Bega, D., et al. AZTEC: anticipatory capacity allocation for zero-touch network slicing, IEEE INFOCOM 2020 - IEEE Conference on Computer Communications 6-9 July 2020 (Virtual Conference). *IEEE, 2020, Pp. 794-803*
DOI: https://doi.org/10.1109/INFOCOM41043.2020.9155299

AZTEC: Anticipatory Capacity Allocation for Zero-Touch Network Slicing

Dario Bega*†, Marco Gramaglia†, Marco Fiore*, Albert Banchs*† and Xavier Costa-Perez‡
*IMDEA Networks Institute, Madrid, Spain, Email: {dario.bega, marco.fiore, albert.banchs}@imdea.org
†University Carlos III of Madrid, Madrid, Spain, Email: mgramagl@it.uc3m.es
§NEC Laboratories Europe, Heidelberg, Germany, Email: xavier.costa@neclab.eu

***Abstract*—The combination of network softwarization with network slicing enables the provisioning of very diverse services over the same network infrastructure. However, it also creates a complex environment where the orchestration of network resources cannot be guided by traditional, human-in-the-loop network management approaches. New solutions that perform these tasks automatically and in advance are needed, paving the way to *zero-touch* network slicing. In this paper, we propose `AZTEC`, a data-driven framework that effectively allocates capacity to individual slices by adopting an original multi-timescale forecasting model. Hinging on a combination of Deep Learning architectures and a traditional optimization algorithm, `AZTEC` anticipates resource assignments that minimize the comprehensive management costs induced by resource overprovisioning, instantiation and reconfiguration, as well as by denied traffic demands. Experiments with real-world mobile data traffic show that `AZTEC` dynamically adapts to traffic fluctuations, and largely outperforms state-of-the-art solutions for network resource orchestration.**

I. Introduction

Mobile data traffic is becoming more and more heterogeneous: the variety of applications run by a single smartphone is staggering [1], and a growing number of devices are becoming always connected, heralding new and distinctive traffic demand models [2]. From the infrastructure provider's viewpoint, this trend imposes the need to support increasingly diverse specifications at once, and makes service differentiation a prime requirement for future-generation mobile networks.

Zero-touch network slicing. Network slicing enables the desired strong service differentiation by capitalizing on recent developments in Network Function Virtualization (NFV). It creates multiple logical instances of the physical network, the so-called *network slices*, ensuring strict traffic isolation among them [3], and tailoring the network resources of each slice to a specific (class of) application [4]. Network slicing has the potential to enable the coexistence of a wide range of mobile services in the same network infrastructure; however, it also poses several of technical challenges.

A prominent difficulty is resource management. Isolation of resources across slices inherently increases network capacity requirements [5], and a dynamic, preemptive and efficient allocation of resources to slices is a key instrument to keep capital expenditure (CAPEX) and operating expenses (OPEX) under control in sliced networks [6]. The rapid fluctuations in service demands, as well as the size and complexity of the slicing ecosystem, make legacy reactive, human-driven approaches to resource management inadequate to the emerging context. In sliced networks, automated decisions on resource assignment shall be taken by open-source orchestrators such as those promoted by Open Source MANO (OSM) [7] and Open Network Automation Platform (ONAP) [8]. By running dedicated Artificial Intelligence solutions [9], network orchestrators are expected to enable the vision of *zero-touch networks* [10], *i.e.*, fully self-operating communication infrastructures whose standardization has lately started to be discussed [11].

Capacity forecasting for resource management. When developing the intelligence for the automated allocation of resources in zero-touch network slicing, forecasting holds a fundamental role. Indeed, the orchestrator needs to know in advance the capacity that will be required by each slice to take informed decisions and maximize resource utilization. Unlike traditional prediction, network capacity forecasting is driven by monetary costs: errors lead to resource misconfigurations that entail different economic penalties for the operator [12].

Current state-of-the-art solutions for capacity forecasting in network slicing take into account the costs due to (i) the allocation of unnecessary resources that go unused, and (ii) the insufficient provisioning of resources that cannot accommodate the demand and lead to violations of the Service-Level Agreements (SLA) with the slice tenant. Hence, they aim at minimizing overprovisioning while avoiding SLA violations.

However, limiting the problem to this simple trade-off implicitly assumes that resource instantiation and reconfiguration occurs at no cost. While this may hold for some types of resource (*e.g.*, CPU time within the same bare metal machine), it is not generally valid for slice resource management scenarios. Instantiation and reconfiguration costs are capital in NFV technologies that enable the *cloudification* of the access and core networks by entrusting many network functions to Virtual Machines (VMs) running in datacenters. Examples include baseband processing in Cloud Radio Access Networks (C-RAN) [13], interconnection functionalities towards the external packet networks through the User Plane Function (UPF) [14], or central office operations [15].

In all the above cases, resource instantiation does not take place for free: VM boot times in prominent public cloud services like Amazon AWS or Microsoft Azure consistently exceed 40 seconds, topping at 400 seconds in worst-case scenarios [16]; even in recent tests, booting a lightweight VM containing an Alpine Linux takes around 30 seconds in a local deployment [17]. Reconfiguring already allocated resources has also a non-negligible cost: modern software architectures

such as Kubernetes need several seconds to execute new pods, *e.g.*, on VMs that are already running [17]. In addition, re-orchestration often implies recomputing paths on the transport networks and implementing them via, *e.g.*, Software Defined Networking (SDN) architectures: the latency is in the order of hundreds of milliseconds in a small five-switch topology and with precomputed routing [18], and this figure has to be scaled to thousands of switches with on-the-fly path re-calculation.

All unavoidable delays above entail monetary fees for the operator, in terms of both violations of the SLA with the tenants (*e.g.*, due to infringement of guarantees on end-to-end latency), and user dissatisfaction (with ensuing high churn rates). By neglecting these sources of cost, present capacity forecast solutions risk to introduce uncontrolled data flow latency once deployed in operational networks, ultimately causing economic losses to the operator.

Key contribution. In this paper, we propose an original model for the anticipatory allocation of capacity to network slices, which is mindful of all operating costs associated to

- (i) unnecessary resource overprovisioning,
- (ii) non-serviced demands,
- (iii) resource instantiation, and
- (iv) resource reconfiguration.

To this end, we adopt a novel approach is based on the concept of *multi-timescale orchestration* illustrated in Figure 1.

On the left, a state-of-the-art solution for capacity forecasting [19] tries to accommodate the demand and to limit overprovisioning, by reconfiguring resources at every re-orchestration opportunity (top); by doing so, it minimizes costs (i) and (ii) above (second plot). However, it also ceaselessly instantiates or de-commissions capacity, and reallocates available resources in a sustained way. This incurs in substantial instantiation and reconfiguration fees (third plot) that ultimately lead to a high overall economic cost (bottom).

On the right, our model performs the orchestration at two timescales, and by telling apart two classes of resources. A *long-timescale orchestrator* operates over extended intervals that span multiple re-orchestration opportunities; it allocates a *dedicated capacity* to each slice and also reserves an additional *shared capacity* accessible by any slice. Both capacities remain constant across the extended interval, limiting the frequency of instantiation and thus cost (iii). Only the shared capacity is then reallocated at every re-orchestration opportunity by a *short-timescale orchestrator*, while the configuration of the dedicated capacity is preserved throughout the extended interval, thus reducing cost (iv). Both long- and short-timescale orchestrators decide on the amount of (dedicated and shared) resources to be allocated to each slice to also minimize the usual costs (i) and (ii). This comprehensive strategy results in a 47% reduction of the total cost in the example in Figure 1.

