A resource allocation framework for network slicing

Mathieu Leconte, Georgios Paschos, Panayotis Mertikopoulos, Ulas Kozat

A Resource Allocation Framework for Network Slicing

Mathieu Leconte Georgios S. Paschos Panayotis Mertikopoulos Ulaş C. Kozat

***Abstract*—Telecommunication networks are converging to a massively distributed cloud infrastructure interconnected with software defined networks. In the envisioned architecture, services will be deployed flexibly and quickly as *network slices*. Our paper addresses a major bottleneck in this context, namely the challenge of computing the best resource provisioning for network slices in a robust and efficient manner. With tractability in mind, we propose a novel optimization framework which allows fine-grained resource allocation for slices both in terms of network bandwidth and cloud processing. The slices can be further provisioned and auto-scaled optimally based on a large class of utility functions in real-time. Furthermore, by tuning a slice-specific parameter, system designers can trade off traffic-fairness with computing-fairness to provide a mixed fairness strategy. We also propose an iterative algorithm based on the alternating direction method of multipliers (ADMM) that provably converges to the optimal resource allocation and we demonstrate the method's fast convergence in a wide range of quasi-stationary and dynamic settings.**

I. Introduction

The infrastructure of telecommunication networks is undergoing a radical transformation towards a massively distributed cloud with network functionalities implemented as virtual network functions (VNFs). When this transition is complete, services will be instantiated as *network slices* – i.e. virtual networks utilizing transport and computing resources in order to provide high-fidelity communications [1]. Unlike their physical counterparts, VNFs can be launched, placed, and scaled flexibly to meet fluctuating workload demands, thus opening new doorways for on-demand resource provisioning. However, quick and accurate resource allocation for multiple network slices is a formidable task, primarily due to the following reasons:

1) Each network slice requires specific functionalities, possibly executed in a particular order.
2) The operation of a network slice involves (and requires the close coordination of) several stakeholders: *a*) *slice owners*; *b*) the *cloud provider*; and *c*) the *network controller*.
3) Each slice requires two types of resources, bandwidth and processing power. As a result, resource allocation must be balanced between these two components, and notions of fairness and efficiency become much more intricate.

From a modeling standpoint, network slices can be represented as collections of interconnected VNFs. So far, the most common approach in the literature is to treat a VNF as the unit of computing. Thus, to provision resources for network slices, considerable effort has been dedicated towards the problem of VNF placement and routing [2–8]. This problem and its variations are NP-hard, so the resulting resource allocation techniques are cumbersome and inaccurate for large networks. By this token, providing an efficient framework for allocating resources to network slices remains a highly relevant problem.

M. Leconte and G. S. Paschos are with with Huawei Technologies, France.
P. Mertikopoulos is with Univ. Grenoble Alpes, CNRS, Grenoble INP, Inria, LIG, F-38000, Grenoble, France.
U. Kozat is with Huawei R&D, USA.
PM was partially supported by the Huawei HIRP project ULTRON.

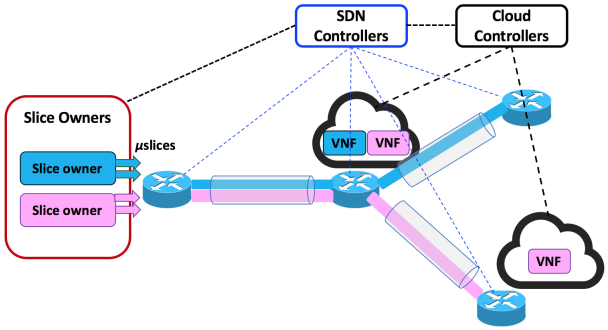

Fig. 1. System architecture under consideration.

To address this problem, we envision an architecture along the lines of the emerging *cloud-native* paradigm [9, 10]. According to its principles, VNFs are divided into smaller components, each of which is placed in a software container [11]. This finer granularity allows network functionality to be launched quickly anywhere in the cloud and simplifies load balancing across different execution instances [12]. We refer to these instances, collectively with their interconnections as *μslices* (micro-slices); therefore, in a cloud-native setting, network slices are composed of many μslices. Since the state of different μslices is decoupled from one another, the corresponding network slices can be scaled in and out without impacting ongoing services. Hence, the cloud-native framework allows us to focus on how to efficiently (re)distribute the transport and processing load of a network slice instead of focusing on how to individually place and interconnect each VNF.

A. Our contributions

The approach of *zooming-out* and viewing the big picture of resource allocation, has been very successful in legacy communication networks, cf. [13]. With this in mind, we propose here the first flexible and *lightweight* resource allocation framework for network slicing by leveraging the cloud-native architecture concepts and state-of-the-art methods from continuous optimization. We summarize our contributions:

- We introduce a novel resource allocation framework which allows us to provision and auto-scale slices in real time.
- We formulate the *network slice resource allocation problem* based on a tunable utility function subject to the network's bandwidth and the cloud's processing power capacities. Its solution allows the designer to drive the system to any operational trade-off between traffic-fairness and computing-fairness.

- We propose an iterative algorithm that provably solves the network slicing resource allocation problem above by leveraging the so-called *alternating direction method of multipliers* (ADMM). By design, the algorithm incorporates the natural roles of the three stakeholders shown in Fig. 1: *(i)* the *slice owners* (who admit user sessions and require the scaling of the slice service); *(ii)* the *network controller* (who provisions end-to-end paths interconnecting cloud locations and allocates bandwidth to network slices); and *(iii)* the *cloud controller* (who allocates processor resources to serve incoming network flows).
- We demonstrate the algorithm's very fast convergence, a critical desideratum for quasi-stationary and dynamic settings, e.g. when configuring new slices on-demand, or auto-scaling slices based on fluctuating workloads, etc.

B. Related work

Prior work on resource allocation for network slicing focuses on the joint *routing and VNF placement problem*, which decides how to place VNFs on the physical network and which routes to use to interconnect them [3]. This problem can be cast as a minimum cost *Virtual Network Embedding* (VNE) problem, which is known to be NP-hard [2]. An interesting line of research focuses on adding practical constraints [5–8] but the resulting generalizations are also NP-hard, so these approaches become cumbersome if applied to multiple slices or large physical networks.

The fractional counterpart of the integral routing and VNF placement problem is studied in [14] where it is formulated as a multicommodity-chain and a linear utility is minimized via a dual stochastic subgradient algorithm [15]. This approach is similar to backpressure methods where queueing is used to relax the capacity constraints. A similar approach was also used in [16] in a different context; however, despite being fully distributed, dual ascent methods are often very slow in practice due to the high coordination overhead required between network nodes and links. [17] views the problem as a congestion mitigation game between virtual network operators and evaluates the price of anarchy. By contrast, modern software-defined networking (SDN) and cloud controllers have a centralized resource view which means that full distributedness is not necessary; our ADMM-based solution scheme leverages precisely this intermediate degree of centralization, thus providing significant performance gains in convergence time.

In our framework we include fairness considerations as those encountered in the theory of joint routing and congestion control [13, 18]. In the context of network slicing, we generalize these concepts to the case of two resources: network bandwidth and cloud processing power. This provides an elastic framework for slice autoscaling and resource sharing, with a significant degree of flexibility in balancing these two resources.

