to orchestrate resources and VNFs at much faster timescales than those of NSaaS brokering. Thus, while the network slice tenant requests resources for its peak consumption over long reservation periods, the actual allocation and re-configuration of slice-dedicated resources can be performed at a finer time granularity. In addition, the operator has in-depth visibility of the actual infrastructure utilization, and can hence allocate resources based on the real resource occupancy generated by the service demands, beyond the capacity requests issued by the vertical tenants. These technical advantages, illustrated in Fig. 1a, open the door to large slice multiplexing gains, letting the MNO make a more efficient use of its resources [20] and ultimately increasing its profit.

Specifically, reducing the amount of capacity needed to serve each slice can free up space for accommodating more requests, as exemplified in Fig. 1b. In other words, the MNO can sell more capacity than it has deployed, considering that vertical customers will not use all the capacity they requested all the time. The strategy maps to *overbooking*, a well-known revenue management approach used to maximize profit in scenarios where limited resources must be reserved based on stochastic requests [30]. In the case of overbooking for NSaaS, errors in admitting excess slices come at the cost of monetary fees for violating the SLA with one or multiple tenants during some fraction of time, as also shown in Fig. 1b. Overbooking has been recently considered as a way to increase NSaaS revenues for the MNO, with promising results [26–28]. Yet, as later detailed in Sec. 2, prior studies are few, have technical limitations, and none has demonstrated practical solutions with real-world traffic demands of vertical customers collected in actual operational networks. The latter aspect is especially critical now that cloud-native networks are bringing slicing closer to deployment, and there is a clear need to understand how overbooking would perform in production systems.

Contributions. In this paper, we make the following contributions towards an efficient realization of NSaaS overbooking.

- We design and implement kaNSaaS, a novel complete solution for overbooking-aware NSaaS, which solves the joint problem of slice admission control and anticipatory resource allocation by combining (*i*) deep-learning slice demand forecasting and (*ii*) optimization-based decision-making. Our formulation is modular and can accommodate any SLA expressions and Operating Expenses (OPEX) cost definitions.
- We provide a first assessment of overbooking gains in presence of real-world demands generated by multiple service providers, as measured in a metropolitan-scale production network. We investigate advantages for the MNO in terms of net profit along diverse dimensions that include the resource orchestration flexibility, the cost of allocated resources to slices, or the overdimensioning strategy of the operator. Ours is the very first evaluation of overbooking for NSaaS in presence of realistic slice requests, which unveils the actual advantages that the technology may bring in practical settings.
- We prove that kaNSaaS increases the MNO profits by 300% with respect to legacy NSaaS management strategies, and above 20% over state-of-the-art slice overbooking.
- We show that results stay consistent under original synthetic slice demands that we generate to mimic the measurement data. While we cannot disclose the latter due to confidentiality agreements, we release[1] the synthetic traffic together with our implementation of kaNSaaS, so as to foster the reproducibility of our study and support further investigations.

Overall, our work contributes to advance the state of the art in NSaaS management, and sheds light on the actual gains that overbooking can bring to the MNO in production settings.

[1]Code and data are available at https://github.com/nds-group/kansaas.

2 RELATED WORK

Most studies on network slicing have investigated the key management functionalities of admission control of slice requests [10, 18] and allocation of resources to individual slices [11, 35] in isolation. Previous works that jointly addressed the two tasks [4, 7, 8] have overlook the important trade-offs entailed by: (*i*) the added revenues of accepting slices that request capacity beyond that available, while not using it all the time; and, (*ii*) the potential cost of violating SLAs in the moments when the actual demand of all accepted slices exceeds the total capacity.

Overbooking specifically tackles the trade-off above. Its application to the communications field is very recent, with a focus on pricing and billing strategies [31] or resource trading [15] in network edge clouds. When considering the specific context of network slicing, overbooking must not be confused with the simpler problem of slice multiplexing. For instance, there exist data-driven analyses of the multiplexing efficiency of network slices [20] or works that propose optimized resource allocation to multiplexed network slices [6, 37]. However, plain multiplexing does not consider the additional problem dimension of reserving less resources than those requested by the SPs, which is the focus of overbooking. The literature considering overbooking as an approach to maximize NSaaS revenues for the operator is in fact very thin. Sexton *et al.* [28] derive analytical models of the performance of slicing with overbooking under perfect prediction, but does not present a practical solution to the problem. Saxena *et al.* [27] propose a model for network slice overbooking, which relies on a Long Short-Term Memory (LSTM) neural network for demand forecasting, and a Reinforcement Learning (RL) approach for admission control. Yet, the solution operates on inflexible SLA costs and does not tackle the resource allocation part of the problem. The current state of the art in NSaaS overbooking is represented by the work by Salvat *et al.* [26], who first introduced the concept and demonstrated its practical viability in a small-scale experimental platform [36]. They propose a solution to the joint admission control and resource reservation problem under overbooking: the approach is based on traffic prediction via multiplicative Holt-Winters exponential smoothing, combined with a stochastic yield management optimization problem for slice admission and resource allocation. We use this solution as a benchmark in our performance evaluation. It is also worth noting that ours is one of the very few works in the network slicing literature to build upon large-scale measurements of tenant demands, and the very first to do so in the context of overbooking for NSaaS. Indeed, the vast majority of the literature relies on synthetic data [12], which undermines the dependability of results. When real-world data is employed, it often describes aggregate traffic over all services and is thus not representative of actual tenant demands [13, 14, 34]. Evaluations of NSaaS management solutions with service-level measurements are rare, and, as mentioned above, do not consider overbooking [17]. Indeed, previous solutions for NSaaS overbooking have been tested with synthetic workloads only [26], or on traces of resource utilization in cloud datacenters that are hardly representative of mobile service demands [27]. By evaluating NSaaS overbooking solutions with production-level mobile traffic measurements, we offer an unprecedented view on the real-world performance of slicing.

3 SYSTEM MODEL

We consider a dynamic resource allocation scenario where an MNO running the mobile network infrastructure aims at maximizing the profit obtained from NSaaS. To this end, the MNO needs to take decisions on admission control of slice requests and allocation of resources[2] to active slices. The problem can be instantiated at any target network location where slicing is implemented, *e.g.*, from individual Remote Units (RUs) where spectrum can be sliced, all the way to Core Network (CN) datacenters where compute and memory resources are reserved to run slice-tailored VNFs. Let *layer l* denote the layer whose nodes serve, on average, the aggregated traffic of l RUs; then for each node at layer l, we model the system as follows.

3.1 NSaaS operation

The MNO serves a set of N Service Providers (SPs), which we denote by $\mathcal{S} \triangleq \{s_n\}_{n\in\mathcal{N}}$, $|\mathcal{S}| = N$, where we define $\mathcal{N} \triangleq \{1,\ldots,N\}$ for any natural number N. The MNO monitors the demand generated by each SP within a short interval (*e.g.*, per minute in Fig. 1a). We denote the traffic generated by SP s at the monitoring interval k as $\ell_s[k]$ (see Fig. 1a). At any time, a service provider can request a network slice associated to an SLA with the following parameters.

- T_{slice}: time during which the slice must be active and the related SLA satisfied, *e.g.*, the whole span of Fig. 1a.
- T_{SLA}: duration of an *SLA block* of an SP request, *i.e.*, time interval during which a constant capacity is requested by the SP.
- $\bar{\ell}_s^{(\text{P})}(t)$: Requested capacity by an SP s for the t-th SLA block, *e.g.*, the ordinates of the 4 constant segments requested in Fig. 1a.
- $M_s\bar{\ell}_s^{(\text{P})}(t)$: Price that the SP s is offering to pay for the t-th SLA block. We model prices as linearly proportional to the capacity by a factor M_s (in \$/bps), but other definitions are possible.

The MNO decides whether to accept the slice requests,[3] and what resources to allocate to them if accepted. These decision are based on the request attributes above, as well as on the next parameters.