Interestingly, the sample case study also clarifies that reducing instantiation and reconfiguration costs has a price in terms of increased overprovisioning. A multi-timescale orchestration model allows exploring this and more trade-offs for the first time, empowering an unprecedently comprehensive solution

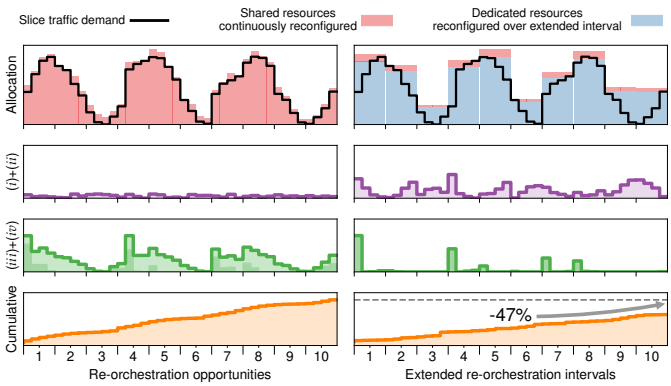

Fig. 1. Toy example illustrating the costs of resource allocation in network slicing. Top: traffic demand generated by a representative slice (black solid line), along with the capacity allocated based on predictions, and reconfigured at all available re-orchestration opportunities (*i.e.*, shared, in red) or only periodically over extended intervals (*i.e.*, dedicated, in blue). Second row: Monetary costs of (i) overprovisioning and (ii) non-serviced slice traffic. Third row: Monetary costs of resource (iii) instantiation and (iv) reconfiguration. The costs are obtained with a system configuration $\kappa_o = \kappa_s = \kappa_i = 1$ and $\kappa_r = 0.5$, whose meaning is explained in Section II. Bottom: cumulative overall cost over time, for components (i)-(iv). Left: a legacy capacity forecasting model [19] updates the prediction at the fastest rate possible, closely following the demand fluctuations, but forcing continuous reconfigurations. Right: our proposed multi-timescale capacity forecasting model trades slightly increased overprovisioning for much reduced instantiation costs (which are only incurred once per extended interval) and reconfiguration fees (which are completely avoided for dedicated resources).

for cost-driven orchestration of slice resources [20] via a mixture of Artificial Intelligence and traditional optimization.

Document outline. We present our orchestration model in Section II, where we formalize the different costs and trade-offs of sliced networks management. Building this model, in Section III we introduce AZTEC, a complete framework for the anticipatory allocation of capacity to network slices that relies on a combination of deep learning architectures and a numerical optimization method. When informed of the economic penalty associated to each source of cost, AZTEC anticipates the dedicated and shared capacity to be allotted to network slices so as to cut down monetary losses due to instantiation and reconfiguration, while keeping resource provisioning and non-serviced demands fees under control. We demonstrate the effectiveness of our solution with measurement data collected in a metropolitan-scale mobile network, in Section IV; in typical settings, AZTEC outperforms state-of-the-art traffic [21] and capacity [19] predictors by at least a factor of 1.7. Finally, conclusions are drawn in Section V.

II. Orchestration Model and Trade-offs

Our orchestration model is outlined in Figure 2, which also serves the purpose of illustrating the notation used in the remainder of the paper. Let us denote by $\lambda_i(t)$ the traffic demand generated by services running in slice $i \in \mathbb{S}$ at time t. The long-timescale orchestrator operates on extended intervals of duration T_l. At the beginning of each such interval, it takes decisions on the dedicated capacity $x_i^d(t)$ allotted to slice i, $\forall i \in \mathbb{S}$, and on the additional shared capacity $x^s(t)$ available to all slices; all capacities are conserved throughout the following

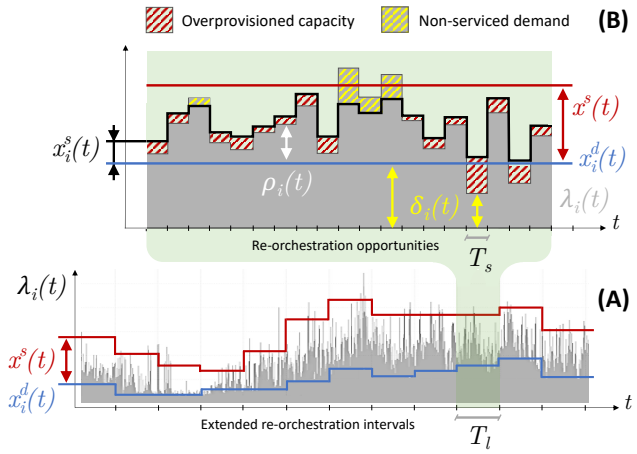

Fig. 2. Orchestration model. (A) Long-timescale orchestration. The background time series represents the traffic demand generated by slice i (grey). The curves portray the time evolution of the dedicated capacity $x_i^d(t)$ allocated to slice i (blue), and of the shared capacity $x^s(t)$ (red) over extended intervals of duration T_l. Note that $x^s(t)$ is added to the dedicated resources to determine the total available capacity, and, unlike $x_i^d(t)$, is not reserved for slice i but available to all slices. (B) Short-timescale orchestration during one extended interval. At every $T_s < T_l$, a portion $x_i^s(t)$ (black solid curve) of the (fixed) shared capacity $x^s(t)$ is allocated to slice i, based on the residual demand $\rho_i(t)$ not satisfied by the (fixed) dedicated resources $x_i^d(t)$. The plot also highlights the volume of overprovisioned capacity and non-serviced demand (pattern regions), and the slice traffic below dedicated capacity $\delta_i(t)$.

T_l. The bottom plot (A) in Figure 2 depicts an example of allocation resulting from a long-timescale orchestration.

Within an extended interval, the short-timescale orchestrator assigns resources to each slice i at all re-orchestration opportunities, occurring at every T_s. Decisions are based on the (estimated) future residual demand $\rho_i(t) = \max\{0, \lambda_i(t) - x_i^d(t)\}$, and lead to the allocation of an additional capacity $x_i^s(t)$, for each slice i. The resources $x_i^s(t)$ may be re-configured at every T_s, and are provisioned on top of the dedicated $x_i^d(t)$. The top plot (B) in Figure 2 illustrates these definitions for a sample short-timescale orchestration during one extended interval.

A. *Sources of monetary cost*

Building on the notation above, we can formally introduce the different costs associated to the management of resources in sliced networks. As anticipated in Section I, there are four sources of economic penalty for the operator, as follows.

(*i*) Unnecessary resource provisioning: the operator incurs a monetary cost in terms of both Capital Expenditure (CAPEX) and Operating Expenses (OPEX) that is directly proportional to the amount of unused resources it allocates to a slice. Such capacity it is instantiated and configured to no purpose and could be allotted, *e.g.*, to other slices to increase the global system efficiency. This cost at time t is

$$\sum_{i\in\mathbb{S}} f_1\left(\max\{0, x_i^d(t) - \delta_i(t)\}\right) + \sum_{i\in\mathbb{S}} f_1\left(\max\{0, x_i^s(t) - \rho_i(t)\}\right) + f_1\big(x^s(t) - \sum_{i\in\mathbb{S}} x_i^s(t)\big), \quad (1)$$

where $\delta_i(t) = \min\{\lambda_i(t), x_i^d(t)\}$ denotes the portion of the demand of slice i served by the dedicated capacity at time t, as shown in plot (B) of Figure 2. The first two terms in (1) denote the cost of overprovisioning at slice i and time t, due to the unneeded allocation of dedicated and shared capacity, respectively; again, we refer the reader to plot (B) of Figure 2 for an exemplification. The third term captures instead the overprovisioned shared capacity that is not allocated to any slice by the short-term orchestrator. Function $f_1(\cdot)$ describes the scaling of cost with capacity overprovisioning. As in [19], in our evaluation we assume a linear increase of the penalty, *i.e.*, $f_1(x) = \kappa_o x$, where κ_o is the monetary cost of one unit of capacity and is expressed in \$/Mbps. However, our model can easily accommodate different definitions of the scaling law, which may apply to specific network functions.

(*ii*) Non-serviced demand: every time the operator does not allocate sufficient resources to serve the traffic demand of a slice, it violates the SLA with the tenant, which triggers a monetary compensation. The associated cost at time t is

$$\sum_{i\in\mathbb{S}} \kappa_s \cdot \mathbb{1}_{<\rho_i(t)}\left(x_i^s(t)\right), \quad (2)$$

where $\mathbb{1}_A(x)$ is an indicator function that takes a value 1 if the argument satisfies condition A, and 0 otherwise. Thus, $\mathbb{1}_{<\rho_i(t)}\left(x_i^s(t)\right)$ activates when the portion of shared capacity assigned to i does not suffice to meet the service demand; this corresponds to an underprovisioning situation, as depicted in Figure 2. In these cases, the operator has to indemnify the tenant for a value κ_s, in \$, per SLA violation. This definition is also in line with those used in the literature [19].