II. System Model

In this section, we describe our system model and the basics of the proposed resource allocation framework. To begin with, we represent the underlying physical infrastructure as a directed multi-graph $\mathcal{G} = (\mathcal{N}, \mathcal{L})$ with (i) link bandwidth $b_\ell > 0$ per link $\ell \in \mathcal{L}$ (measured in bits/s); and (ii) processing power π_n per node $n \in \mathcal{N}$ (measured in flops/s). A node with a positive processing power represents a cloud location with capability to execute VNFs, while a node with zero processing power serves only as an SDN router.

A. Slice model

We consider the co-existence of multiple slices on the same physical network. For clarity and simplicity, we focus on scenarios where each slice serves network traffic from a single source to a single destination, and hence we represent a slice by a source-destination (S/D) pair. Slices with multiple S/D pairs require an extension of the model, discussed in Section VI.

The definition of each slice includes a VNF forwarding graph, which determines the sequence of execution of the different VNFs. In this paper we consider *simple VNF chains*, in which case the forwarding graph is a directed line network. Extending to directed acyclic graphs (DAGs) is straightforward by applying the flow decomposition theorem [19, Ch. 3] to break the DAG into a set of paths, and then consider the set of paths as one slice with multiple S/D pairs. Furthermore, we collapse the entire forwarding graph (involving multiple VNFs) to one amalgamated VNF. Notice, that under the assumption that the VNF consists of multiple fine-grained and decoupled instances (so it can be executed in fractions at multiple cloud locations), this collapsing is done without loss of generality in the model. We mention the similarity to the "big switch" abstraction used for the network fabric in SDN. In what follows, each slice will consist of a S/D pair and one VNF.

B. Routing and processing

Indexing each slice by $s \in \mathcal{S} = \{1, \dots, S\}$, we will write x_s for the traffic volume it generates, and y_s for the computing power required to process it. We further posit that the traffic of slice s can be split among a set $\mathcal{P}_s$ of paths joining its source to its destination, and we write x_{sp} for the amount of traffic belonging to slice s and going through path $p \in \mathcal{P}_s$. Thus, by definition, we have

$$x_s = \sum\nolimits_{p \in \mathcal{P}_s} x_{sp} \quad \text{for all } s \in \mathcal{S}. \tag{1}$$

In terms of computing, the processing load of each slice can be split among each of the nodes it is routed through. To model this, let y_{sn} be the amount of traffic belonging to slice s and processed at node n, so $y_s = \sum_{n \in \mathcal{N}} y_{sn}$ (with the understanding that $y_{sn} = 0$ by default if slice s does not pass through node n). This load can be decomposed further by considering the load y_{snp} induced by slice s on node n and originating from path p (with the analogous convention that $y_{snp} = 0$ if p does not go through n). By definition, we have $y_{sn} = \sum_{p \in \mathcal{P}_s} y_{snp}$ and hence

$$y_s = \sum\nolimits_{n \in \mathcal{N}} y_{sn} = \sum\nolimits_{n \in \mathcal{N}} \sum\nolimits_{p \in \mathcal{P}_s} y_{snp}. \tag{2}$$

Our basic assumption relating the traffic volume of a slice and the required computing resources to process it is that each unit of traffic requires w_s units of computing. This linear coupling of the two resources is discussed in the Internet draft for VNF chaining resource management [20] and is commonly

observed in practice – for instance, in cloud routers [21]. Importantly, the traffic-to-processing ratio w_s may be slice-dependent: in particular, we have $w_s = 0$ if slice s does not require any processing, while w_s could be very large if slice s requires intensive processing for a relatively small amount of traffic. The main premise is that the two core network services (packet routing versus function processing) are interrelated and their relation is fixed per slice.

With all this in mind, we obtain the basic relations

$$y_{sp} = w_s x_{sp} \quad \text{and} \quad y_s = w_s x_s. \tag{3}$$

Motivated by the above, we introduce a new set of *routing allocation variables* z_{snp} to denote the amount of *traffic* of the s-th slice which is routed through path $p \in \mathcal{P}_s$ and is processed at node n. This set of variables will be particularly useful to our analysis because all of the previous variables can be described in terms of z_{snp} via the elementary relations

$$x_{sp} = \sum\nolimits_{n\in\mathcal{N}} z_{snp} \quad \text{and} \quad x_s = \sum\nolimits_{n\in\mathcal{N}} \sum\nolimits_{p\in\mathcal{P}_s} z_{snp}, \tag{4a}$$

$$y_{snp} = w_s z_{snp} \quad \text{and} \quad y_s = w_s \sum\nolimits_{n\in\mathcal{N}} \sum\nolimits_{p\in\mathcal{P}_s} z_{snp}, \tag{4b}$$

again assuming that $z_{snp} = 0$ if p does not pass through n.

Remark. Suppose that the routing allocation profile $\mathbf{z} = (z_{snp})_{s\in\mathcal{S},n\in\mathcal{N},p\in\mathcal{P}_s}$ satisfies Eqs. (4a) and (4b) and each slice s is composed of many smaller μslices of the same type but small demand. Then, applying randomized rounding to $\mathbf{z}$ [22], we obtain a feasible integral assignment (path and VNF placement) for each μslice. Hence, even though we will be considering a continuous model which is amenable to efficient algorithms, the above ensures that we remain true to the practical system constraints of μslice networks.

C. Capacity constraints and feasibility

Since multiple slices share the same physical network, the bandwidth consumption at each link may not exceed the available link bandwidth. Thus, letting $z_\ell = \sum_s \sum_{n\in p} \sum_{p\ni\ell} z_{snp}$ denote the amount of traffic being routed through link $\ell \in \mathcal{L}$, we get the *link bandwidth constraints:*

$$z_\ell = \sum\nolimits_{s\in\mathcal{S}} \sum\nolimits_{p\ni\ell} \sum\nolimits_{n\in p} z_{snp} \le b_\ell \quad \text{for all } \ell \in \mathcal{L}. \tag{5}$$

Likewise, letting $y_n = \sum_s y_{sn} = \sum_s \sum_{p\ni n} y_{snp}$ denote the total processing load at node n, and given that said load cannot exceed the node's processing capacity, we also have the *node processing constraints:*

$$y_n = \sum\nolimits_{s\in\mathcal{S}} \sum\nolimits_{p\ni n} w_s z_{snp} \le \pi_n \quad \text{for all } n \in \mathcal{N}. \tag{6}$$

In what follows, we will refer to b_ℓ and π_n as *resources* (provided by the network), and to $x_s = \sum_{n\in p} \sum_{p\in\mathcal{P}_s} z_{snp}$ and $y_s = w_s \sum_{n\in p} \sum_{p\in\mathcal{P}_s} y_{snp}$ as *services* (demanded from the network by slice s). In this way, aggregating over all slices, the profile of *decision variables* $\mathbf{z} = (z_{snp})_{s\in\mathcal{S},n\in\mathcal{N},p\in\mathcal{P}_s}$ captures the service demands of each slice (in terms of both bandwidth and processing), leading to the *feasible region*

$$\mathcal{Z} = \left\{\mathbf{z} \in \prod\nolimits_{s\in\mathcal{S}} \prod\nolimits_{n\in\mathcal{N}} \mathbb{R}_+^{\mathcal{P}_s} : \text{Eqs. (5) and (6) hold}\right\}. \tag{7}$$

Since the constraints Eqs. (5) and (6) that define $\mathcal{Z}$ are linear and no routing allocation variable z_{snp} can grow without bound, it follows readily that $\mathcal{Z}$ is a compact convex polytope. In what follows, we will use this property of $\mathcal{Z}$ freely.