- T_{hor}: time horizon for the overall system optimization, *e.g.*, the multiple repetitions of the span of Fig. 1a.
- T_{dec}: time interval between admission decisions. The operation is batched, such that the MNO considers all requests arrived over the last T_{dec}, decides which slices are accepted, continued or dismissed, and estimates the resources that shall be dynamically reserved to each slice for its duration.
- T_{RA}: duration of one *Resource Allocation (RA) block*, *i.e.*, the interval during which the capacity allocated by the MNO to a slice remains constant, which is typically bounded by the technology available at the target network domain. We also define as $n_{\text{RA}} \triangleq \frac{T_{\text{SLA}}}{T_{\text{RA}}}$ the number of RA blocks per SLA block, *i.e.*, the amount of re-allocation opportunities for the MNO while the requested capacity $\bar{\ell}_s^{(\text{P})}$ stays fixed. As an example, in Fig. 1a we have $T_{\text{RA}} = 30$ and $T_{\text{SLA}} = 120$ minutes, yielding $n_{\text{RA}} = 4$.
- $C_r^{(l)}$: capacity available at the target node r of layer l, which sets the boundary to the total traffic demand that can be served at any time instant, as shown in Fig. 1b.
- $\ell_s^{(\text{O})}(t, n)$: Capacity actually reserved by the MNO for slice s.

[2]We use the terms *resources* and *capacity* interchangeably in the following, as the amount of dedicated resources directly determines the capacity that can be provisioned.
[3]We refer to the slice that serves the traffic of SP s as slice s, $\forall s \in \mathcal{S}$.

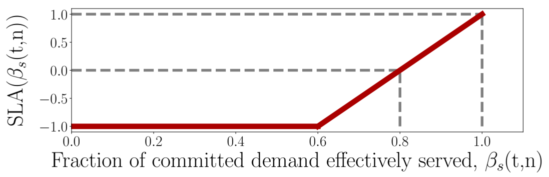


Figure 2: SLA function adopted for our experiments.

- c_{OPEX}: MNO's OPEX (in \$/bps) of reserving one unit of capacity to a slice during a whole RA block, due, *e.g.*, to the energy or monetary cost of running dedicated VNF containers or CPU cores.
- ρ_{op}: operational profit ratio between the revenue from the SP and the OPEX of reserved resources, *i.e.*, $M_s/(n_{\text{RA}} c_{\text{OPEX}})$.[4]

Based on the above, we define a hierarchical time indexing. The index t refers to the SLA block index, and the notation (t, n) refers to the n-th RA block of the t-th SLA block; *e.g.*, in Fig. 1a, T_{RA} is represented for the $(3, 2)$ RA block. Moreover, we consider $\ell_s(t, n) \triangleq \max_{k \in [T_{\text{RA}}]} \ell_s[k]$ as the real demand in the (t, n) RA block. To avoid cluttering notation, we focus hereinafter on a given node r in layer l, and omit the dependence on r and l. Also, we let $T_{\text{slice}} = T_{\text{SLA}} = T_{\text{dec}}$, *i.e.*, a slice has the same duration (T_{slice}) of an AC time slot of the MNO (T_{dec}), and the capacity requested by SP s is constant for the whole duration of the slice (T_{SLA}), typically in the order of hours. In this way, a certain scenario can be succinctly referred to as $\mathcal{E} = \{T_{\text{hor}}, T_{\text{SLA}}, T_{\text{RA}}, C\}$.

3.2 SLA function

The SLA defines the monetary compensation or penalty associated to an accepted slice. It is modeled as a function that depends on the requested capacity (for which the SP pays a fee as set out in Sec. 3.1) and the actual traffic served by the MNO. Specifically, at the n-th RA block of the t-th SLA block, the MNO commits to serve slice s with a capacity $\ell_s^c(t, n) \triangleq \min(\ell_s(t, n),\ \bar{\ell}_s^{(\text{P})}(t))$: if the slice traffic is below the level requested in the SLA, the operator only needs to serve such traffic and not the requested capacity in the SLA. The actual served traffic is $\ell_s^{\text{eff}}(t, n) = \min(\ell_s(t, n),\ \ell_s^{(\text{O})}(t, n))$, *i.e.*, an over-allocation of resources does not bring any benefit to the MNO.

The full monetary compensation set out by the SLA as per Sec. 3.1 is paid by the SP s to the MNO when $\ell_s^{\text{eff}}(t, n) \geq \ell_s^c(t, n)$. If instead the served demand is below that the MNO committed to accommodate, the SLA defines a reduction of the MNO's revenues. We model the compensation as a generic function $\text{SLA}_s(\beta_s(t, n))$, where

$$\beta_s(t, n) \triangleq \min\left(1,\ \frac{\ell_s^{\text{eff}}(t,n)}{\ell_s^c(t,n)}\right) \tag{1}$$

is the fraction of committed demand that is effectively served by the MNO, capped at 1 in the case where MNO unnecessarily serves more traffic than committed.

While our definition of SLA above is general, and our solution can accommodate other expressions, for the experiments carried out in this paper we leverage the SLA portrayed in Fig. 2. The rationale is that the agreed compensation drops linearly as the MNO fails to deliver the required capacity down to 80% of what it committed to. Below such a threshold, the MNO must pay a monetary fee (*i.e.*, a negative gain in the plot) to the SP, which grows up to the original compensation when only 60% of the slice traffic is served.

[4] $n_{\text{RA}} = T_{\text{SLA}}/T_{\text{RA}}$ transforms the cost per RA block into cost per SLA block.

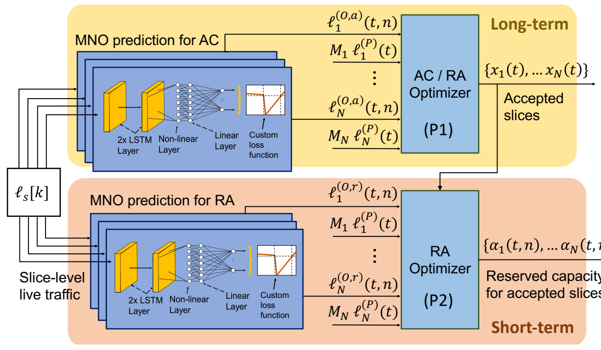


Figure 3: Overall architecture of kaNSaaS, with long-term and short-term prediction-enabled AC and RA components.

3.3 MNO profit

The MNO's objective is to maximize the total net profit over the operating horizon T_{hor}. The profit is the result of the overall revenue obtained from the slice brokering minus the costs incurred by the operator, *i.e.*, SLA violations that may induce a penalty as discussed in Sec. 3.2 and operating expenses derived from allocating the network resources. Formally:

- **Revenues** correspond to the compensations from meeting SLAs with SPs, *i.e.*, $M_s \bar{\ell}_s^{(\text{P})}(t)$ for t-th SLA block of duration T_{SLA}, possibly decreased according to $\text{SLA}_s(\beta_s(t, n))$.
- **SLA violation costs** are incurred when accepted slices are poorly served. This cost is embedded in the SLA definition when it takes values < 0, hence a single expression $M_s \bar{\ell}_s^{(\text{P})}(t) \cdot \text{SLA}_s(\beta_s(t, n))$ captures both revenues and SLA violation costs.
- **OPEX costs** are proportional to the capacity $\ell_s^{(\text{O})}(t, n)$ actually reserved by the MNO, by the c_{OPEX} factor.
- **Profits**, denoted by $p_{\mathcal{E}}$, combine the previous as follows

$$p_{\mathcal{E}} = \sum_{t \in \mathcal{T}} \sum_{s \in \mathcal{S}} x_s(t) \sum_{n=1}^{n_{\text{RA}}} \left(\frac{M_s \bar{\ell}_s^{(\text{P})}(t)}{n_{\text{RA}}} \text{SLA}_s(\beta_s(t, n)) - c_{\text{OPEX}} \ell_s^{(\text{O})}(t, n) \right). \tag{2}$$

In (2), the binary variable $x_s(t) \in \{0, 1\}$ is set to 1 if slice s is accepted in SLA block t. In case the slice is admitted, the profit is the difference between the revenue (or SLA violation cost) and the OPEX cost across all n_{RA} RA blocks composing the SLA block. The total profit is then computed over all slice requests $\mathcal{S}$ and through the whole temporal set of SLA blocks $\mathcal{T} \triangleq \{t \mid t \in [T_{\text{hor}}/T_{\text{SLA}}]\}$.