(*iii*) Resource instantiation: in presence of substantial variations of the total traffic demand, the operator needs to instantiate new resources to serve the demand of the slice. In these cases, as discussed in Section I, there exists a cost associated to enabling such new resources. As an example, if additional Virtual Machines (VMs) have to be bootstrapped or migrated to serve the slice, the operator has increased expenses in terms of power consumption and CPU cycles. In addition, there may be an indirect penalty in terms of perceived Quality of Service (QoS), as this operation may take minutes [17] and disrupt the end user experience. The cost, triggered at every T_l in our multi-timescale model, can be modeled as

$$\sum_{i\in\mathbb{S}} f_2\left(\delta_i(t)\right) \cdot \mathbb{1}_{>x_i^d(t-1)}\left(x_i^d(t)\right) + f_2\left(\min\{\rho_i(t), x_i^s(t)\}\right) \cdot \mathbb{1}_{>x^s(t-1)}\left(x^s(t)\right). \quad (3)$$

The first term in (3) represents the penalty incurred when new dedicated resources must be instantiated, which occurs when $x_i^d(t) > x_i^d(t-1)$. The second term is equivalent to the first one, but refers to the shared capacity instantiated to all slices in $\mathbb{S}$. Note that the costs induced by both terms are functions $f_2(\cdot)$ of the affected traffic that may experience disruption, *i.e.*,

$\delta_i(t)$ and $\min\{\rho_i(t), x_i^s(t)\}$, respectively.[1] In our performance evaluation, we consider the cost to be directly proportional to the affected traffic, *i.e.*, $f_2(x) = \kappa_i x$, where the parameter κ_i captures the estimated fee for delaying one unit of capacity due to resource instantiation, expressed in \$/Mbps.

(*iv*) Resource reconfiguration: while resources are only instantiated at every T_l, a short-timescale orchestration of the shared capacity within each extended interval allows accommodating faster fluctuations of the slice demand. Every time the operator reconfigures the shared capacity, it incurs a cost; as mentioned in Section I, this is the case with the reconfiguration of the SDN transport networks, the setup of load balancers, or the creation of new instances of a VNF on a VM previously used by another slice. All these operations have a price in terms of management delay [17], expressed as

$$\sum_{i \in \mathbb{S}} f_3\left(\min\{\rho_i(t), x_i^s(t)\}\right) \cdot \mathbb{1}_{\neq x_i^s(t-1)}\left(x_i^s(t)\right). \qquad (4)$$

The above cost is present whenever the shared resources must be reconfigured for a slice i, *i.e.*, $x_i^s(t) \neq x_i^s(t-1)$. In such situations, the cost is dependent on the amount of traffic affected by the reconfiguration process, *i.e.*, $\rho_i(t)$ bounded by $x_i^s(t)$. In our study, we assume that the economic penalty is the same for any bit of traffic using reconfigured resources, hence $f_3(x) = \kappa_r x$, where κ_r is in \$/Mbps. Also in this case, other functions can be easily embedded in the overall framework to represent distinctive cost models identified by the operator.

B. Trade-offs in capacity allocation

The basic trade-off in anticipatory resource assignment is that between overprovisioning and non-serviced demands.

- **Trade-off A.** Increasing the amount of resources makes overprovisioning more likely, but reduces the probability that the allocated capacity is not sufficient to serve the future demand. This results in opposing costs (i) and (ii).

Current capacity forecasting models aim at identifying the optimal compromise that minimizes the joint penalty of the costs in trade-off A above [19]. However, these models do not offer any control over instantiation and reconfiguration. By adopting a multi-timescale approach, we are instead capable of factoring such variables in. Specifically, the model presented in Section II-A tells apart the capacity allocated to each slice into a dedicated capacity, re-orchestrated over long timescales with period T_l, and a shared capacity, re-orchestrated over short timescales with period T_s. This unlocks additional degrees of freedom: the orchestrator can decide not only how many resources to assign to a slice, but also which portion of those shall be of each type, and for how long they stay unaltered.

Our model still allows addressing trade-off A above, by controlling the total allocated capacity during each extended interval T_l, *i.e.*, $x^s(t) + \sum_{i \in \mathbb{S}} x_i^d(t)$, and modulate the costs of overprovisioning and non-serviced demands. Yet, its flexibility enables the exploration of the following additional trade-offs.

- **Trade-off B.** By increasing the dedicated capacity $x_i^d(t)$ allocated to slice i during an extended interval, the orchestrator can serve a larger fraction of the slice traffic with resources that do not need reconfiguration. However, such resources cannot be reused by other slices during T_l whenever they are not needed by slice i. For instance, in plot (B) of Figure 2, increasing $x_i^d(t)$ would reduce the residual demand $\rho_i(t)$ that is served with reconfiguration-heavy shared capacity; but it would also generate additional overprovisioning, *e.g.*, in the fourth and second to last re-orchestration opportunities. This leads to a trade-off between costs (i) and (iv).
- **Trade-off C.** Allocating a larger shared capacity $x_i^s(t)$ to slice i during an interval T_s reduces the risk that the resources will not be sufficient to serve the future slice demand. Nevertheless, it also causes the reconfiguration of more resources. As an example, in plot (B) of Figure 2, increasing $x_i^s(t)$ in the third T_s slot could remove the non-serviced demand, but would also grow the reconfiguration penalty. A trade-off exists between costs (ii) and (iv).
- **Trade-off D.** Increasing the duration T_l of the extended interval reduces the cost of resource instantiation, which only occurs once per extended interval. However, a higher T_l also forces the dedicated capacities $x_i^d(t)$ and the total shared capacity $x^s(t)$ to remain constant for a longer time. With a reduced capability to tailor the network resources to the fluctuations of the slice traffic demands, the orchestrator may incur in increased overprovisioning or underprovisioning. For instance, extending the timespan of plot (B) in Figure 2 to cover a prolonged demand $\lambda_i(t)$ may create additional situations where $x_i^d(t) > \lambda_i(t)$, *i.e.*, dedicated resources go wasted, as in the second to last T_s interval; it can also generate new cases where $x_i^s(t) < \rho_i(t)$, and traffic peaks are not serviced, as in the central part of the example. This results in a trade-off between cost (iii) and joint costs (i) and (ii).

Next, we present a framework that takes automated, anticipatory decisions on capacity allocation by addressing all trade-offs outlined above, thanks to the multi-timescale model.

III. The AZTEC framework

Our framework, named AZTEC (*i.e.*, capacity Allocation for Zero-Touch nEtwork sliCing), automatically solves trade-offs A, B and C above by finding en effective compromise among the opposing goals of reducing the operator's costs in terms of (i) overprovisioning, (ii) non-serviced slice demands, and (iv) resource reconfiguration. In addition, AZTEC offers the operator a handle to control, via a single system parameter, the penalty associated with (iii) capacity instantiation, to address trade-off D. Next, we first provide an overview of the framework in Section III-A, and then discuss the implementation of its different components for long- and short-timescale orchestration, in Sections III-B and III-C, respectively.

[1] Instantiation and reconfiguration costs are proportional to the full amount of impacted demand (rather than, *e.g.*, to the increment of demand with respect to the previous re-orchestration opportunity) since finding a stable and optimal allocation of resources typically requires reconsidering their organization within a large portion of the network, or even across all slices [22].