III. Fairness and Service Balancing

To orchestrate the simultaneous operation of multiple slices, the routing allocation profile $\mathbf{z}$ must balance the received service for each slice $s \in \mathcal{S}$. Our goal in this section is to provide a general framework to optimize the allocation of the network's resources in this context.

A. Slice viewpoint: Pareto efficiency

To begin with, at the slice level, the primary objective is to scale up the service of each slice $s \in \mathcal{S}$ until the network resources are saturated (both in terms of bandwidth x_s and processing power $y_s = w_s x_s$). Since the demand x_s depends linearly on the routing allocation variables z_{snp} (via Eqs. (4a) and (4b) respectively), this means that solutions will lie at the boundary of the feasible region $\mathcal{Z}$. We formalize this via the well-known concept of *Pareto efficiency* below:

Definition 1. We say that a routing allocation profile $\mathbf{z} \in \mathcal{Z}$ is *Pareto efficient* if, for any $\mathbf{z}' \in \mathcal{Z}$, we have

$$x'_s > x_s \text{ for some } s \in \mathcal{S} \Rightarrow x'_{s'} < x_{s'} \text{ for some } s' \in \mathcal{S}. \tag{8}$$

where x_s and x'_s denote the total traffic demand of slice s in the profiles $\mathbf{z}$ and $\mathbf{z}'$ respectively, as induced by (4a). In words, the routing allocation profile $\mathbf{z}$ is Pareto efficient if it is impossible to improve the bandwidth service of a slice without penalizing the bandwidth service of another.

Remark. As $y_s = w_s x_s$, a Pareto efficient profile $\mathbf{z}$ is also Pareto efficient with respect to processing demands (so it does not matter if (8) is formulated in terms of x or y). However, since the *constraints* for bandwidth and processing are different (cf. Eqs. (5) and (6) above), it is possible that an increase in the *bandwidth* demand x_s could violate the node processing constraints (6) because of the commensurate increase of the computing demand $y_s = w_s x_s$ (and, conversely, increasing y_s might end up violating the link bandwidth constraints). These interrelations are reflected in the definition of the feasible region $\mathcal{Z}$, hence the definition of Pareto efficiency in terms of the routing allocation variables $\mathbf{z}$.

B. Network viewpoint: Fairness of service

In view of the above, our primary efficiency requirement at the slice level will be to achieve a resource allocation scheme that is Pareto efficient. However, an important efficiency aspect that is not taken into account by the definition of Pareto efficiency is that increased service typically exhibits *diminishing returns*: for instance, an increase from 1 bit/s to 2 bits/s is often much more beneficial than an increase from 1001 bits/s to 1002 bits/s. This becomes especially relevant in realistic network conditions where, to avoid slice starvation, we also need to ensure that service is distributed among slices in a fair manner (similar to legacy TCP congestion control mechanisms for packet flows). As such, the following key question arises: *is it possible to capture all of the above objectives in a flexible, network-wide utility function?*

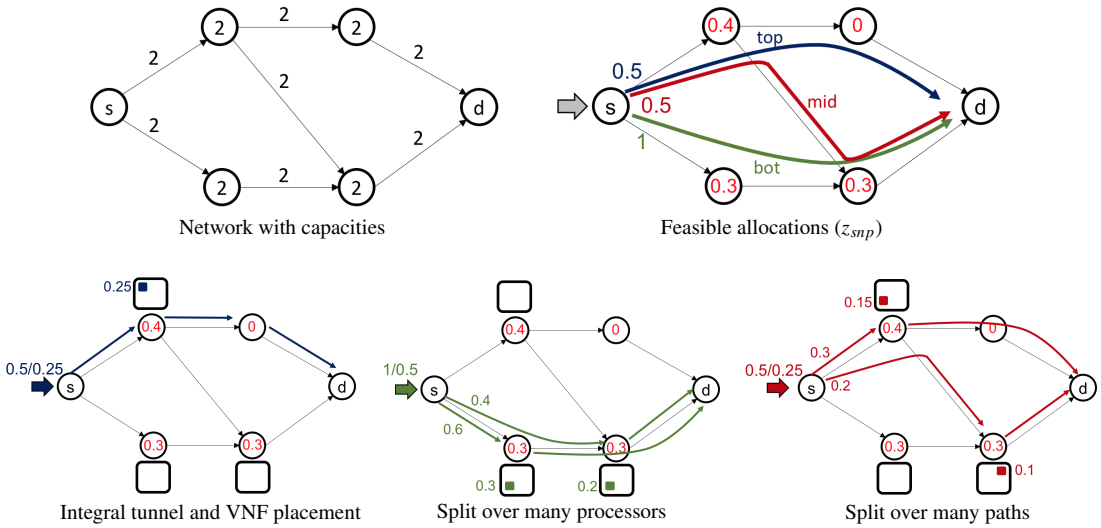

Fig. 2. Bandwidth and processing allocation for μslices: each link has bandwidth $b_\ell = 2$ bits/s and each node provides processing $\pi_n = 2$ flops/s, and we want to provision a slice with traffic 2 bits/s and computing 1 flop/s (so $w_s = 1/2$). In the top-right corner of Fig. 2, we depict a feasible resource allocation profile that splits the traffic of the slice over three paths with routed volume 0.5, 0.5 and 1 bits/s respectively. In turn, this means that the top path incurs a processing load of 0.25 flops/s, the middle one incurs 0.25 flops/s, and the bottom 0.5 flops/s. The bottom part of the figure shows integral allocations for μslices. Each integral allocation involves a single path and VNF placement and there is a total of 5 possible integral allocations shown (one blue, two green and two red).

To do so, we will focus on the *weighted α-fairness problem*

$$\text{maximize} \quad g(\mathbf{x}) = \sum\nolimits_{s\in\mathcal{S}} U_\alpha(\theta_s x_s), \tag{9}$$

where $\theta_s \geq 0$ is a slice-specific *balancing parameter* and the *α-fair utility model* $U_\alpha(\cdot)$ is defined for $\alpha \in [0,\infty)$ as

$$U_\alpha(\xi) = \begin{cases} \frac{\xi^{1-\alpha}}{1-\alpha} & \text{if } \alpha \geq 0,\ \alpha \neq 1, \\ \log \xi & \text{if } \alpha = 1. \end{cases} \tag{10}$$

Before illustrating the relation between the optimization problem (9) and (8), we briefly discuss the role of the fairness parameter α and the balancing parameter θ.