4 KANSAAS

In order to maximize the MNO profit in (2), we propose a novel solution for overbooking-aware NSaaS, or kaNSaaS. Our approach addresses the joint problem of (*i*) *Admission Control* (AC), *i.e.*, deciding which slice requests to accept, and (*ii*) *Resource Allocation* (RA), *i.e.*, determining how many resources to allocate to each of the accepted slices. It is important to note that both parts of the problem are inherently *anticipatory* in nature, but operate *at different timescales*. In AC, the MNO must admit slices at the start of each SLA block so as to ensure that their demand is accommodated during the future T_{SLA} time interval. In RA, the MNO has to reserve resources at the beginning of each RA block in a way to best serve the traffic through the following T_{RA} interval.

The overall architecture of kaNSaaS is illustrated in Fig. 3. Our design abides by the observations above, and hinges upon tailored forecasting models that support decision-making at two different timescales, *i.e.*, long-term SLA blocks and short-term RA blocks, as highlighted in the figure and detailed next.

- First, at every T_{SLA} time slots, the MNO computes slice demand predictions that inform a tentative resource allocation for the next T_{SLA} time interval. In turn, this provisional allocation is used to guide AC and decide which slices are accepted. Formally, the long-timescale component sets the binary variables $x_s(t) \in \{0,1\}$ that indicate whether a service s is accepted at SLA block t.
- Then, at each T_{RA} time slot n within the SLA block t, the operator performs the actual allocation of resources for the admitted slices for the next RA block, leveraging a more accurate forecast that is limited to the shorter T_{RA} interval. Formally, the short-timescale component defines $\alpha_s(t,n)$, *i.e.*, the percentage of the (predicted) traffic capacity that is to be reserved by the MNO to slice s. Clearly, AC actions at the longer timescale constrain and drive RA decisions at the shorter timescale, as shown by the arrow connecting the long- and short-term operations in Fig. 3.

The design above lets kaNSaaS take advantage of traffic forecasting to spot gaps between the capacity $\bar{\ell}_s^{(\text{P})}(t)$ requested by the SPs at each SLA block t and the future demand generated by the slices at every RA block (t,n). Then, it uses suitable optimizers to maximize the MNO profit in (2) from the identified gaps, via overbooking.

From a technical viewpoint, kaNSaaS combines apt machine learning and optimization tools to implement each component. Specifically, we adopt data-driven approaches to implement the prediction components at both short and long timescales, since Deep Learning (DL) models have been largely proven to outperform statistical models in time series forecasting tasks [19]. The predicted demands are then fed to dedicated optimizers that take rapid, effective decisions on AC and RA based on explainable logic.

Next, we detail the structure and operation of the two components, focusing on the long-term first and on the short-term after.

4.1 Long-term admission control

The AC operation takes place at the beginning of each SLA block, and aims at selecting slices so as to maximize the MNO profit, through overbooking based on long-timescale forecasts. The traffic prediction and decision-making parts are implemented as follows.

Traffic prediction. The MNO forecasts the expected traffic volume for each one of the RA blocks belonging to the next SLA block. Consequently, this Long-Term Predictor (LTP) outputs $T_{\text{SLA}}/T_{\text{RA}} = n_{\text{RA}}$ traffic values for the next T_{SLA} time slots. Our implementation uses a separate LTP for each requested slice s, which (*i*) best adapts to the diverse temporal dynamics of the heterogeneous mobile services associated with each slice, and (*ii*) allows for a modular design where predictors for SPs entering or leaving the slice brokering process can be dynamically added or removed.

This long-term prediction, denoted as $\bar{\ell}^{(\text{O,a})}(t,n)$, makes use of the state-of-the-art TES-RNN model for traffic forecasting [16]. This model combines statistical modeling and machine-learning tools, via a Recurrent Neural Networks (RNN) whose inputs are first passed through a Thresholded Exponential Smoothing (TES) [33] block. The key aspect is that the TES coefficients are simultaneously optimized with the RNN weights through a unified gradient descent [16]. The original TES-RNN is limited to output a single-value forecast, which is not suitable for our problem. We thus extend TES-RNN to support a multidimensional output (*i.e.*, a set of n_{RA} values), as well as to handle different time scales of input (*i.e.*, the monitoring samples $\ell_s[k]$) and output (*i.e.*, the RA block interval). Our TES-RNN implementation takes as input traffic samples for the last 8 hours for each service, and hinges on a deep neural network architecture with 2 LSTM hidden layers with dilations (1,3) and (6,12), both having a state size of 50, followed by a nonlinear layer with 50×50 state size and a linear adapter to the output size.

We train our enhanced TES-RNN model offline, with the α-OMC loss parametrized with $\gamma = 0.75$ [2]. This asymmetric loss avoids underestimations that may lead to SLA violations, while trying to minimize overprovisioning that increases OPEX costs.

Admission control. The predicted values $\bar{\ell}^{(\text{O,a})}(t,n)$ are fed together with the compensation for each slice s to an admission control optimizer. Let $\mathcal{V}_\alpha \triangleq \{\alpha_s(t,n) | n \in \{1,\dots,n_{\text{RA}}\}, s \in \mathcal{S},\ t \in \mathcal{T}\}$ and $\mathcal{V}_x \triangleq \{x_s(t) | s \in \mathcal{S},\ t \in \mathcal{T}\}$. The optimization problem is

$$\max_{\mathcal{V}_\alpha,\mathcal{V}_x} \sum_{t\in\mathcal{T}}\sum_{s\in\mathcal{S}} x_s(t) \sum\nolimits_{n=1}^{n_{\text{RA}}} \Big(\frac{M_s \bar{\ell}_s^{(\text{P})}(t)}{n_{\text{RA}}} \text{SLA}_s(\alpha_s(t,n)) - c_{\text{OPEX}} \alpha_s(t,n) \ell^{(\text{O,a})}(t,n) \Big) \quad \text{(P1)}$$

$$\text{s.t.} \sum\nolimits_{s\in\mathcal{S}} x_s(t)\alpha_s(t,n)\bar{\ell}_s^{(\text{O,a})}(t,n) \le C \quad \forall n,t \quad (3)$$

$$0 \le \alpha_s(t,n) \le 1 \qquad \forall \alpha_s(t,n) \in \mathcal{V}_\alpha \quad (4)$$

$$x_s(t) \in \{0,1\} \qquad \forall x_s(t) \in \mathcal{V}_x. \quad (5)$$

Problem (P1) maximizes the operational profit of the MNO defined in (2), *i.e.*, revenue minus costs of SLA violations and OPEX, subject to the available network capacity C and on the basis of the predicted demands $\bar{\ell}_s^{(\text{O,a})}(t,n)$ of each slice s through the next SLA block. It does so by identifying the admitted slices for which $x_s(t) = 1$, via a tentative allocation of resources $\alpha_s(t,n)$ in each RA block n of the future SLA block t. We remark that the MNO does not directly apply the resource allocation $\alpha_s(t,n)$ obtained from (P1); instead, it triggers the short-term stage to fine tune the resource allocation, as described in Sec. 4.2 hereafter. Problem (P1) is NP-hard and can be modeled as a knapsack problem, as we formally prove in Appendix A. However, the number of variables (*i.e.*, slices) that would be handled in NSaaS do not grow exponentially, and efficient solvers exist for knapsack problems [22].