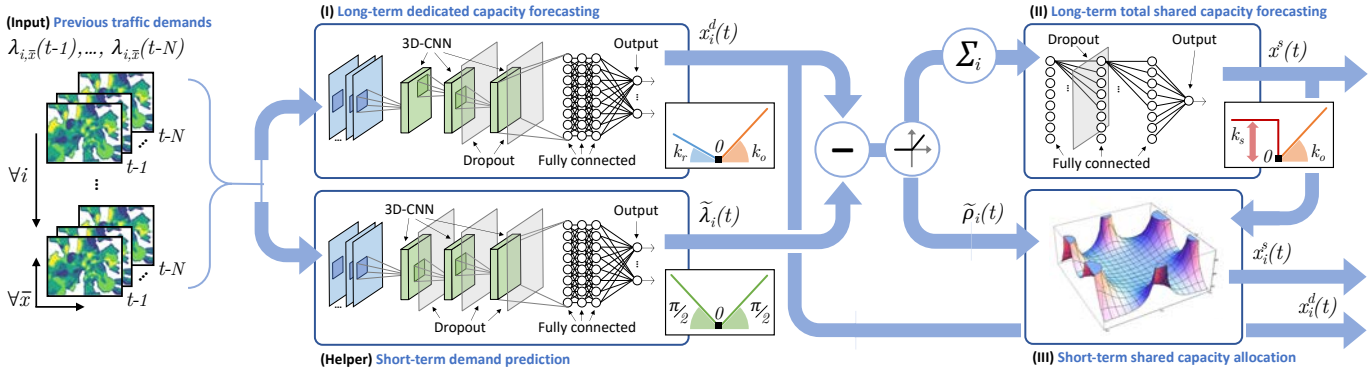

Fig. 3. Overview of the AZTEC framework. The learning flow proceeds from left to right. The input mobile traffic data is processed by deep neural networks that return, for each slice $i \in \mathbb{S}$, the long-term dedicated capacity $x_i^d(t)$ and the short-term estimated demand $\tilde{\lambda}_i(t)$, respectively. These values are combined to obtain the estimated residual demands $\tilde{\rho}_i(t)$. The aggregate residual demand over all slices is input to a further deep neural network to determine the long-term shared capacity $x^s(t)$. Such capacity is then fed, along with per-slice residuals, to an optimization module that allocates the shared resources $x_i^s(t)$.

A. AZTEC *in a nutshell*

To solve the complex problem of finding an adequate balance of all fees, we adopt a *divide-et-impera* approach. We separate the different trade-offs between pairs of cost sources and sequentially solve them in isolation. The overall organization of our proposed framework is outlined in Figure 3.

AZTEC performs both long- and short-timescale orchestration. As explained in Section I, the long-timescale orchestrator triggers at the beginning of each extended interval, and is in charge of allocating the dedicated capacities $x_i^d(t)$ and the total shared capacity $x^s(t)$, which will then be preserved over the following T_l interval. This function is realized in our framework by blocks (I) and (II), which operate as follows.

Block (I) performs the forecasting of the long-term dedicated capacity for each network slice $x_i^d(t)$, using as input information about the actual traffic generated by each slice during the preceding N re-orchestration opportunities. As discussed for trade-off B in Section II-B, the capacity $x_i^d(t)$ modulates the impact of (i) provisioning unnecessary dedicated resources versus (iv) re-configuring the shared resources needed to serve the residual demand beyond $x_i^d(t)$. Hence, block (I) identifies $x_i^d(t)$ minimizing costs (i) and (iv).

Block (II) determines the long-term shared capacity $x^s(t)$ available to any slice during the subsequent time interval T_l. The shared capacity is used to serve the residual demands of all slices, hence $x^s(t)$ shall be dimensioned to the aggregate residual traffic $\sum_{i\in\mathbb{S}} \rho_i(t)$. Thus, block (II) receives as input an estimate[2] of such aggregate during the previous extended interval, *i.e.*, $\sum_{i\in\mathbb{S}} \tilde{\rho}_i(t)$, $t \in [t-(T_l/T_s), t-1]$, and predicts the next $x^s(t)$ such that (i) overprovisioning of shared resources is reduced as much as possible, and (ii) all residual demands can be accommodated within $x^s(t)$. In this way, block (II) addresses trade-off A, jointly minimizing costs (i) and (ii).

Once the long-term capacities $x_i^d(t)$, $\forall i \in \mathbb{S}$, and $x^s(t)$ are set, the short-timescale orchestrator assigns portions $x_i^s(t)$ of the total shared resources to each slice. This allocation occurs at every re-orchestration opportunity, spaced by T_s, and is carried out by block (III) of the AZTEC framework as follows.

Block (III) receives as input the long-term shared capacity $x^s(t)$, and the future residual demand $\tilde{\rho}_i(t)$ expected for each slice i during the following T_s. The available total capacity is allotted to each slice in a way to solve the trade-off C described in Section II-B. Therefore, block (III) computes $x_i^s(t)$ for each $i \in \mathbb{S}$, by minimizing the combination of costs (ii) and (iv), corresponding to insufficient allocated capacity and additional shared resource reconfiguration, respectively.

Overall, blocks (I)-(III) return a forecast of all capacities $x_i^d(t)$, $x^s(t)$ and $x_i^s(t)$ that the operator shall allocate over both long and short intervals of duration T_l and T_s, respectively. The resulting anticipatory allotment reduces the network management costs associated with the provisioning of exceeding or inadequate resources, and of their reconfiguration over time.

We remark that the penalty of network resource instantiation is not included in this picture. As explained in Section II-B, control on instantiation costs can be achieved by acting on the duration of the extended re-orchestration interval, *i.e.*, T_l: the larger this duration, the lower the costs resulting from resource instantiation in (3). AZTEC does not take automated decisions on the value of T_l; instead, it provides via the parameter T_l an explicit knob for the operator to implement any strategy for coping with trade-off D. The rationale is that T_l is often a constant system setting, constrained by the underlying virtualization technology and best set by the operator based on its expert knowledge of the network architecture.

Having clarified the role of each block, we detail in the following their implementation, which leverages a combination

[2]Each approximate $\tilde{\rho}_i(t) = \max\{0, \tilde{\lambda}_i(t) - x_i^d(t)\}$ is computed from a forecast $\tilde{\lambda}_i(t)$ returned by a *helper* legacy traffic predictor that forecasts per-slice demands over the next re-orchestration opportunity t, as per Figure 3. Note that, at this stage, the actual residuals $\rho_i(t)$ could be directly determined from the (known) real traffic demand $\lambda_i(t)$ observed during the previous interval T_l. However, we opt to feed block (II) with an estimate based on $\tilde{\lambda}_i(t)$ to ensure overall system consistency. Indeed, the short-timescale orchestrator –implemented by block (III) and presented next– necessarily operates on the predicted residuals $\tilde{\rho}_i(t)$ over the future T_s interval. Considering the same estimates in the long-timescale module allows allocating the shared capacity $x^s(t)$ in a way that is conscious of the inaccuracy of the information available during the following short-term resource assignment phase of the framework.

of deep learning and numerical optimization methods.[3]

B. Long-timescale orchestration

The long-timescale orchestration is carried out by blocks (I) and (II) of the AZTEC framework, as follows.

1) Long-term dedicated capacity forecasting: Block (I) is implemented by a Deep Neural Network (DNN) whose architecture is inspired by the one originally proposed in [19], which builds on recent breakthroughs in machine learning for image processing [23]. This approach requires pre-processing the mobile data traffic, so as to map base station positions into the matrix form required by DNN, which we do by means of a correlation-preserving transformation [19]. The resulting input is a 4D tensor $\lambda_{i,\bar{x}}(t-1), \ldots, \lambda_{i,\bar{x}}(t-N)$, where $\lambda_{i,\bar{x}}(t)$ is the offered load at the base station associated with matrix element $\bar{x} = \{m, n\}$, for services running in slice $i \in \mathbb{S}$ and at time t. We enhance the implementation in [19] by treating each slice $i \in \mathbb{S}$ in the same way of a color channel in DNN for imaging, as also suggested by recent works in machine learning for mobile network traffic analysis [24]. This approach lets us process the input along the $\{\bar{x}, t\}$ dimensions via 3D Convolutional Neural Network (3D-CNN) layers, which are very efficient in extracting spatiotemporal features; at the same time, different slices i can be examined in parallel as multiple levels of the same data.

The DNN architecture of block (I), summarized in Figure 3, consists of three 3D-CNN layers interleaved by dropout layers [25]. The convolutional layers constitute the encoder, in charge of extracting meaningful complex features from the data. They are followed by a decoder whose objective is learning global patterns from the feature space. Fully Connected (FC) layers are especially well suited to that purpose, and we leverage three in our implementation, with 64, 32, and $\|\mathbb{S}\|$ neurons, respectively, where operator $\|\cdot\|$ denotes the cardinality of the argument set. Note that the last layer outputs one value per channel, resulting in one value for each slice $i \in \mathbb{S}$. All layers employ ReLU as the neuron activation function, except for a linear function in the last FC layer.