We begin with the α-fairness model (9). Letting $\xi_s = \theta_s x_s$ denote an abstract service demand by slice s,[1] we see that different values of α lead to different fairness criteria for the service demand profile $\boldsymbol{\xi} = (\xi_s)_{s\in\mathcal{S}}$. Specifically, as α ranges from 0 to ∞, we have the following cases of special interest:

- $\alpha = 0$: sum-utility maximization
- $\alpha = 1$: proportionally fair allocation [23],
- $\alpha = 2$: potential delay fair allocation [13],
- $\alpha \to \infty$: max-min fair allocation [24].

In our setting however, there is an extra degree of complication because we need to balance two types of service demands per slice: traffic (x_s) and processing (y_s). Indeed, by choosing $\theta_s = w_s$, $\forall s \in \mathcal{S}$, and in light of $y_s = w_s x_s$, $\forall s \in \mathcal{S}$, (9) becomes

$$\text{maximize} \quad \sum\nolimits_{s\in\mathcal{S}} U_\alpha(y_s),$$

whose solution for different values of α corresponds to an α-fair allocation with respect to processing. For example, if $\alpha = 1$, the traffic-fair allocation ($\theta_s = 1$) will scale slices so that they occupy resources in a proportionally-fair manner with respect to traffic; instead, the computing-fair allocation ($\theta_s = w_s$) will scale them in a different manner so that computing is proportionally fair. This difference is highlighted in the example of Fig. 3 where we consider a toy network with one link, one node, and two slices and we showcase four examples depending on fairness orientation (traffic or computing), and the limiting resource (bandwidth or processing).

Clearly, both traffic and computing balancings are of practical interest, depending on the context: For example, in TCP flows with software firewalls, traffic is more important; however, for computation-intensive flows, computing power is much more important, so fairness should be established with respect to the latter. Intermediate cases are also relevant and can be obtained if the received service is valued by slice s as $U_\alpha(\beta_s x_s + (1-\beta_s) y_s)$ for some (slice-specific) mixing parameter $\beta_s \in [0,1]$. Obviously, if we change the value of β, the various sum-utility, proportional, and max-min fair solutions will yield different resource allocation profiles, reflecting the different way of balancing bandwidth and processing demands. In particular,

- If $\theta_s = 1$, $\forall s$, (9) describes traffic-fair balancing solutions.
- If $\theta_s = w_s$, $\forall s$, (9) yields computing-fair solutions.
- If $\theta_s = \beta_s + (1-\beta_s) w_s$, $\forall s$, (9) describes a mixed traffic-computing fair balancing.
- Finally, if $\theta_s = 1$ for some slices and $\theta_s = w_s$ for the rest, this describes networks with a combination of traffic-driven and processing-driven slices.

Now, to relate the weighted α-fairness optimization problem (9) to the notion of Pareto efficiency, we begin with the observation that solving (9) converges as $\alpha \to \infty$ to a weighted max-min allocation [24, 25]:

Definition 2. We say that a routing allocation profile $\mathbf{z} \in \mathcal{Z}$ is *weighted max-min fair with weights $\boldsymbol{\theta}$* if, for all $\mathbf{z}' \in \mathcal{Z}$, we have

$$\exists s \in \mathcal{S}, x'_s > x_s \implies \exists s \in \mathcal{S}, x'_{s'} < x_{s'} \text{ and } \theta_{s'} x_{s'} \leq \theta_s x_s, \tag{11}$$

[1] For instance, depending on the value of θ_s, we could have $\xi_s = x_s$ for $\theta_s = 1$ (bandwidth demand) or $\xi_s = y_s$ for $\theta_s = w_s$ (processing demand).

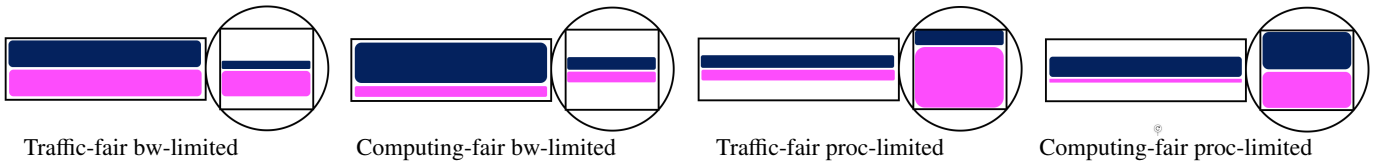

Fig. 3. Fair autoscaling of slices.

where x_s and x'_s denote the total traffic demand of slice s in the profiles $\mathbf{z}$ and $\mathbf{z}'$ respectively. We can improve a coordinate only by worsening a poorer (in weighted-sense) one.

Proposition 1. If $\mathbf{z} \in \mathcal{Z}$ is Pareto efficient, then it is also weighted max-min fair with weights

$$\theta_s = \begin{cases} 1/x_s & \text{if } x_s > 0, \\ \infty & \text{otherwise.} \end{cases} \tag{12}$$

Proof: Choose another profile $\mathbf{z}' \in \mathcal{Z}$ such that for some s we have $x'_s > x_s$, to prove the result it suffices to show (11) holds. Since $\mathbf{z} \in \mathcal{Z}$ is Pareto, it immediately follows from the definition that there must exist s' such that $x'_{s'} < x_{s'}$. Additionally, $x'_s > x_s$ also implies $x'_s > 0$ (since $x_s \geq 0$), so we obtain $\theta_{s'} x_{s'} = 1 \leq \theta_s x_s$ and our proof is complete. ∎

Thus, combined with the fact that solving (9) for $\alpha \to \infty$ gives a weighted max-min fair allocation, we conclude that Pareto-efficient routing solutions are just a limiting special case of the network-wide utility framework (9).

Summing up the above, we obtain the *general slice resource allocation problem:*

$$\begin{aligned}
\text{maximize} \quad & g(\mathbf{x}) = \sum\nolimits_{s\in\mathcal{S}} U_\alpha(\theta_s x_s), && \text{(Opt)} \\
\text{subject to} \quad & y_n \leq \pi_n \quad \text{for all } n \in \mathcal{N}, && \text{(13a)} \\
& z_\ell \leq b_\ell \quad \text{for all } \ell \in \mathcal{L}, && \text{(13b)} \\
& x_s = \sum\nolimits_{p\in\mathcal{P}_s} \sum\nolimits_{n\in p} z_{snp}, && \text{(13c)} \\
& y_n = \sum\nolimits_{s\in\mathcal{S}} \sum\nolimits_{p\ni n} w_s z_{snp}, && \text{(13d)} \\
& z_\ell = \sum\nolimits_{s\in\mathcal{S}} \sum\nolimits_{p\ni\ell} \sum\nolimits_{n\in p} z_{snp}. && \text{(13e)}
\end{aligned}$$

From a mathematical viewpoint, the true decision variables in (Opt) are the routing allocation variables $\mathbf{z} \in \mathcal{Z}$. However, the augmented formulation above is more transparent because it illustrates the interplay between service demands and network resources. Specifically, by increasing x_s, we increase traffic and computing and hence scale out the slice service s; instead, by decreasing x_s, we scale in. As such, the solution of (Opt) ensures that the slices are scaled in a fair and efficient manner.