4.2 Short-term resource allocation

Once the admission control for the next n_{RA} RA blocks is decided in (P1), the MNO performs the actual resource reservation at the start of each RA block. The rationale is that this requires a forecast over a shorter future horizon, which is inherently more accurate and allows higher savings on OPEX for the MNO. The implementation of such short-term operation, depicted in the bottom half of Figure 3, has a structure similar to the long-term decision component.

Traffic prediction. As anticipated, the MNO only forecasts the expected traffic volume for the next RA block. To this end, we use one Short-Term Predictor (STP) for each admitted slice s. The STP is based on the exact same hybrid model as the LTP, although the STP only outputs a single value. The predicted values $\bar{\ell}^{(\text{O,r})}(t,n)$ serve as input for the RA shot-term optimization presented next.

Resource allocation. At every T_{RA}, *i.e.*, n_{RA} times per SLA block, kaNSaaS solves an optimization problem to determine the exact resources $\alpha_s(t,n)$ reserved for each slice for the following RA block. Formally, for a given RA block (t,n), the optimization is

$$\max_{\mathcal{S}^{(a)}} \sum_{s\in\mathcal{S}^{(a)}} \Big(\frac{M_s \bar{\ell}_s^{(\mathrm{P})}(t)}{n_{\mathrm{RA}}} \, \mathrm{SLA}_s(\alpha_s(t,n)) - c_{\mathrm{OPEX}} \alpha_s(t,n) \bar{\ell}_s^{(\mathrm{O,r})}(t,n) \Big) \tag{P2}$$

$$\text{s.t.} \sum_{s\in\mathcal{S}^{(a)}} \alpha_s(t,n) \bar{\ell}_s^{(\mathrm{O,r})}(t,n) \le C \tag{6}$$

$$0 \le \alpha_s(t,n) \le 1 \qquad \forall s \in \mathcal{S}^{(a)}. \tag{7}$$

Problem (P2) aims at maximizing the contribute of the current RA block to the profit of the MNO, hence has a similar expression to that of (P1). The main differences are that (*i*) there is no admission control decision, (*ii*) it only considers the services in the set accepted in the last (P1) problem, denoted as $\mathcal{S}^{(a)} \subseteq \mathcal{S}$, and (*iii*) it leverages the STP forecast $\bar{\ell}_s^{(\mathrm{O,r})}(t,n)$. This problem is considerably simpler than the MIP in (P1), as it does not contains discrete variables, and its complexity depends solely on the expression of the SLA function. Thus, (P2) is a linear programming (LP) problem with the SLA in Sec. 3.2, yet it could turn non-linear under a different SLA function.

4.3 MNO profit from AC/RA decisions

The AC/RA decisions taken by the components described in Sec. 4.1 and Sec. 4.2 determine the MNO profit, as follows. The values $\alpha_s(t,n)$ denote the percentages of the *predicted* future traffic that is reserved for slice s: thus, the actual served traffic at RA block (n,t) is $\ell_s^{\mathrm{eff}}(t,n) = \min(\ell_s(t,n),\ \alpha_s(t,n)\ell_s^{(\mathrm{O,r})}(t,n))$. From this expression, we obtain the value of $\beta_s(t,n)$ in (1), *i.e.*, the percentage of agreed traffic volume that is effectively served by the MNO. The final profit uses the $\beta_s(t,n)$ above and is given by

$$p_{\mathcal{E}} = \sum_{t\in\mathcal{T}} \sum_{s\in\mathcal{S}} x_s(t) \sum_{n=1}^{n_{\mathrm{RA}}} \Big(\frac{M_s \bar{\ell}_s^{(\mathrm{P})}(t)}{n_{\mathrm{RA}}} \, \mathrm{SLA}_s(\beta_s(t,n)) - c_{\mathrm{OPEX}} \alpha_s(t,n) \ell_s^{(\mathrm{O,r})}(t,n) \Big). \tag{8}$$

Discrepancies between (8) and the objective functions of (P1) and (P2) are possible due to slice traffic prediction errors, and are part of the complexity of an overbooking-based NSaaS.

5 PERFORMANCE EVALUATION

We evaluate kaNSaaS using both real and synthetic datasets of slice demands, presented in Sec. 5.1, and we compare its performance to benchmark NSaaS management approaches, described in Sec. 5.2. The results of our evaluation are presented in Sec. 5.3.

5.1 Datasets

A contribution of our study is a first assessment of the potential gain of overbooking in NSaaS in presence of real-world demands generated by a variety of service providers, based on measurements of the traffic of individual mobile applications in a metropolitan-scale network of a major European MNO. As this dataset is protected by a Non-Disclosure Agreement (NDA) with the MNO, we also generate a synthetic dataset that mimics the main properties of the real-world service-level demands, and leads to comparable results in our evaluations. We disclose this second dataset to allow reproducibility of our results and foster further research in NSaaS.

5.1.1 Measurement dataset. The real-world mobile demand dataset captures all traffic generated by several millions of users in a large metropolitan area during 8 consecutive weeks. The data consist of the traffic loads served by each of the hundreds of base station covering the target geographical region, at a time granularity of 1 minute. The traffic is reported separately for 20 different mobile services, which include the most popular smartphone applications, such as YouTube, Instagram, Twitter or various Google services.

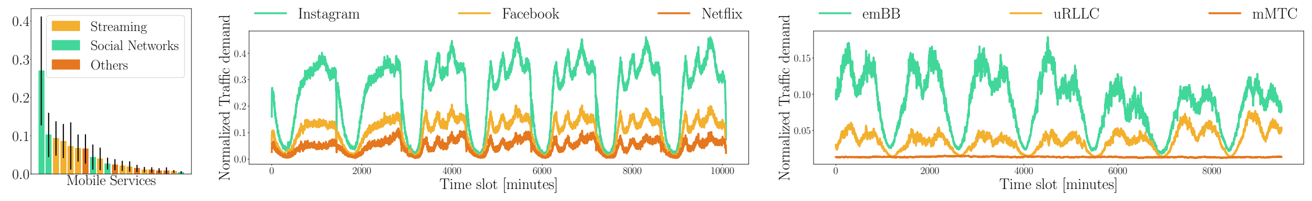
Such data was collected and aggregated by the MNO in its production infrastructure, using passive measurement probes deployed in the Evolved Packet Core (EPC). The probes run commercial and proprietary traffic classifiers to identify the service associated to each IP flow. The measurement dataset allows defining realistic SP demands, by assigning one slice to each service. As shown in Fig. 4a-b, the resulting slices have heterogeneous traffic volumes and time dynamics, which opens opportunities for multiplexing.

5.1.2 Synthetic dataset. We emulate demands for the three main network slicing categories for 5G, namely enhanced Mobile BroadBand (eMBB), Ultra-High Reliability & Low Latency (uRLLC) and Massive Machine-type communications (MMTC). For the eMBB and uRLLC classes, we created a weekly pattern of 5 work days and 2 weekend days, with each day containing a sinusoidal dynamic that mimics the well-known circadian rhythm of mobile traffic. Based on our measurement data, we also model two daily traffic peaks, higher in the morning and lower in the afternoon. Instead, MMTC slices are characterized by a steady demand over time, to reflect to deterministic behavior of many applications in that class.

Individual slices are told apart by their generated traffic volumes and weekly patterns. We use peak traffic values of 60 Mbps for eMBB, 7 Mbps for mMTC, and 35 Mbps for uRLLC, and introduce diversity across slices of the same category, by scaling all values of a slice by a random factor between 0.7 and 1.4. Also, we reflect the temporal variability observed in the measurement data by randomly shifting each slice demand by up to 1.5 hours. We then imitate the inherent randomness of mobile device behaviors, by adding a coloured noise with a standard deviation equal to 35% of the mean throughput. This power-law noise has a power spectral density per unit of bandwidth proportional to $\frac{1}{f^{\phi}}$ [23], where $\phi = 0$ implies white noise, $\phi = 1$ represents pink noise, and Brownian noise corresponds to $\phi = 2$. We select $\phi = 1.08$ to strike a balance between trend and noise. Ultimately, our synthetic dataset consists of 90,720 data points at 1 minute granularity for each emulated slice, which is consistent with the real-world data, as exemplified in Fig. 4c.