The loss function that drives the DNN training is a custom expression designed to account for the actual management costs incurred by the operator in case of errors in the orchestration of the dedicated capacity. If the operator were able to allocate to a slice i a constant capacity $x_i^d(t)$ that perfectly matched the actual demand $\lambda_i(t)$ over the next T_l, the error and cost would be nil: this is the ideal scenario where all the demand is serviced, without any overprovisioning or reconfiguration. However, in practical cases, it is impossible to perfectly predict $\lambda_i(t)$, which is also very unlikely to be constant over the whole T_l. In this case, positive errors $x_i^d(t) - \lambda_i(t)$ lead to overprovisioning, with a cost set by the first term of (1) in Section II-A, and negative errors imply that the demand in excess of $x_i^d(t)$ needs to be served by the shared capacity, with (4) setting the re-configuration penalty.[4]

Positive errors yield $\delta_i(t) = \lambda_i(t)$, while $\rho_i(t) = \lambda_i(t) - x_i^d(t)$ for negative errors. Then, the loss function for $x_i^d(t)$ allocated at t is $\sum_{t \in \mathbb{T}} \ell_i^{(\mathrm{I})}(t)$, where $\mathbb{T}$ is the set of concerned re-orchestration opportunities $\{t, \ldots, t + (T_l/T_s) - 1\}$, and

$$\ell_i^{(\mathrm{I})}(t) = \begin{cases} f_3\left(\lambda_i(t) - x_i^d(t)\right) & \text{if } x_i^d(t) \leq \lambda_i(t) \\ f_1\left(x_i^d(t) - \lambda_i(t)\right) & \text{otherwise.} \end{cases} \quad (5)$$

Importantly, (5) has partial derivatives that form a piece-wise constant function, which guarantees robust and fast convergence under popular first-order optimizers like Adam [26].

In order to further improve the quality of the allocation of dedicated resources, we leverage a recent result in neural network design, which allows estimating the uncertainty of the learning outcome. Specifically, adding dropout layers during model testing is mathematically equivalent to generating an approximation of the probabilistic deep Gaussian process [27]. This observation, which holds for DNNs with arbitrary depth and non-linearities, provides a way to return a distribution instead of a scalar output value. We thus activate the dropout layers during testing, and adopt a Monte Carlo strategy; namely, we perform the forward pass L times for each test input, obtaining outputs $\{x_i^{d,1}(t), \ldots, x_i^{d,L}(t)\}$ for, slice i at time t. We then compute the mean $\mu_i^d(t) = 1/L \cdot \sum_{l=1}^{L} x_i^{d,l}(t)$ as well as the variance $\sigma_i^d(t) = 1/L \cdot \sum_{l=1}^{L} (\mu_i^d(t) - x_i^{d,l}(t))^2$, and approximate the model uncertainty as $\mathcal{N}(\mu_i^d(t), \sigma_i^d(t))$.

Knowledge of the model uncertainty allows adding a safety margin to the standard DNN outcome. Since the whole support of $\mathcal{N}(\mu_i^d(t), \sigma_i^d(t))$ represents potentially correct values of the dedicated capacity, block (I) returns the 99^{th} percentile of the distribution; this makes it very unlikely to output a $x_i^d(t)$ that is lower than the best one, minimizing the risk of SLA violations. In other words, when the DNN is confident about the quality of the result, it returns a value close to the mean; otherwise, it adds a substantial safety margin. We provide an example of the advantage of this approach in Section IV-A.

2) Long-term total shared capacity forecasting: Block (II) is similarly implemented using a dedicated DNN, although with a simpler structure due to the reduced richness of the input. Specifically, the DNN is fed a single time series of the total residual demand in the past extended interval, *i.e.*, $\sum_{i \in \mathbb{S}} \tilde{\rho}_i(t)$, $t \in [t-(T_l/T_s), t-1]$. The time series is processed by three FC layers with 128, 64 and 1 neurons, respectively; the first two use ReLU activation functions, while the last uses a linear function. A dropout layer is present between the first and second FC layers.

The DNN is trained with a different custom loss function that accounts for the correct sources of monetary penalty in case of errors. As with the previous DNN, an ideal case where a fixed long-term shared capacity $x^s(t)$ perfectly matches a

[3] A complete Python implementation of the AZTEC framework based on TensorFlow is available at https://github.com/wnluc3m/AZTEC.

[4] As the long-term orchestrator is agnostic of the short-timescale operation, it cannot factor $x_i^s(t)$ in the cost of negative errors. So, block (I) assumes a perfect management of the shared capacity that always accommodates a continuously varying residual demand, reducing (4) to $\sum_{i \in \mathbb{S}} f_3\left(\rho_i(t)\right)$.

constant aggregate residual demand is unrealistic, and errors are unavoidable. Positive errors yield overprovisioning of $x^s(t)$, whereas negative ones have a cost in terms of denied traffic. The former corresponds to the second and third terms[5] in (1), while the latter maps to the monetary fee[6] for SLA violations in (2). The loss function jointly capturing these costs for the extended interval starting at t is $\sum_{t\in\mathbb{T}} \ell_i^{(\mathrm{II})}(t)$, where

$$\ell_i^{(\mathrm{II})}(t)=\begin{cases}\kappa_s & \text{if } x^s(t)<\sum_{i\in\mathbb{S}}\tilde{\rho}_i(t)\\ f_1\left(x^s(t)-\sum_{i\in\mathbb{S}}\tilde{\rho}_i(t)\right) & \text{otherwise.}\end{cases} \quad (6)$$

The expression in (6) needs to be slightly modified by adding minimum slopes that make the function differentiable over $\mathbb{R}$. With this, the loss function has the same desirable properties as the ones mentioned for (5). Finally, we take advantage of the dropout layer also in this case: we thus approximate the model uncertainty, and return the 99^{th} percentile of the distribution as a safety margin on the correct value of $x^s(t)$.

C. Short-timescale orchestration

The short-timescale orchestration consists of block (III) supported by a *helper* short-term demand predictor, as outlined in Figure 3. The predictor uses a DNN architecture that is very similar to that adopted by block (I); indeed, the two DNNs operate on the same input, and produce per-slice forecasts. The main differences between them are in the frequency of operation and, most notably, in the loss function. The helper predictor outputs a prediction at every T_s instead of at every T_l. Furthermore, it uses a traditional Mean Absolute Error (MAE) instead of the cost-aware loss function in (5): MAE considers identical contributions by the positive and negative errors, thus producing an output $\tilde{\lambda}_i(t)$ that tries to follow as closely as possible the upcoming slice traffic demands.

1) Short-term shared capacity allocation: Given the total shared capacity $x^s(t)$, and the estimated residual demands $\tilde{\rho}_i(t)$, AZTEC has to decide how to distribute $x^s(t)$ across the requesting slices at every T_s. This is implemented in block (III) with a numerical optimization method.

The primary objective of the shared resource assignment performed by block (III) is avoiding SLA violations due to insufficient available capacity, which would induce a cost modelled in (2). At the same time, issuing non-essential resources has a penalty in terms of unnecessary reconfiguration cost, as captured by the expression in (4). This corresponds to trade-off C in Section II-B, and can be formulated as

$$\begin{aligned}\min_{x_i^s(t)} \quad & \sum_{i\in\mathbb{S}} \kappa_s P\left(\tilde{\rho}_i(t) > x_i^s(t)\right) \\ \text{subject to} \quad & \sum_{i\in\mathbb{S}} x_i^s(t) \le x^s(t),\end{aligned} \quad (7)$$

[5]Also in this case, the shared capacity allotted to individual slices $x_i^s(t)$ used in (1) has not yet been determined at this stage. The safest option is hence to assume that the whole residual demands will be correctly assigned by the short-term orchestrator. This leads to approximating the allocated shared resources $x_i^s(t)$ by $\tilde{\rho}_i(t)$. Under this hypothesis, the second and third terms in (1) reduce to 0 and $f_1\left(x^s(t)-\sum_{i\in\mathbb{S}}\tilde{\rho}_i(t)\right)$, respectively.

[6]By assuming $x_i^s(t)=\tilde{\rho}_i(t)$, unserviced demands are possible only when $x^s(t)$ is insufficient. Then, (2) translates into $\kappa_s \cdot \mathbb{1}_{<\sum_{i\in\mathbb{S}}\tilde{\rho}_i(t)}\left(x^s(t)\right)$.