IV. Distributed Algorithm for Slice Autoscaling

Even though the mathematical optimization problem (Opt) provides a concise and robust framework for "fair and balanced" resource allocation for network slices, it is not clear how such an optimum allocation could be achieved in practice. Our goal in this section will be to address two main challenges that arise in this context, namely:

1) How can (Opt) be solved in an efficient manner?
2) Is it possible to provide a distributed solution based on easy optimization problems for (*i*) the network's users; (*ii*) the network controller; and (*iii*) the cloud controller?

In [16], the authors propose a solution algorithm for a similar problem based on a dual subgradient method which relaxes both capacity constraints for processing and bandwidth (Eqs. (13a) and (13b) respectively). In our case however, the dimensionality of these constraints would render a dual subgradient approach very slow in practice, even for moderately-sized networks. Moreover, until a dual subgradient method converges (which could take a very large number of iterations), intermediate iterations would typically violate bandwidth and processing capacities (since the objective tends to saturate them), so cutting the algorithm short would exhaust the network's capacity both in terms of bandwidth and processing.

For this reason, we take a different approach and instead relax the *equality* constraints (13c)–(13d) in (Opt). Doing so allows us to design a provably convergent algorithm based on the *alternating direction method of multipliers* (ADMM), which ensures consistency by alternating the optimizing entities in (Opt). In the rest of this section we describe this construction.

A. Distributed ADMM scheme

As we discussed in Section III, the true decision variables in (Opt) are the routing allocation variables $\mathbf{z} = (z_{snp})_{s\in\mathcal{S},n\in\mathcal{N},p\in\mathcal{P}_s}$: the bandwidth and processing demand profiles ($\mathbf{x} = (x_s)_{s\in\mathcal{S}}$ and $\mathbf{y} = (y_n)_{n\in\mathcal{N}}$ respectively) could be dropped if we sought a more parsimonious representation of (Opt). However, in addition to providing conceptual clarity, the augmented form of (Opt) allows us to directly relax the equality constraints (13c)–(13d) by introducing the augmented Lagrangian:

$$\begin{aligned}
L_\rho(\mathbf{x},\mathbf{y},\mathbf{z},\boldsymbol{\lambda}) &= \sum\nolimits_{s\in\mathcal{S}} U_\alpha(\theta_s x_s) - \rho\langle\boldsymbol{\lambda},\mathbf{r}\rangle - \frac{\rho}{2}\|\mathbf{r}\|^2, \\
&= \sum\nolimits_{s\in\mathcal{S}} U_\alpha(\theta_s x_s) - \frac{\rho}{2}\|\mathbf{r}+\boldsymbol{\lambda}\|^2 + \frac{\rho}{2}\|\boldsymbol{\lambda}\|^2, \qquad (14)
\end{aligned}$$

where:

- $\rho > 0$ is a *penalty parameter*, whose role is to control the impact of the quadratic constraint penalization.[2]
- $\boldsymbol{\lambda} = (\lambda_s, \lambda_n)_{s\in\mathcal{S},n\in\mathcal{N}}$ is a vector of multipliers associated to the equality constraints (13c)–(13d).
- $\mathbf{r} = (r_s, r_n)_{s\in\mathcal{S},n\in\mathcal{N}}$ is the corresponding *residual vector*

$$r_s = x_s - \sum\nolimits_{p\in\mathcal{P}_s} \sum\nolimits_{n\in p} z_{snp}, \tag{15a}$$

$$r_n = y_n - \sum\nolimits_{s\in\mathcal{S}} \sum\nolimits_{p\ni n} w_s z_{snp}. \tag{15b}$$

- $\langle\cdot,\cdot\rangle$ and $\|\cdot\|$ denote the standard (Euclidean) inner product and norm respectively.

[2]Due to the form of the augmented Lagrangian, ρ essentially becomes the step size of the multiplier update.

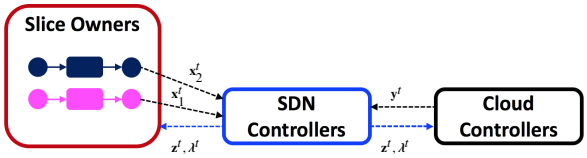

Fig. 4. Signal interfaces between network entities.

Then, by standard results in convex optimization [26], the optimum value of (Opt) can be expressed as

$$\max_{\substack{\mathbf{x},\mathbf{y},\mathbf{z}\geq 0\\ \text{s.t. (13b),(13a)}}} \min_{\boldsymbol{\lambda}} L_\rho(\mathbf{x},\mathbf{y},\mathbf{z},\boldsymbol{\lambda}). \tag{16}$$

Armed with this formulation, we may view the variables $\mathbf{x}$, $\mathbf{y}$, $\mathbf{z}$ and $\boldsymbol{\lambda}$ as belonging to different agents that can alternate turns to optimize each variable in a round-robin fashion, a much simpler task than solving the joint optimization. Motivated by this observation, we consider the ADMM-like recursion:

$$\mathbf{x}^{t+1} = \arg\max_{\mathbf{x}\geq 0} L_\rho(\mathbf{x},\mathbf{y}^t,\mathbf{z}^t,\boldsymbol{\lambda}^t), \tag{17a}$$

$$\mathbf{y}^{t+1} = \arg\max_{\substack{\mathbf{y}\geq 0\\ \text{s.t. Eq. (13a)}}} L_\rho(\mathbf{x}^{t+1},\mathbf{y},\mathbf{z}^t,\boldsymbol{\lambda}^t), \tag{17b}$$

$$\mathbf{z}^{t+1} = \arg\max_{\substack{\mathbf{z}\geq 0\\ \text{s.t. Eq. (13b)}}} L_\rho(\mathbf{x}^{t+1},\mathbf{y}^{t+1},\mathbf{z},\boldsymbol{\lambda}^t), \tag{17c}$$

$$\boldsymbol{\lambda}^{t+1} = \boldsymbol{\lambda}^t + \mathbf{r}^{t+1}, \tag{17d}$$

where $t = 0, 1, \ldots$ denotes the epoch of the algorithm.

Importantly, the specific alternation of variables in (17) is aligned with the decision makers in the original context of the problem. Specifically,

a) Eq. (17a) is the *slice owner problem* (solved over $\mathbf{x}$).
b) Eq. (17b) is the *cloud provider problem* (solved over $\mathbf{y}$).
c) Eq. (17c) is the *network provider problem* (solved over $\mathbf{z}$).
d) Eq. (17d) is the *dual variable update*.

With this in mind, we note that the slice owner and cloud problems above can be readily parallelized because there are no cross-terms involving x_s and y_n in (14). Also, the update of dual variables can all be done by the network provider (alternatively, each entity – slice owner or cloud provider – can compute its residual and update its corresponding dual variable).