5.1.3 Slice requests. We generate the SP requests from the datasets of service demands above, by considering that each SP forecasts the expected future traffic generated by its service, and asks for sufficient slice resources to process it. For fairness, we consider that the SPs make use of the state-of-the-art model from [16], which is suitably configured to predict a single value corresponding to the maximum traffic demand for the next SLA block of duration T_{SLA}.

We also assume that SPs rely on more conservative predictions than the MNO, since they are committed to ensure proper quality of experience to their users. To this end, we parameterize the α-OMC loss with a higher $\gamma = 1.5$ parameter that induces a higher safety margin against underprovisioning in SP forecasts [2].



(a) Ranked SP traffic **(b) Sample measurement slice traffic** **(c) Sample synthetic slice traffic**

Figure 4: (a) Sorted normalized traffic per SP; (b)-(c) Temporal demands of 3 slices in the measurement and synthetic datasets.

5.2 Benchmarks

We consider two baselines that allow contextualizing the performance of kaNSaaS, and one state-of-the-art benchmark, as follows. **Legacy NSaaS.** This is a traditional network slicing management strategy where the MNO allocates exactly what the service provider requests, and does not perform any slice overbooking. This implies that $\bar{\ell}_s^{(O)}(t,n) = \bar{\ell}_s^{(P)}(t)$. We denote by $\mathcal{E}^{(P)}$ the `Legacy NSaaS` counterpart of a given overbooking scenario $\mathcal{E}$ (as defined at the end of Sec. 3.1), and compute the net profit gain of overbooking as

$$\bar{G}_p(\mathcal{E}) \triangleq \frac{p_{\mathcal{E}} - p_{\mathcal{E}^{(P)}}}{p_{\mathcal{E}^{(P)}}}. \tag{9}$$

Oracle NSaaS. This is an unfeasible but optimal slice AC/RA management where $\bar{\ell}_s^{(O)}(t,n) = \ell_s(t,n)$ and $T_{RA} = 1$. We denote by $\mathcal{E}^\star$ the `oracle` counterpart of a given scenario $\mathcal{E}$, and calculate the similarity of a practical solution with the optimal as the ratio

$$\bar{D}_p(\mathcal{E}) \triangleq \frac{p_{\mathcal{E}}}{p_{\mathcal{E}^\star}}. \tag{10}$$

CoNEXT. This is the state-of-the-art solution for NSaaS overbooking, originally introduced by Salvat *et al.* [26] and denoted by CoNEXT in the following. As partially anticipated in Sec. 2, CoNEXT relies on SP demand forecasts returned by the multiplicative version of the three-smoothing Holt-winters (HW) algorithm with seasonality. Admission decisions of slices are taken by solving a stochastic yield management optimization problem, which is however based only on a short-term forecast over the future T_{RA} interval. As slice AC is enforced through a longer interval of T_{SLA} time slots, CoNEXT then updates the forecast and compute the resource re-allocation at every subsequent RA block of duration T_{RA}. In other words, CoNEXT lacks the two-timescale operation of kaNSaaS, which forces it to take long-term SLA AC decisions based on short-term demand forecasts. In addition to such a fundamental design gap, the solution differs from kaNSaaS in the implementation of both the prediction and decision modules.

5.3 Evaluation

We assess the performance of kaNSaaS, `Legacy NSaaS` and CoNEXT in presence of both measurement and synthetic demands. All results are expressed in terms of the profit similarity from the performance of the `oracle` approach $\bar{D}_p$, as defined in (10). Unless stated otherwise, we use the following default settings throughout all experiments: $T_{RA} = 30$ min, $T_{SLA} = 120$ min, $c_{OPEX} = 0.9\frac{M_s}{n_{RA}}$ (*i.e.*, a profit margin $\rho_{op} = 11\%$), and the network capacity is set to be equal to the maximum aggregated traffic over services over the whole dataset, or $C_r^{(l)} = \max_{t\in\mathcal{T}} \sum_{s\in\mathcal{S}} \ell_s(t)$.

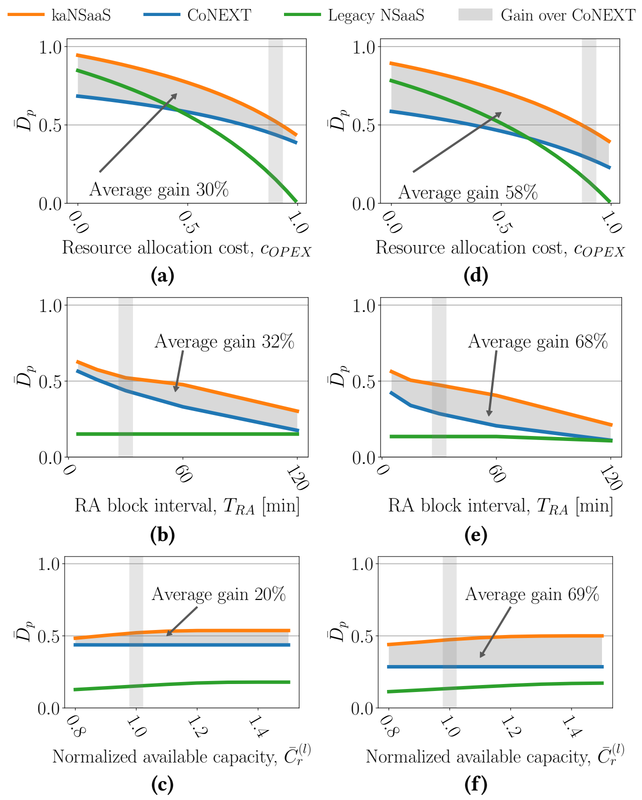


Figure 5: Performance evaluation as function of (a,d) the profit margin, (b,e) the orchestration interval T_{RA}, and (c,f) network capacity. Results refer to (left) real-world demands and (right) synthetic traffic. We compare kaNSaaS, CoNEXT [26], and `Legacy NSaaS`. Performance is expressed as a similarity $\bar{D}_p$ with the unfeasible `oracle`. Gray vertical lines represent the default settings listed in Sec. 5.3.

5.3.1 Overall overbooking gain with kaNSaaS. The main results of our performance evaluation are summarized in Fig. 5. The figure reports the profit of each tested solution (as different curves), under real-world (left column) and synthetic (right column) SP demand datasets. As anticipated, all values are indicated as similarity to the optimum `oracle` performance, so as to favor interpretability.

The vertical greyed region in each plot highlights the default settings, for which the following key observations are in order.

- The gain of kaNSaaS with respect to `Legacy NSaaS` is very large at around 300%. This implies that *overbooking can grow fourfold the economic profit for the MNO* with respect to a case where the operator just abides by the requests of the SPs in a typical case.

- The gain of kaNSaaS with respect to CoNEXT under the default settings is of 20% with real demands, and of 40% with synthetic traffic. In other words, thanks to a more complete two-timescale design and more effective implementations of forecasting and AC/RA decision models, kaNSaaS *significantly grows the net profit for the MNO over the best available solution for NSaaS overbooking.*
- The similarity to the performance of the `oracle` is at 0.5 denoting that the default settings scenario does not allow kaNSaaS to take full advantage of the potential of NSaaS overbooking.

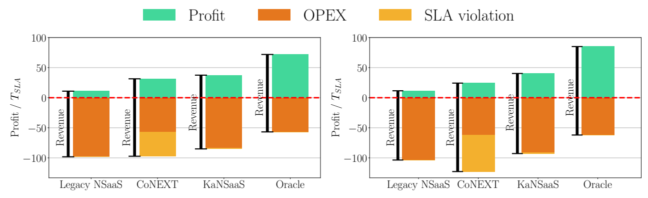
These considerations shed light on the actual monetary advantage that overbooking can bring to an MNO in a dependable slice demand scenario, and unveil in particular how such an advantage can be surprisingly large. We next investigate how the system settings affect the overbooking performance.