Algorithm 1: Shared resource assignment algorithm

Function *TRANSFORM*$(p_1,\dots,p_{\|\mathbb{S}\|})$**:**
  $x^s_{\|\mathbb{S}\|} = \frac{p_{\|\mathbb{S}\|}\,x^s}{\left(\sum_{i=1}^{\|\mathbb{S}\|-1}\prod_{j=i}^{\|\mathbb{S}\|-1}\frac{p_j}{1-p_j}\right)+1}$
  $x_i^s = \left(\prod_{j=i}^{\|\mathbb{S}\|-1}\frac{p_j}{1-p_j}\right)x^s_{\|\mathbb{S}\|},\ i\in[1,\|\mathbb{S}\|-1]$
  return $x_1^s,\dots,x^s_{\|\mathbb{S}\|}$
Function *COST*$(p_1,\dots,p_{\|\mathbb{S}\|})$**:**
  $x_1^s,\dots,x^s_{\|\mathbb{S}\|} \leftarrow$ *TRANSFORM*$(p_1,\dots,p_{\|\mathbb{S}\|})$
  $X \leftarrow \sum_i \kappa_s P\left(\tilde{\rho}_i > x_i^s\right)$
  return X
Function *MAIN*$(x^s,\tilde{\rho}_i)$**:**
  $c \leftarrow$ BOBYQA(COST$(p_1=0.5, p_2=0.5,\dots,p_{\|\mathbb{S}\|-1}=0.5, p_{\|\mathbb{S}\|})$)
  $p_1^0,\dots,p^0_{\|\mathbb{S}\|} \leftarrow$ *GOLDEN* $(c(p_{\|\mathbb{S}\|}))$
  $p_1,\dots,p_{\|\mathbb{S}\|} \leftarrow$ *BOBYQA* $(p_1^0,\dots,p^0_{\|\mathbb{S}\|})$
  $x_1^s,\dots,x^s_{\|\mathbb{S}\|} \leftarrow$ *TRANSFORM*$(p_1,\dots,p_{\|\mathbb{S}\|})$
  return $x_1^s,\dots,x^s_{\|\mathbb{S}\|}$

The above minimizes the expected value of the expenditure for non-serviced slice demands in (2). Note that, by squeezing as many slices as possible within the total capacity, the solution to (7) implicitly addresses the challenge of minimizing reconfiguration costs. Also, the formulation in (7) deals with probabilities: this is consistent with the probabilistic nature of $\tilde{\rho}_i(t)$ granted by uncertainty approximation via dropout layers.

Due to the empirical nature of the probability distribution $\tilde{\rho}_i(t)$, (7) has no closed form and thus we cannot employ classical optimization methods. Furthermore, as we do not have a differentiable objective function, we cannot apply approaches that depend on gradients. Hence, we resort to numerical methods that search for the optimal solution within the feasible set of $[x_i^s(t)]$. The following change of variable

$$p_i(t) = \begin{cases} x_i^s(t)/\left(x_i^s(t)+x_{i+1}^s(t)\right) & \text{if } i\in[1,\|\mathbb{S}\|-1] \\ \sum_i x_i^s(t)/x^s(t) & \text{if } i=\mathbb{S}\end{cases} \quad (8)$$

yields $p_i(t)\in[0,1],\ \forall i\in\mathbb{S}$, which simplifies the search to the N-dimensional variable space with fixed bounds $[0,1]$ on all variables. Note that the first $\|\mathbb{S}\|-1$ variables $p_i(t)$ represent the relative amount of shared capacity assigned to slice i with respect to the slice $i+1$, while the last $p_{\|\mathbb{S}\|}(t)$ represents the overall amount of shared capacity assigned to the services. Therefore, the variable change in (8) serves the following purposes: first, the constraint on the sum of the variables is now enforced through a constraint on each variable; second, we avoid ties between variables, allowing a safe exploration of the solution space where we can focus one variable, and changing its $p_i(t)$ value within the entire range without impacting any of the other variables $p_j(t)$ for $j\neq i$.

Algorithm 1 details the numerical solution for shared resource assignment adopted by AZTEC. A main function takes as input the total available shared capacity $x^s(t)$ and the empirical distribution of the capacity needed by each service in the next time interval $\tilde{\rho}_i(t)$. Two helper functions, COST and TRANSFORM, compute the cost expected value for a given assignment $p_1(t),\dots,p_{\|\mathbb{S}\|}(t)$, and transform $p_i(t)$ back to $x_i^s(t)$, respectively. Thanks to the variable change in (8), we can use a gradient-free algorithm which works with constrained input variable for the minimization of (7). In particular, we chose as numerical optimizer the BOBYQA algorithm [28], which is a gradient-free optimizer that allows constrained variables.

All all ratios of shared capacity are initialized to 0.5, *i.e.*, all slice start with the same amount of resources. However, given the possibly high number of slices that can be included in the system, there may be different local minima and there exists the chance of getting stuck in a local minima that does not deliver a good performance. To reduce the probability that this happens, we perform a preliminary search for the best starting $p_{\|\mathbb{S}\|}(t)$, via the Golden Section method [29] and considering $p_{\|\mathbb{S}\|}(t)$ as the only variable for the cost.

IV. Evaluation Results

We evaluate AZTEC with an extensive dataset of mobile data traffic collected at 470 eNodeBs of a real-world network serving a large metropolitan region in Europe. The measurement data concerns a set of five popular and heterogeneous mobile services, namely Youtube, Facebook, Instagram, Snapchat, and iTunes, whose traffic flows were classified by the operator using proprietary Deep Packet Inspection (DPI) techniques.

We assume that each mobile service is associated to one dedicated slice in $\mathbb{S}$, and investigate the anticipatory allocation of resources at a single datacenter that runs virtualized core network functions over the mobile data traffic generated in the city under study. To this end, we train the AZTEC framework on eight weeks of data, use two additional weeks for validation, and finally run experiments on another two weeks. The three time periods do not overlap, and all results refer to the test phase only. Upon extensive appraisal of the system performance, we have observed that the DNNs in AZTEC best operate with data from the previous $N = 6$ intervals of duration T_s, hence we employ a $[6 \times 47 \times 10 \times 5]$ 4D tensor input. Unless stated otherwise, our default settings are $\kappa_o = \kappa_s = \kappa_i = 1$ and $\kappa_r = 0.5$, so as to account for the typically relatively lower cost per Mbps of resource reconfiguration; also, we set $T_s = 5$ minutes and $T_l = 30$ minutes to align with the capabilities of current Virtual Infrastructure Managers (VIM) [30]. In all our tests, AZTEC returned capacity allocations within one second, which is suitable for real-time operation in practical systems.

A. Harnessing the forecast uncertainty

As presented in Section III, we leverage a recent result on the approximation of uncertainty in DNN to include a safety margin in the model forecast. As a preliminary step in our evaluation of AZTEC, we investigate the impact of including the knowledge of the estimated uncertainty in the forecast produced by the different DNNs that are part of the framework.

Figure 4 visually shows the benefit of this design choice, by juxtaposing the performance of AZTEC with and without uncertainty; in the second case, dropout layers are deactivated during test, and all DNNs produce a single-value output. In each plot, we report the anticipatory allocation of capacities $x_i^d(t)$ and $x_i^s(t)$ to the target slice i, as well as that of the total shared capacity $x^s(t)$; the actual demand is on the background.

The plots illustrate well how the time-varying resource allocation achieved by AZTEC nicely follows the fluctuations of the slice traffic. More interestingly, AZTEC (in the top plot) achieves a more reliable assignment of slice resources

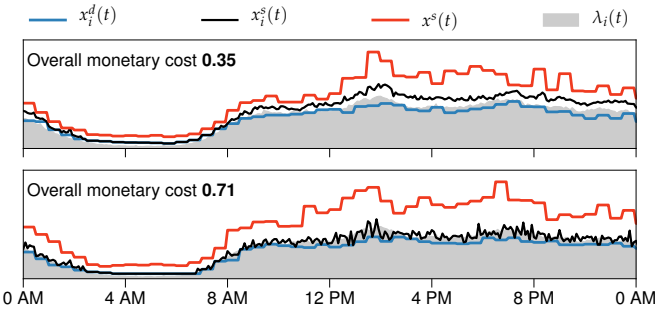

Fig. 4. Time series of sample resource allocations. Top: AZTEC framework. Bottom: equivalent framework where no uncertainty estimates are used.

by accounting for the level of uncertainty of the predictions. The total capacity $x_i^d(t) + x_i^s(t)$ is smoother and avoids situations where the demand cannot be serviced. Conversely, the framework not accounting for uncertainties (in the bottom plot) yields a capacity allocation that is noisy and incurs in substantial SLA violations due to unsatisfied demands.