B. Problem splicing and decomposition

In what follows, we discuss in detail how to splice the method over the various optimizing entities, viz. *a*) the slice owners; *b*) the network provider; and *c*) the cloud provider. The signaling interface between the various entities can be seen on Fig. 4.

a) The slice owner problem: At the slice level, since the objective and the constraints of (17a) are both separable in x_s, the problem can be distributed by asking each slice owner to solve the "congestion control" problem:

$$\underset{\text{s.t. } x_s\geq 0}{\text{maximize}} \quad U_\alpha(\theta_s x_s) - \frac{\rho}{2}\Big(x_s + \lambda_s^t - \sum\nolimits_{p\in\mathcal{P}_s}\sum\nolimits_{n\in p} z_{snp}^t\Big)^2 \tag{18}$$

Problem (18) is concave (as the objective is the sum of two concave functions). Since x_s is only constrained from below by 0, the solution of (18) at the $(t+1)$-th stage is the (unique, positive) solution of the stationarity equation

$$\theta_s^{1-\alpha} x_s^{-\alpha} = \rho\Big(x_s + \lambda_s^t - \sum\nolimits_{p\in\mathcal{P}_s}\sum\nolimits_{n\in p} z_{snp}^t\Big). \tag{19}$$

Algorithm 1 DSRA algorithm

Fix $\rho > 0$; set $t \leftarrow 0$; initialize $\mathbf{x}^0$, $\mathbf{y}^0$, $\mathbf{z}^0$, $\boldsymbol{\lambda}^0$.
repeat
  Determine $\mathbf{x}^{t+1}$ from (19); {Slice flow control}
  Determine $\mathbf{y}^{t+1}$ from (21); {Processing allocation}
  Determine $\mathbf{z}^{t+1}$ from (22); {Routing allocation}
  $\lambda_s^{t+1} \leftarrow \lambda_s^t + x_s^{t+1} - \sum_{p\in\mathcal{P}_s}\sum_{n\in p} z_{snp}^{t+1}$; {Slice price}
  $\lambda_n^{t+1} = \lambda_n^t + y_n^{t+1} - \sum_{s\in\mathcal{S}}\sum_{p\ni n} w_s z_{snp}^{t+1}$; {Node price}
  $t \leftarrow t+1$; {Next iteration}
until convergence
return allocation profile $(\mathbf{x}^t, \mathbf{y}^t, \mathbf{z}^t)$

The solution $\mathbf{x}^{t+1}$ is then fed back to the network and cloud providers (see below).

b) The cloud provider problem: At the $(t+1)$-th epoch, the cloud provider must solve the optimization problem:

$$\begin{aligned}\text{minimize} \quad & \sum\nolimits_{n\in\mathcal{N}}\Big(y_n + \lambda_n^t - \sum\nolimits_{s\in\mathcal{S}}\sum\nolimits_{p\ni n} w_s z_{snp}^t\Big)^2 \\ \text{subject to} \quad & 0 \leq y_n \leq \pi_n.\end{aligned} \tag{20}$$

As before, the cloud provider problem is separable with respect to y_n (since both the objective and the constraints of (20) are separable in y_n). As a result, the required load demand at the $(t+1)$-th iteration is

$$y_n^{t+1} = \Big[\sum\nolimits_{s\in\mathcal{S}}\sum\nolimits_{p\ni n} w_s z_{snp}^t - \lambda_n^t\Big]_0^{\pi_n}, \tag{21}$$

where, in standard notation, $[y]_a^b = \max\{a, \min\{y, b\}\}$. After the cloud controller has determined $\mathbf{y}^{t+1}$, it passes the information to the network controller.

c) The network provider problem: At the $(t+1)$-th epoch, the network provider must solve the problem:

$$\begin{aligned}\text{minimize} \quad & \sum\nolimits_{s\in\mathcal{S}}\Big(x_s^{t+1} + \lambda_s^t - \sum\nolimits_{p\in\mathcal{P}_s}\sum\nolimits_{n\in p} z_{snp}\Big)^2 \\ & + \sum\nolimits_{n\in\mathcal{N}}\Big(y_n^{t+1} + \lambda_n^t - \sum\nolimits_{s\in\mathcal{S}}\sum\nolimits_{p\ni n} w_s z_{snp}\Big)^2 \\ \text{subject to} \quad & z_{snp} \geq 0, \quad z_\ell \leq b_\ell.\end{aligned} \tag{22}$$

The obtained routing solution $\mathbf{z}^{t+1}$ prescribes how routing should be made inside the network and can be instantiated by means of an SDN controller. It is easy to see that (22) is a convex quadratic program, so it can be solved in polynomial time using Karmarkar's algorithm [27–29].[3] The solution of the problem $\mathbf{z}^{t+1}$ along with the updated dual prices $\boldsymbol{\lambda}^{t+1}$ are fed back to the cloud provider and the slice owners, and the process repeats.

In summary, we may characterize our proposed framework as *lightweight*, since each entity must solve simple optimizations. For a pseudocode implementation, see the distributed slice resource allocation (DSRA) algorithm below (Algorithm 1).

C. Algorithm and convergence

In view of all this, our main convergence result is as follows:

[3] In our simulations we use a Column Generation method for quadratic programming to efficiently explore the paths with sparse intermediate solutions.

Theorem 1. The routing allocation profile $(\mathbf{x}^t, \mathbf{y}^t, \mathbf{z}^t)$ returned by Algorithm 1 satisfies the following optimality guarantees:

$$g(\mathbf{x}^t) \to \max g \quad \text{as } t \to \infty, \tag{23a}$$

$$\mathbf{r}^t \to 0 \quad \text{as } t \to \infty, \tag{23b}$$

$$y_n^t \le \pi_n \quad \text{for all } n \in \mathcal{N}, \tag{23c}$$

$$z_\ell^t \le b_\ell \quad \text{for all } \ell \in \mathcal{Z}, \tag{23d}$$

where $\max g$ is the maximum value of the core optimization problem (Opt) and $\mathbf{r}^t$ is the residual vector (15).

In particular, the above theorem states that the routing allocation profile $\mathbf{z}^t$ returned by Algorithm 1 is asymptotically optimal with respect to the slice resource allocation problem (Opt) and satisfies the network's bandwidth and capacity constraints for all t. Theorem 1 is our main theoretical guarantee for Algorithm 1, so we close this section with a few remarks (to maintain the flow of the discussion, the proof of Theorem 1 is relegated to Appendix A:

(i) The algorithm provably converges to a solution of (Opt). Moreover, even if it is cut short prematurely, its output will not exhaust the network's resources (in terms of either bandwidth or processing).

(ii) Algorithm 1 allows the key network entities (slice owners, the cloud provider and the network provider) to coordinate towards obtaining an optimal slice resource allocation.

(iii) As an iterative method, Algorithm 1 is inherently adaptive to changing environments – e.g. if the network has elements that fluctuate over time (wireless links, intermittent link failures, changing traffic demands, etc).

(iv) Algorithm 1 enjoys the fast convergence rate of ADMM: as we show by simulations in the next section, it outperforms the approach of relaxing the capacity constraints.

V. Numerical Results

In this section, we evaluate the performance of the slice resource allocation scheme developed in the previous sections via extensive numerical simulations. Even though we only present here a representative subset of results, our conclusions apply to a wide range of network parameters and specifications. For concreteness, we focus on two complementary settings: *a*) a small toy network to illustrate the flexible functionality of our framework with respect to network slice allocation; and *b*) large networks to demonstrate the scheme's fast convergence and scalability properties.