5.3.2 Impact of profit margin. A chief parameter of interest for the MNO is the profit margin at which it can operate the service. This element directly determines the feasibility and interest of NSaaS use cases. We analyze the impact of the profit margin by varying the ratio $\rho_{op} = \frac{M_s}{c_{\text{OPEX}} n_{\text{RA}}}$. In particular, we vary the coefficient c_{OPEX} from 0 (*i.e.*, neglecting OPEX) to $0.99\frac{M_s}{n_{\text{RA}}}$ (beyond which NSaaS is not profitable for the MNO, as $\rho_{op} \rightarrow 1$). Fig. 5a shows how varying c_{OPEX} (as a coefficient of $\frac{M_s}{n_{\text{RA}}}$) impacts the profit in real-world demands, and Fig. 5d shows the same for synthetic traffic.

The two plots highlight the very significant role of resource operation costs in controlling the gain of overbooking strategies.

- On the one hand, the gain of kaNSaaS over `Legacy NSaaS` is maximum at profit margins around 10-20%, *i.e.*, for c_{OPEX} in the $[0.8, 0.9]$ range ($\times\frac{M_s}{n_{\text{RA}}}$). Here, overbooking grants dramatic manyfold boosts in the economic profit.
- On the other hand, the performance of kaNSaaS tends to get closer to the optimal `oracle` as c_{OPEX} decreases, with a similarity $\bar{D}_p$ closer to 1 as c_{OPEX} tends to 0. Indeed, as we will later show in Sec. 5.3.5, all the cost induced by our solution is imputable to overdimensioning, and reducing c_{OPEX} shrinks that cost.
- The state-of-the-art CoNEXT solution performs well under low profit margins, yet the quality of its AC/RA decisions tends to rapidly deteriorate as c_{OPEX} is reduced, up to the point where the profits it grants become lower than those of a no-overbooking `Legacy NSaaS` strategy. The reason is that, as shown in Sec. 5.3.5, CoNEXT aggressively overbooks resources, serving more traffic than all other approaches but also incurring in many SLA violations. The cost of such violations dominates in absence of significant OPEX costs, thus penalizing this solution. Our proposed kaNSaaS does not suffer from this problem, and stays a better choice than `Legacy NSaaS` and CoNEXT across all c_{OPEX}.

5.3.3 Impact of resource orchestration interval. We vary T_{RA} from 5 to 120 minutes, which is the same as the duration of the T_{SLA} requested by each SP and thus an upper bound to the RA block duration. Fig. 5b and Fig. 5e show how kaNSaaS increases its profit gain over CoNEXT as T_{RA} increases, *i.e.*, as the resource re-allocation decisions are spaced apart: the gain grows from 20% at $T_{\text{RA}} = 30$ minutes to 72% at $T_{\text{RA}} = 120$ minutes. Also, the profit gain of CoNEXT over `Legacy NSaaS` is only of 16% at $T_{\text{RA}} = 120$ minutes, while kaNSaaS still doubles the profit over the baseline NSaaS without overbooking. These gains demonstrate the importance of DL-based forecasting to predict traffic over longer time horizons.



(a) (b)

Figure 6: Revenue, profit, OPEX, and SLA violation costs of all considered NSaaS solutions under (a) measurement and (b) synthetic demands, and with default system settings.

5.3.4 Impact of network capacity. Let us denote the maximum sum traffic over the whole dataset by $L \triangleq \max_{t \in \mathcal{T}} \sum_{s \in \mathcal{S}} \ell_s(t)$: we then define the normalized available capacity as $\bar{C}_r^{(l)} = C_r^{(l)}/L$, and vary $\bar{C}_r^{(l)} \in [0.8, 1.5]$. The results are in Fig. 5c and Fig. 5f. For kaNSaaS, the performance with respect to the `oracle` approach is not strongly affected by the network capacity, as it is similar (at 48% and 54%) in the extremes case where the network is underdimensioned ($\bar{C}_r^{(l)} = 0.8$) or overdimensioned ($\bar{C}_r^{(l)} = 1.5$). The performance of CoNEXT is similarly not affected by the capacity of the network, with a steadily lower 43% performance with respect to `oracle`. The result of `Legacy NSaaS` is very far from `oracle`, with low similarity that varies from 17.4% for $\bar{C}_r^{(l)} = 0.8$ to 22% at $\bar{C}_r^{(l)} = 1.5$. Ultimately, these results prove how the overbooking gains are only marginally affected by the available capacity.

5.3.5 Profit deconstruction. In order to better understand how kaNSaaS outperforms the benchmarks, we break apart the economic gain and costs incurred by each NSaaS scheme, so as to reveal how the final net revenue is obtained. Fig. 6 shows the total revenues from accepted slices, the OPEX from resource allocation, the fees due to SLA violations, and the profit resulting from subtracting the latter two costs from the initial revenues. We first observe how `Legacy NSaaS` does not incur in any cost in terms of SLA violations, owing to a conservative policy of abiding by the requests of the SPs, which are in turn designed to avoid any service disruption. However, this result comes at a very high OPEX cost, since SP requests tend to be overdimensioned. Also, `Legacy NSaaS` yields the lowest total revenues, as its conservative approach leads to accepting a lower number of slices. On the other end of the spectrum, CoNEXT is the most aggressive approach, which accepts the highest number of slices and thus attained the highest total revenues. To do so, this NSaaS solution heavily employs overbooking, but it also pays a substantial penalty from SLA violations. Ultimately, SLA violation costs curb the MNO net profit. kaNSaaS achieves a better trade-off between slice overbooking (accepting more slices than `Legacy NSaaS`, thus increasing the total revenue) and SLA violation avoidance (paying a negligible cost for those, especially when compared with CoNEXT). By striking this balance, and even though the total revenues are lower and the OPEX is higher than those of CoNEXT, kaNSaaS creates a significantly higher total profit for the MNO. We also observe that our solution is the one that resembles the most the optimal `oracle` approach: indeed, the two solutions accepts roughly the same SP requests, which results in a very similar total revenue. The difference is then ascribed to the fact that `oracle` relies on a perfect prediction over instantaneous RA block intervals, which is not feasible in practice.

6 EXHAUSTIVE SYSTEM ANALYSIS

In this section, we carry out a complete analysis of the NSaaS overbooking performance across the additional system dimensions that we did not explore in Sec. 5. Specifically, we analyze how sensitive the performance is to the network layer at which the NSaaS management is performed, as well as how different levels of accuracy in the MNO traffic prediction affect the final net profit. In order to control the second aspect, we replace the actual LTP and STP models with a parametrizable overprovisioning factor a that multiplies the actual peak throughput in the following RA block. In other words, we employ an abstract predictor that achieves a tunable accuracy instead of practical forecasting solutions. Specifically, the default values for overprovisioning are 20% for SPs and 5% for the MNO, such that we have that $\bar{\ell}_s^{(\mathrm{P})}(t) = 1.2 \max_{n\in[n_{\mathrm{RA}}]} \ell_s(t,n)$ and $\bar{\ell}_s^{(\mathrm{O,a})}(t,n) = 1.05\ell_s(t,n)$. In the following, we only report results obtained with the real-world measurement demands, since those returned with synthetic data did not show significant differences and are omitted due to space limits. Also, all results are presented in terms of the gain of `kaNSaaS` over `Legacy NSaaS`, as per (9).