In addition, AZTEC achieves such a result while saving on the amount of allocated resources (note the lower $x^s(t)$ curve), which ultimately results in an overall monetary cost that is half of that incurred by the framework without uncertainties. While this is an excerpt from a specific test, we recorded similar gains for all different settings explored in our analysis.

B. Comparative evaluation

We next assess the performance of AZTEC against two recent benchmarks: INFOCOM17, a state-of-the-art mobile network traffic predictor [21], and INFOCOM19, a fresh capacity forecasting model for network resource allocation [19]; both solutions are based on custom-built DNNs. INFOCOM17 is a traditional demand predictor, agnostic of all resource management costs, whereas INFOCOM19 takes anticipatory decisions on capacity allocation that aim exclusively at minimizing the trade-off A of overprovisioning and non-serviced demands.

Figure 5a summarizes the result of the comparative evaluation, showing the overall normalized[7] monetary cost of the anticipatory resource management against reconfiguration prices spanning two orders of magnitude. The gain of AZTEC over the benchmarks is clear, as even a state-of-the-art capacity predictor like INFOCOM19 increments the cost for the operator by a factor that ranges from 1.7 to beyond 10. As expected, the benchmark solutions inherently suffer more when reconfiguration costs –which they neglect– grow; however, even in a situation favorable to reconfiguration-agnostic approaches where such costs are small (*e.g.*, for $\kappa_r = 0.05$), AZTEC still yields a lower economic fee.

Finally, not only the relative performance is promising, but also the absolute (normalized) values show the potential advantage of a zero-touch network slicing paradigm. Indeed, by automatically and dynamically allocating capacity in advance,

[7]We normalize all results by the cost of an optimal but completely static resource allocation that dimensions the dedicated capacity to the traffic peak of each slice during the test period. This optimal allocation only incurs into overprovisioning costs, computed as $\sum_{i \in \mathbb{S}} \sum_t f_1 \left(\max_t \{\lambda_i(t)\} - \lambda_i(t) \right)$, which represents the normalization term.

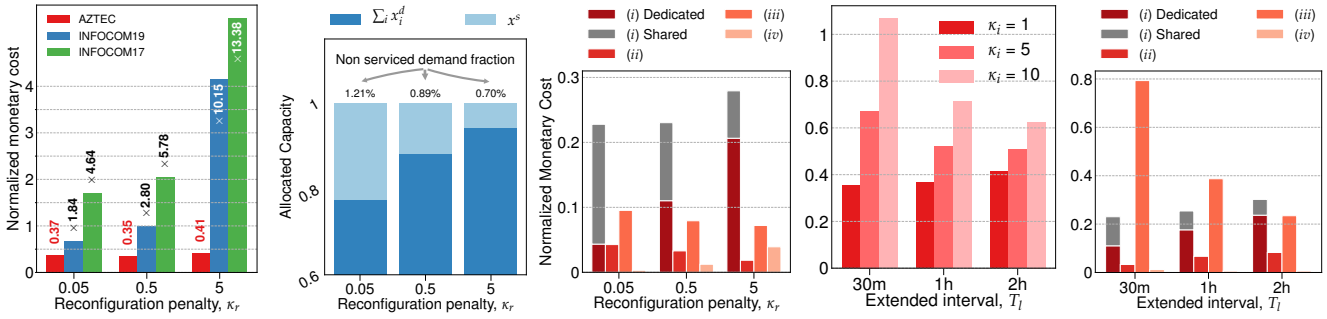

Fig. 5. Evaluation results. (a) Normalized monetary cost of AZTEC and two benchmarks versus the reconfiguration cost scaling factor κ_r. Numbers denote the exact cost for AZTEC, and the added cost factor for the benchmarks. (b) Total dedicated capacity $\sum_{i\in\mathbb{S}} x_i^d(t)$ and shared capacity $x^s(t)$ allocated by AZTEC versus κ_r. Numbers denote the fraction of re-orchestration opportunities with insufficient allocated resources. (c) Breakdown of the normalized monetary cost by penalty type. Tags refer to cost definitions in Section II-A, with overprovisioning cost (i) separated into contributions of the dedicated capacity, *i.e.*, the first term of (1), and of the shared capacity, *i.e.*, the second and third terms of (1). (d) Normalized monetary cost vs the duration of the extended re-orchestration interval T_l, for different scaling factors κ_i of the resource instantiation cost. (e) Breakdown of the normalized monetary cost by penalty type, for $\kappa_i = 10$.

AZTEC can cut management costs down to 35-41% of those incurred with an optimal static provisioning of resources.

C. Monetary cost breakdown

The above results prove that AZTEC maintains the combined costs described in Section II-A under control across a wide range of reconfiguration penalties, by properly adapting the portion of capacity allocated as dedicated, *i.e.*, $x_i^d(t)$, and as shared, *i.e.*, $x^s(i)_t$, to each slice i. To gain further insights on this, Figure 5b illustrates the capacity breakdown for tests in Figure 5a: a wider fraction of traffic is allocated to the more flexible shared capacity when the reconfiguration fee is low; instead, moving traffic to the dedicated capacity becomes more cost-efficient as κ_r grows. An important remark is that in all cases AZTEC incurs into SLA violations due to insufficient available resources in 0.7-1.21% of the re-orchestration opportunities: as a term of comparison, INFOCOM19 causes violations in at least 5.80% of the system observation time.

The ductility of the AZTEC orchestration results in a relative contribution of each cost source that stays fairly constant for different values of κ_r, as shown in Figure 5c. Here, the most notable trend is in the split of the overprovisioning cost, which grows for the dedicated capacity and decreases for the shared capacity. Indeed, and consistently with Figure 5b, provisioning dedicated resources in excess becomes more convenient than reconfiguring shared resources as κ_r becomes larger.

Overall, these results demonstrate the capability of AZTEC to properly solve the multiple trade-offs among resource management costs outlined in Section II-B, helping the operator to drastically reduce operation expenses in an automated way.

D. Controlling resource instantiation costs

Although AZTEC does not take autonomous decisions on trade-offs involving the instantiation of resources, it still offers direct control on the associated cost (iii) by means of the T_l parameter. Indeed, as operators incur into this type of fee once every extended re-orchestration interval, making T_l longer allows limiting the penalty. In Figure 5d we investigate the effectiveness of such a lever to control the instantiation cost, where T_l varies from 30 minutes to 2 hours.

The results show that T_l has an impact on the total cost when it is actually needed. In other words, when the scaling factor κ_i is small, the influence of the resource instantiation penalty on the total cost is negligible, and varying T_l has little effect. However, as κ_i grows, the duration of the extended interval becomes a functional handle to control the (now substantial) resource instantiation fee: increasing T_l from 30 minutes to 2 hours can reduce the overall cost by 40% when $\kappa_i = 10$.

Further detail is provided in Figure 5e, where the contribution of the different management costs are told apart, for $\kappa_i = 10$. We observe that, as expected, the cost of (i) overprovisioning and (ii) non-serviced demands grow with T_l. Indeed, if capacities $x_i^d(t)$ and $x^s(t)$ remain fixed over a longer interval, this limits the flexibility of the orchestrator to follow fluctuations in the demand, and forces a more challenging forecast over a longer time horizon. However, and more importantly, longer T_l have a clear positive impact on the instantiation cost, which they can reduce by a factor 4 in this specific case study. Ultimately, these results demonstrate the effectiveness of AZTEC in offering the operator with a means to control resource instantiation costs.

V. Conclusions

In this paper we designed AZTEC, a practical multi-timescale orchestration approach for network slicing. AZTEC combines deep learning tools with classic optimization algorithms to provide a zero-touch anticipatory capacity forecasting for individual slices. By separating the long-term assignment of slice-dedicated resources from the short-term re-orchestration of shared resources, it manages to (i) minimize resource instantiation costs, while (ii) ensuring that the service demands of all slices always are met via a timely reconfiguration of shared resources. Evaluation results on extensive real-world data show that AZTEC yield significantly gains over state-of-the-art solutions, and provides operators with a fine-level, automated control on the management of slice resources.