A. Small network example

We begin by considering a compact mobile edge cloud topology as illustrated in Fig. 5. The network's infrastructure includes two ingress nodes (a and b), two edge cloud servers (c and d), and a single egress core cloud node e. For our experiments we assume that all bandwidth and processing capacities are equal to 1, except on backhaul links (c, e) and (d, e) that are assumed to have unlimited capacity, and nodes a, b, e that are assumed to have zero processing power: this corresponds to a scenario where nodes a and b are SDN nodes while node e is excluded from processing due to high latency.

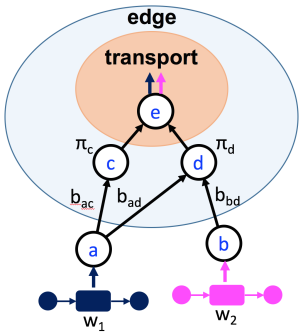

Fig. 5. Toy network used to illustrate the various fairness/balancing trade-offs.

In this setting, we want to provision resources for two slices, one with processing-to-bandwidth ratio $w_1 = 2$, and another with $w_2 = 1/2$. As such, Slice 1 has two possible paths, (a, c, e) and (a, d, e), and can receive processing either at cloud c or d; on the other hand, Slice 2 only has the path (b, d, e) and can receive processing at d. The resource allocation problem in this case is to determine the traffic load per path and processing load per server that optimizes fairness in the sense of (Opt).

In Tables I–III, we present the results for different bandwidth/traffic balancings, as captured by the parameters $\boldsymbol{\theta} = (\theta_1, \theta_2)$: *i*) Table I describes bandwidth-fair allocations; *ii*) Table II is processing-fair; and *iii*) Table III presents a hybrid bandwidth/processing-fair allocation. For each case, we present the optimal allocation for $\alpha = 0$ (maximum total service), $\alpha = 1$ (proportional fair service), $\alpha = 2$ (potential delay fair service), and $\alpha = 10$ (max-min fair service). Focusing for example on this last case, we see that the allocation equalizes service among the slices: in Table I with respect to bandwidth, in Table II with respect to processing (as much as possible while maintaining Pareto efficiency), and in Table III with respect to a mixed requirement. On the contrary, the first line of each table corresponds to the resource allocation profile that maximizes the total respective service.

TABLE I
Bandwidth-fair allocation: $\boldsymbol{\theta} = (1, 1)$.

α	Slice 1		Slice 2
	Path (a, c, e)	Path (a, d, e)	Path (b, d, e)
0	0.5001/1.0001	0.0308/0.0616	0.9692/0.4846
1	0.5001/1.0001	0.2503/0.5006	0.7497/0.3748
2	0.5001/1.0002	0.2499/0.4998	0.7501/0.3751
10	0.5000/1.0000	0.2501/0.5001	0.7499/0.3750

TABLE II
Processing-fair allocation: $\boldsymbol{\theta} = (2, 0.5)$.

α	Slice 1		Slice 2
	Path (a, c, e)	Path (a, d, e)	Path (b, d, e)
0	0.5000/1.0000	0.3844/0.7689	0.4622/0.2311
1	0.5001/1.0001	0.2503/0.5006	0.7497/0.3748
2	0.4999/0.9999	0.0000/0.0000	1.0000/0.5000
10	0.5001/1.0000	0.0000/0.0000	1.0000/0.5000

TABLE III
Mixed processing/bandwidth-fair allocation: $\boldsymbol{\theta} = (2, 1)$.

α	Slice 1		Slice 2
	Path (a, c, e)	Path (a, d, e)	Path (b, d, e)
0	0.5002/1.0005	0.3336/0.6671	0.6664/0.3332
1	0.5001/1.0001	0.2503/0.5006	0.7497/0.3748
2	0.5000/1.0000	0.1217/0.2435	0.8783/0.4391
10	0.4999/0.9999	0.0237/0.0475	0.9763/0.4881

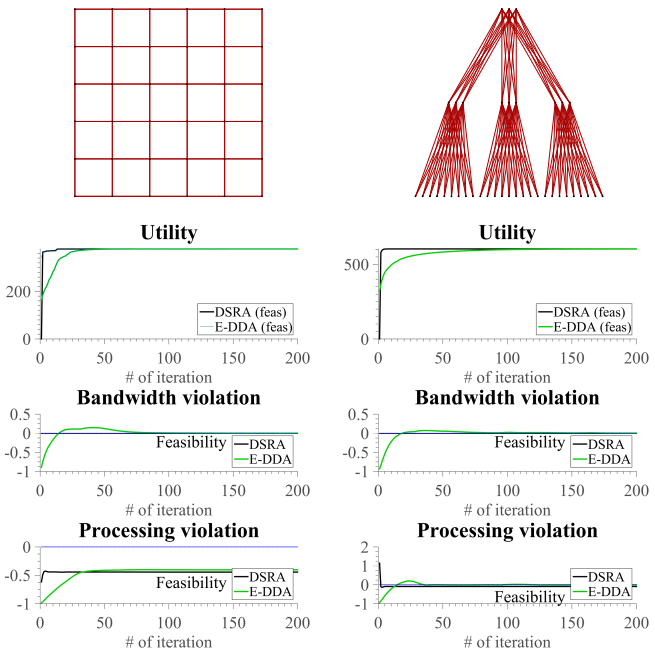

Fig. 6. Convergence to optimal utility and feasibility for 36-node grid (left) and 39-node fat tree topology (right).

B. Large network test-cases

For testing the optimization framework of Sections III and IV in large networks, we consider below *a*) a 6×6 square grid topology with 36 nodes; and *b*) a 39-node fat-tree topology, typical of datacenter networks (cf. Fig. 6). In both networks, roughly 20% of nodes were chosen randomly to represent clouds with processing capabilities, while link and node capacities were drawn uniformly from $[0, 1]$. The network's fairness parameter was set to $\alpha = 0.9$ and, in both cases, we deployed between 70 and 80 slices with processing-to-bandwidth ratios w_s uniformly drawn from $[0, 1]$ (and $\theta_s = 1$ for all s).

To assess the performance of Algorithm 1, we compare the utility gains per iteration against a dual descent algorithm (DDA) that operates by relaxing the problem's capacity constraints, as in the method proposed by [16]. Since DDA is based on relaxing the problem's capacity constraints, the number of iterations required to explore the best paths in the network is immense. In practical scenarios (where path explosion is a key limiting factor), this would immediately disqualify DDA; however, to have a comparison benchmark for the DSRA algorithm (which does not encounter this difficulty because it relaxes the problem's *equality* constraints), we implemented an enhanced dual descent algorithm (E-DDA) which is fed the optimal paths by the DSRA algorithm. Despite this artificial boost, we see that the DSRA algorithm greatly outperforms E-DDA and achieves convergence within a few iterations (less than ten), in both network topologies.