6.1 Network layer

The NSaaS management problem tackled by `kaNSaaS` can be instantiated at different locations of the network infrastructure. For instance, slicing could occur for spectrum at the level of individual Remote Units (RUs), for radio scheduling at the level of Distributed Units (DUs), for compute resources at the level of Centralized Units (CUs), or for transport or Cloud resources in the Core Network (CN). We model such different network layers as nodes that aggregate an increasing volume of traffic, or equivalently serve a growing number of clustered RUs, as we move from the radio access to the network core. Fig. 7a shows the total network capacity required to serve the whole demand in our measurement dataset at different network layers. The x-axis represents the average traffic demand per minute per cluster, normalized, in a logarithmic scale. As we move towards the RU layer (left-most value), the required capacity grows exponentially. Hence, although handling NSaaS at RUs provides the MNO with higher gains and flexibility, the required management and resources escalate at an unaffordable rate. We represent the gain over `Legacy NSaaS` in Fig. 7b and Fig. 7c, where the x-axis is the number of RU per cluster, and where $x = 843$ represents the CN layer. Fig. 7b shows the different performance when the orchestration interval T_{RA} varies from 1 to 120 min, whereas Fig. 7c shows the variation with respect to the level of overprovisioning, modeled by a coefficient a, such that $\bar{\ell}_s^{(\mathrm{O})}(t,n) = a \cdot \ell_s(t,n)$, and where a varies from 1 to 2. We only highlight the extreme and the default values for the sake of clarity.

The profit gain $\bar{G}_r$ exceeds 600%, *i.e.*, the operator can multiply its profit by seven times with overbooking. Importantly, the profit gain increases as we approach the RU layer, where traffic is more dynamic and there are more opportunities to multiplex demands.

6.2 Traffic prediction accuracy

Finally, we jointly analyze the impact of jointly varying T_{RA} (30 min by default) and the prediction accuracy level ($a = 1.05$ by default). The gain over `Legacy NSaaS` ($\bar{G}_p$) is show in the 3D plot of Fig. 7d.

We observe a non-monotonic behavior in the accuracy axis. This is due to the fact that, for $a \le 1.2$, a increases for the MNO but not for the SPs, which keeps the default value $a = 1.2$. Hence, reducing a improves the performance because we are assuming perfect predictions with no uncertainty. For $a > 1.2$, however, both MNO and SP share the same a (because SP will never be more aggressive provisioning than the MNO); then, increasing a is beneficial for the MNO due to the fastest resource allocation decision. We also observe that the peak gain is achieved at $T_{\mathrm{RA}} = a = 1$, because we do not have prediction errors (and thus SLA violations) in this controlled evaluation. Even when $T_{\mathrm{RA}} = 60$ and $a = 1.2$, overbooking increases the MNO profit by 75%.

7 CONCLUSIONS

We have proved that, under real-world service-level demands collected in a large-scale production network and with realistic operating cost margins, the net profit of the MNO from a practical overbooking-based NSaaS solution can be multiplied by a factor four. We also presented `kaNSaaS`, a practical solution that achieves the gains above over legacy NSaaS models, and largely outperforms the state-of-the-art scheme for NSaaS overbooking.

ACKNOWLEDGMENTS

We thank the Mobihoc anonymous reviewers for their helpful comments and guidance. The work of S. Alcalá-Marín was supported by the European Union's Horizon 2020 research and innovation programme under grant agreements no.101017109 “DAEMON". The work of A. Bazco-Nogueras was supported by the Regional Government of Madrid through the grant 2020-T2/TIC-20710 for Talent Attraction. The work of A. Banchs was also supported in part by the European Union's Horizon-JU-SNS-2022 Research and Innovation Programme under Grant Agreement No. 101095871 (TrialsNet) and by the Spanish Ministry of Economic Affairs and Digital Transformation and the European Union-NextGenerationEU through the UNICO 5G I+D 6G-CLARION project. The work of M.Fiore was supported by the European Union-NextGenerationEU through the UNICO 5G I+D project no.TSI-063000-2021-52 “AEON-ZERO”.

A NP-HARDNESS OF PROBLEM (P1)

The proof that (P1) is NP-hard follows by reduction from the Knapsack Problem (KP) [21], whose description we omit due to space constraint. Let us consider a particular case of our problem, in which $n_{\mathrm{RA}} = 1$ and the SLA function is given by $\mathrm{SLA}_s(\alpha_s(t)) = 1$ if $\alpha_s(t) \ge 1$ and $\mathrm{SLA}_s(\alpha_s(t)) = -1$ if $\alpha_s(t) < 1$, *i.e.*, any underprovisioning incurs a penalty equivalent to the possible revenue. Here, we have that, for any solution in which some slice s is accepted ($x_s(t) = 1$) with α_s different than 1 for that same slice, we can find another solution improving the objective value just by setting $x_s = 0$. Thus, our problem is equivalent to maximizing $\sum_{s\in\mathcal{S},\, t\in\mathcal{T}, n\in n_{\mathrm{RA}}} M_s(t)\bar{\ell}_s^{(\mathrm{P})}(t)x_s(t)$ over the set $\{x_s(t)\}_{s\in\mathcal{S}}$ subject to $x_s(t) \in \{0,1\}$ and $\sum_{s\in\mathcal{S}} \bar{\ell}_s^{(\mathrm{O,a})}(t,n)x_s(t) \le C_r^{(l)}$ for each $t \in \mathcal{T}$ and $n \in n_{\mathrm{RA}}$. Our problem is then equivalent to a KP with weights $\bar{\ell}_s^{(\mathrm{O,a})}(t,n)$, rewards $M_s(t)\bar{\ell}_s^{(\mathrm{P})}(t)$, and capacity $C_r^{(l)}$. Ultimately, the NP-hard KP is a particular case of our problem, and since this reduction can be built in polynomial time, our problem is NP-hard.

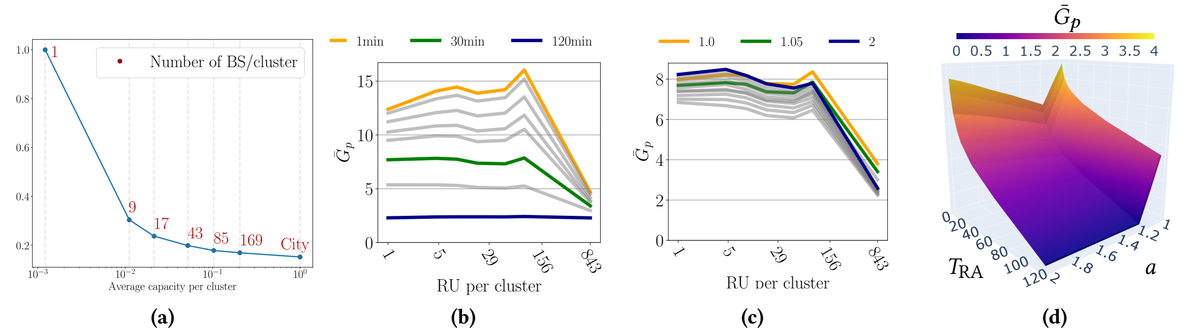


Figure 7: (a) Total capacity required to serve all demands at different network layers. (b,c,d) System performance as function of: (b) the network layer where NSaaS occurs, for different T_{RA} from 1 to 120 minutes; (c) the network layer where where NSaaS occurs, for different prediction accuracy a from 1 to 2; and, (d) the RA block interval T_{RA} and prediction accuracy a, at CN layer.

REFERENCES

[1] S. Arora and A. Ksentini. 2021. Dynamic Resource Allocation and Placement of Cloud Native Network Services. In *IEEE Int. Conf. Commun. (ICC)*. 1–6.

[2] D. Bega et al. 2019. DeepCog: Cognitive Network Management in Sliced 5G Networks with Deep Learning. In *Proc. of IEEE INFOCOM*. 1–9.

[3] D. Bega, M. Gramaglia, A. B., V. Sciancalepore, K. Samdanis, and X. Costa-Perez. 2017. Optimising 5G infrastructure markets: The business of network slicing. In *Proc. of IEEE INFOCOM*. 1–9.

[4] W. Ben-Ameur, L. Cano, and T. Chahed. 2021. A framework for joint admission control, resource allocation and pricing for network slicing in 5G. In *2021 IEEE Global Communications Conf. (GLOBECOM)*. 1–6.

[5] M. Burman and M. Gall. 2022. Ericsson and Red Hat empower service providers to build multi-vendor networks.