Acknowledgments

The work of University Carlos III of Madrid was supported by H2020 5G-TOURS project (grant agreement no. 856950).

The work of NEC Laboratories Europe was supported by H2020 5GROWTH project (grant agreement no. 856709). The research of M. Fiore was partially supported by ANR CANCAN project (ANR-18-CE25-0011).

REFERENCES

[1] AppAnnie, *The State of Mobile 2019*, Accessed: 2019-07-26. [Online]. Available: https://www.appannie.com/en/go/state-of-mobile-2019/

[2] M. Giordani, M. Polese, M. Mezzavilla, S. Rangan, and M. Zorzi, "Towards 6G Networks: Use Cases and Technologies," Mar. 2019, arXiv:1903.12216.

[3] P. Rost, C. Mannweiler, D. S. Michalopoulos, C. Sartori, V. Sciancalepore, N. Sastry, O. Holland, S. Tayade, B. Han, D. Bega, D. Aziz, and H. Bakker, "Network Slicing to Enable Scalability and Flexibility in 5G Mobile Networks," *IEEE Communications Magazine*, vol. 55, no. 5, pp. 72–79, May 2017.

[4] P. Caballero, A. Banchs, G. de Veciana, and X. Costa-Pérez, "Network slicing games: Enabling customization in multi-tenant networks," in *Proc. of IEEE Conference on Computer Communications (IEEE INFOCOM)*, Atlanta, GA, USA, May 2017, pp. 1–9.

[5] C. Marquez, M. Gramaglia, M. Fiore, A. Banchs, and X. Costa-Pérez, "How Should I Slice My Network?: A Multi-Service Empirical Evaluation of Resource Sharing Efficiency," in *Proc. of the 24th Annual International Conference on Mobile Computing and Networking (ACM MobiCom)*, New Delhi, India, Oct. 2018, pp. 191–206.

[6] V. Sciancalepore, C. Mannweiler, F. Z. Yousaf, P. Serrano, M. Gramaglia, J. Bradford, and I. L. Pavón, "A Future-Proof Architecture for Management and Orchestration of Multi-Domain NextGen Networks," *IEEE Access*, vol. 7, pp. 79 216–79 232, Jun. 2019.

[7] ETSI, *Open Source MANO (OSM) Project*, Accessed: 2019-07-26. [Online]. Available: https://osm.etsi.org/

[8] LFN, Linux Foundation, *Open Network Automation Platform*, accessed: 2019-07-26. [Online]. Available: https://www.onap.org/

[9] C. Zhang, P. Patras, and H. Haddadi, "Deep learning in mobile and wireless networking: A survey," *IEEE Communications Surveys Tutorials*, vol. 21, no. 3, pp. 2224–2287, Mar. 2019.

[10] B. Koley, "The zero touch network," in *Proc. of the 12th International Conference on Network and Service Management (IEEE CNSM)*, Montreal, Quebec, Canada, Oct. 2016.

[11] European Telecommunications Standards Institute (ETSI), "ZSM Scenarios and key requirements," ETSI ISG ZSM 001, Oct. 2018.

[12] D. Bega, M. Gramaglia, M. Fiore, A. Banchs, and X. Costa-Perez, "DeepCog: Optimizing Resource Provisioning in Network Slicing with AI-based Capacity Forecasting," *IEEE Journal on Selected Areas in Communications*, pp. 1–1, 2019.

[13] A. Garcia-Saavedra, X. Costa-Pérez, D. J. Leith, and G. Iosifidis, "FluidRAN: Optimized vRAN/MEC Orchestration," in *Proc. of IEEE International Conference on Computer Communications (IEEE INFOCOM)*, Honolulu, HI, USA, Apr. 2018, pp. 2366–2374.

[14] J. Kim, D. Kim, and S. Choi, "3GPP SA2 architecture and functions for 5G mobile communication system," *ICT Express*, vol. 3, no. 1, pp. 1–8, Mar. 2017.

[15] L. Peterson, A. Al-Shabibi, T. Anshutz, S. Baker, A. Bavier, S. Das, J. Hart, G. Palukar, and W. Snow, "Central office re-architected as a datacenter," *IEEE Communication Magazine*, vol. 54, no. 10, pp. 96–101, Oct. 2016.

[16] M. Mao and M. Humphrey, "A Performance Study on the VM Startup Time in the Cloud," in *Proc. of IEEE Fifth International Conference on Cloud Computing (IEEE CLOUD)*, Honolulu, HI, USA, Jun. 2012, pp. 423–430.

[17] 5G-CORAL, "Refined design of 5G-CORAL orchestration and control system and future directions," Public Deliverable, D3.2, May 2019.

[18] S. González, A. De la Oliva, C. J. Bernardos, and L. M. Contreras, "Towards a Resilient Openflow Channel Through MPTCP," in *Proc. of IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (IEEE BSMB)*, Valencia, Spain, Jun. 2018, pp. 1–5.

[19] D. Bega, M. Gramaglia, M. Fiore, A. Banchs, and X. Costa-Perez, "Deepcog: Cognitive network management in sliced 5g networks with deep learning," in *Proc. of IEEE International Conference on Computer Communications (IEEE INFOCOM)*, Paris, France, Apr. 2019, pp. 280–288.

[20] J. X. Salvat, L. Zanzi, A. Garcia-Saavedra, V. Sciancalepore, and X. Costa-Pérez, "Overbooking Network Slices Through Yield-driven End-to-end Orchestration," in *Proc. of the 14th International Conference on emerging Networking EXperiments and Technologies (ACM CoNEXT)*, Heraklion, Greece, Dec. 2018, pp. 353–365.

[21] J. Wang, J. Tang, Z. Xu, Y. Wang, G. Xue, X. Zhang, and D. Yang, "Spatiotemporal modeling and prediction in cellular networks: A big data enabled deep learning approach," in *Proc. of IEEE International Conference on Computer Communications (IEEE INFOCOM)*, Atlanta, GA, USA, May 2017, pp. 1–9.

[22] L. Zanzi, V. Sciancalepore, A. Garcia-Saavedra, and X. Costa-Pérez, "OVNES: Demonstrating 5G network slicing overbooking on real deployments," in *Proc. of IEEE International Conference on Computer Communications Workshops (IEEE INFOCOM WKSHPS)*, Honolulu, HI, USA, Apr. 2018, pp. 1–2.

[23] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," Dec. 2014, arXiv:1409.1556.

[24] C. Zhang, M. Fiore, and P. Patras, "Multi-Service Mobile Traffic Forecasting via Convolutional Long Short-Term Memories," in *Proc. of IEEE International Symposium on Measurements and Networking (IEEE M&N)*, Catania, Italy, Jun. 2019, pp. 1–6.

[25] G. E. Dahl, T. N. Sainath, and G. E. Hinton, "Improving deep neural networks for LVCSR using rectified linear units and dropout," in *Proc. of IEEE International Conference on Acoustics, Speech and Signal Processing (IEEE ICASSP)*, Vancouver, BC, Canada, May 2013, pp. 8609–8613.

[26] K. Janocha and W. M. Czarnecki, "On Loss Functions for Deep Neural Networks in Classification," *Schedae Informaticae*, vol. 25, pp. 49–59, Mar. 2017.

[27] Y. Gal and Z. Ghahramani, "Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning," in *Proc. of The 33rd International Conference on Machine Learning (ICML)*, New York, NY, USA, Jun. 2016, pp. 1050–1059.

[28] M. J. Powell, "The BOBYQA algorithm for bound constrained optimization without derivatives," *Cambridge Technical Report NA2009/06, University of Cambridge*, pp. 26–46, Aug. 2009.

[29] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes in FORTRAN 77: The Art of Scientific Computing*. New York, NY, USA: Cambridge University Press, 1993, vol. 2.

[30] J. G. Herrera and J. F. Botero, "Resource Allocation in NFV: A Comprehensive Survey," *IEEE Transactions on Network and Service Management*, vol. 13, no. 3, pp. 518–532, Sep. 2016.