[6] P. Caballero, A. Banchs, G. de Veciana, and X. Costa-Pérez. 2017. Multi-Tenant Radio Access Network Slicing: Statistical Multiplexing of Spatial Loads. *IEEE/ACM Transactions on Networking* 25, 5 (2017), 3044–3058.

[7] P. Caballero, A. Banchs, G. de Veciana, X. Costa-Pérez, and A. Azcorra. 2018. Network Slicing for Guaranteed Rate Services: Admission Control and Resource Allocation Games. *IEEE Transactions on Wireless Communications* 17, 10 (2018), 6419–6432.

[8] S. Gholamipour, B. Akbari, N. Mokari, M. M. Tajiki, and E. A. Jorswieck. 2021. Online Admission Control and Resource Allocation in Network Slicing under Demand Uncertainties.

[9] D. Giannopoulos, P. Papaioannou, C. Tranoris, and S. Denazis. 2021. Monitoring as a Service over a 5G Network Slice. In *Joint European Conf. on Networks and Commun. & 6G Summit (EuCNC/6G Summit)*. 329–334.

[10] B. Han, V. Sciancalepore, D. Feng, X. Costa-Perez, and Hans D. Schotten. 2019. A Utility-Driven Multi-Queue Admission Control Solution for Network Slicing. In *Proc. of IEEE INFOCOM*. 55–63.

[11] Y. Hua, R. Li, Z. Zhao, X. Chen, and H. Zhang. 2020. GAN-Powered Deep Distributional Reinforcement Learning for Resource Management in Network Slicing. *IEEE J. Selected Areas in Communications* 38, 2 (2020), 334–349.

[12] J. A. Hurtado Sánchez, K. Casilimas, and O. M. Caicedo Rendon. 2022. Deep Reinforcement Learning for Resource Management on Network Slicing: A Survey. *Sensors* 22, 8 (2022).

[13] J. Koo, V. B. Mendiratta, M. R. Rahman, and A. Walid. 2019. Deep Reinforcement Learning for Network Slicing with Heterogeneous Resource Requirements and Time Varying Traffic Dynamics. In *Int. Conf. on Network and Service Management (CNSM)*. 1–5.

[14] Q. Liu, T. Han, and E. Moges. 2020. EdgeSlice: Slicing Wireless Edge Computing Network with Decentralized Deep Reinforcement Learning. In *IEEE Int. Conf. on Distributed Computing Systems (ICDCS)*. 234–244.

[15] M. Liwang, X. Wang, and R. Chen. 2022. Computing Resource Provisioning at the Edge: An Overbooking-Enabled Trading Paradigm. *IEEE Wireless Commun.* 29, 5 (2022), 68–76. https://doi.org/10.1109/MWC.104.2100380

[16] L. Lo Schiavo, M. Fiore, M. Gramaglia, A. Banchs, and X. Costa-Perez. 2022. Forecasting for Network Management with Joint Statistical Modelling and Machine Learning. (2022).

[17] Z. Luo, C. Wu, Z. Li, and W. Zhou. 2019. Scaling Geo-Distributed Network Function Chains: A Prediction and Learning Framework. *IEEE J. Selected Areas in Communications* 37, 8 (2019), 1838–1850.

[18] Q. T. Luu, S. Kerboeuf, and M. Kieffer. 2021. Uncertainty-Aware Resource Provisioning for Network Slicing. *IEEE Transactions on Network and Service Management* 18, 1 (2021), 79–93.

[19] S. Makridakis, E. Spiliotis, and V. Assimakopoulos. 2020. The M4 Competition: 100,000 time series and 61 forecasting methods. *Int. Journal of Forecasting* 36, 1 (2020), 54 – 74. https://doi.org/10.1016/j.ijforecast.2019.04.014

[20] C. Marquez, M. Gramaglia, M. Fiore, A. Banchs, and X. Costa-Pérez. 2019. Resource Sharing Efficiency in Network Slicing. *IEEE Transactions on Network and Service Management* 16, 3 (2019), 909–923.

[21] S. Martello. 1990. Knapsack Problems: Algorithms and Computer Implementations. *Wiley-Interscience series in discrete mathematics and optimiza tion* (1990).

[22] S. Martello and P. Toth. 1990. *Knapsack Problems: Algorithms and Computer Implementations*. Wiley. https://books.google.es/books?id=0dhQAAAAMAAJ

[23] F. Patzelt. 2022. Colored Noise. https://github.com/felixpatzelt/colorednoise.

[24] A. Pino, P. Khodashenas, X. Hesselbach, E. Coronado, and S. Siddiqui. 2021. Validation and Benchmarking of CNFs in OSM for pure Cloud Native applications in 5G and beyond. In *2021 Int. Conf. on Computer Communications and Networks (ICCCN)*. 1–9.

[25] JE Rachid and J Erfanian. 2015. NGMN 5G Initiative White Paper.

[26] J. X. Salvat, L. Zanzi, A. Garcia-Saavedra, V. Sciancalepore, and X. Costa-Perez. 2018. Overbooking Network Slices through Yield-Driven End-to-End Orchestration. In *Proc. Int. Conf. Emerging Networking EXperiments and Technologies (CoNEXT)*. 353–365.

[27] S. Saxena and K. M. Sivalingam. 2022. Slice admission control using overbooking for enhancing provider revenue in 5G Networks. In *NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium*. 1–7.

[28] C. Sexton, N. Marchetti, and L. A. DaSilva. 2020. On Provisioning Slices and Overbooking Resources in Service Tailored Networks of the Future. *IEEE/ACM Transactions on Networking* 28, 5 (2020), 2106–2119.

[29] S. D. A. Shah, M. A. Gregory, and S. Li. 2021. Cloud-Native Network Slicing Using Software Defined Networking Based Multi-Access Edge Computing: A Survey. *IEEE Access* 9 (2021), 10903–10924.

[30] K. T. Talluri and G. Van Ryzin. 2004. *The theory and practice of revenue management*. Vol. 1. Springer.

[31] Z. Tang, F. Zhang, X. Zhou, W. Jia, and W. Zhao. 2022. Pricing Model for Dynamic Resource Overbooking in Edge Computing. *IEEE Transactions on Cloud Computing* (2022).

[32] The Linux Foundation. 2022. The Linux Foundation and Google Cloud Launch Nephio to Enable and Simplify Cloud Native Automation of Telecom Network Functions. Consulted on March 10th 2023.

[33] D. Tikunov and T. Nishimura. 2007. Traffic prediction for mobile network using Holt-Winter's exponential smoothing. In *Int. Conf. on Software, Telecommunications and Computer Networks*. IEEE, 1–5.

[34] S. Troia, R. Alvizu, and G. Maier. 2019. Reinforcement Learning for Service Function Chain Reconfiguration in NFV-SDN Metro-Core Optical Networks. *IEEE Access* 7 (2019), 167944–167957.

[35] N. Van Huynh, D. Thai Hoang, D. N. Nguyen, and E. Dutkiewicz. 2019. Optimal and Fast Real-Time Resource Slicing With Deep Dueling Neural Networks. *IEEE J. Selected Areas in Communications* 37, 6 (2019), 1455–1470.

[36] L. Zanzi, J. X. Salvat, V. Sciancalepore, A. Garcia-Saavedra, and X. Costa-Perez. 2018. Overbooking Network Slices End-to-End: Implementation and Demonstration. In *Proc. of ACM SIGCOMM*. 144–146.

[37] J. Zheng, P. Caballero, G. de Veciana, S. J. Baek, and A. Banchs. 2018. Statistical Multiplexing and Traffic Shaping Games for Network Slicing. *IEEE/ACM Transactions on Networking* 26, 6 (2018), 2528–2541. https://doi.org/10.1109/TNET.2018.2870184

[38] X. Zhou, R. Li, T. Chen, and H. Zhang. 2016. Network slicing as a service: enabling enterprises' own software-defined cellular networks. *IEEE Communications Magazine* 54, 7 (2016), 146–153.