VI. Conclusions and Perspectives

In this paper, we introduced a novel, flexible and *lightweight* resource allocation framework for network slicing by fusing cloud-native design principles with state-of-the-art methods from continuous optimization. The proposed framework admits a natural decomposition with respect to the problem's various stakeholders: *a*) the slice owners (who request bandwidth and processing resources); *b*) the cloud provider (who allocates computing resources); and *c*) the network provider (who assigns end-to-end paths and provides bandwidth). By virtue of this natural splicing, we were able to design an ADMM-based algorithm for distributed slice resource allocation which allows network slices to be provisioned and auto-scaled in real time, while ensuring optimality and efficiency from a traffic-fair and/or computing-fair standpoint.

For simplicity, we focused on network slices comprising a single source-destination pair, used as ingress/egress nodes respectively. Allowing for multiple such entries is a promising extension of our work that would require all S/D flows associated to the same slice to be scaled in or out together. This can be achieved by introducing an additional decision variable in (Opt) to control the relative weight of each S/D pair in a given slice; we intend to explore this in future work.

Finally, in many practical situations arising in the standardization activities for network slicing, the allocation of resources to network slices has to satisfy additional constraints such as slice isolation and satisfaction of delay constraints. Regarding the former, these constraints typically take the form of box constraints that can be handled efficiently in our ADMM framework by incorporating a separable barrier function per slice. The latter may be addressed in a robust manner by using a weighted hop-count metric to bias the paths towards proximal cloud servers. We intend to explore this problem in future work.

Appendix A
Convergence of Algorithm 1

Before presenting the proof of Theorem 1, we introduce for simplification some additional notation. First, in what follows, we write $k = [snp]$ for the multi-index $[snp]$ with $s \in \mathcal{S}, n \in \mathcal{N}$, $p \in \mathcal{P}_s$. Doing so allows us to write the equality constraints of (Opt) in a much more compact form by introducing the matrices $\mathbf{A}, \mathbf{B}, \mathbf{C}$ with components

$$A_{sk} = \mathbb{1}_{p\in\mathcal{P}_s}\,\mathbb{1}_{n\in p} \implies x_s = \textstyle\sum_k A_{sk} z_k, \tag{24a}$$

$$B_{nk} = w_s\,\mathbb{1}_{p\in\mathcal{P}_s} \implies y_n = \textstyle\sum_k B_{nk} z_k, \tag{24b}$$

$$C_{\ell k} = \mathbb{1}_{\ell\in p}\,\mathbb{1}_{n\in p} \implies z_\ell = \textstyle\sum_k C_{\ell k} z_k. \tag{24c}$$

To distinguish between slice and node aggregation, we will also write $\boldsymbol{\lambda}_S = (\lambda_s)_{s\in\mathcal{S}}$ and $\boldsymbol{\lambda}_N = (\lambda_n)_{n\in\mathcal{N}}$ for the profile of slice and node multipliers respectively. Finally, given a sequence of vectors $\mathbf{p}^t$, $t = 0, 1, \dots$, we will write $\Delta_{t_1}^{t_2}\mathbf{p} = \mathbf{p}^{t_2} - \mathbf{p}^{t_1}$ for the drift increment process of $\mathbf{p}$ and $\Delta_{t_1}^{t_2} g(\mathbf{p}) = g(\mathbf{p}^{t_2}) - g(\mathbf{p}^{t_1})$ for every function g taking $\mathbf{p}$ as an argument.

To proceed, we will need the following auxiliary lemma:

Lemma 1. Let $\mathbf{u}^t = (\mathbf{A}\mathbf{z}^t, \mathbf{B}\mathbf{z}^t, \boldsymbol{\lambda}^t)$. Then, the "drift" of the distance between $\mathbf{u}^t$ and an optimum solution $(\mathbf{x}^*, \mathbf{y}^*, \boldsymbol{\lambda}^*)$ of (Opt) is decreasing and satisfies the following inequality:

$$\Delta_t^{t+1}\|\mathbf{A}\Delta_*\mathbf{z}\|^2 + \|\mathbf{B}\Delta_*\mathbf{z}\|^2 + \Delta_t^{t+1}\|\Delta_*\boldsymbol{\lambda}\|^2 \tag{25}$$
$$\leq \frac{2}{\rho}\langle\Delta_*^{t+1}\nabla g(\mathbf{x};\theta), \Delta_*^{t+1}\mathbf{x}\rangle - \left\|\mathbf{A}\Delta_t^{t+1}\mathbf{z}\right\|^2 - \left\|\mathbf{B}\Delta_t^{t+1}\mathbf{z}\right\|^2 - \left\|\Delta_t^{t+1}\boldsymbol{\lambda}\right\|^2.$$

Proof: The function $L_\rho(\mathbf{x}, \mathbf{y}, \mathbf{z}, \boldsymbol{\lambda})$ is concave in $(\mathbf{x}, \mathbf{y}, \mathbf{z})$. Thus, by the first-order optimality conditions for $\mathbf{u}^*$ and $\mathbf{u}^t$ at the t-th iteration, we obtain

$$0 \geq \langle \nabla g(\mathbf{x}^*), \Delta_* \mathbf{x} \rangle - \rho \langle \mathbf{r}^* + \boldsymbol{\lambda}^*, \Delta_* \mathbf{r} \rangle, \tag{26}$$

and
$$\begin{aligned} 0 \geq\ & \langle \nabla g(\mathbf{x}^{t+1}), \Delta_{t+1} \mathbf{x} \rangle \\ & - \rho \langle \mathbf{x}^{t+1} - \mathbf{A}\mathbf{z}^t + \boldsymbol{\lambda}_S^t, \Delta_{t+1} \mathbf{x} \rangle - \rho \langle \mathbf{y}^{t+1} - \mathbf{B}\mathbf{z}^t + \boldsymbol{\lambda}_N^t, \Delta_{t+1} \mathbf{y} \rangle \\ & + \rho \langle \mathbf{r}^{t+1} + \boldsymbol{\lambda}_S^t, \mathbf{A}\Delta_{t+1} \mathbf{z} \rangle + \rho \langle \mathbf{r}^{t+1} + \boldsymbol{\lambda}_N^t, \mathbf{B}\Delta_{t+1} \mathbf{z} \rangle, \end{aligned} \tag{27}$$

for all feasible $(\mathbf{x}, \mathbf{y}, \mathbf{z})$. Then, applying the above to the t-th iterate of the algorithm and the optimum of (Opt), we obtain

$$\begin{aligned} \rho \langle \Delta_*^t \boldsymbol{\lambda}^t + \Delta_*^{t+1} \mathbf{r}, \Delta_*^{t+1} \mathbf{r} \rangle + \rho \langle \mathbf{A}\Delta_t^{t+1} \mathbf{z}, \Delta_*^{t+1} \mathbf{x} \rangle + \rho \langle \mathbf{B}\Delta_t^{t+1} \mathbf{z}, \Delta_*^{t+1} \mathbf{y} \rangle \\ \leq \langle \Delta_*^{t+1} \nabla g(\mathbf{x}), \Delta_*^{t+1} \mathbf{x} \rangle. \end{aligned} \tag{28}$$

Using the dual update (17d), the first term above may be written equivalently as $\rho \langle \Delta_*^{t+1} \boldsymbol{\lambda}, \Delta_t^{t+1} \boldsymbol{\lambda} \rangle$. Then, rearranging terms, we obtain

$$\begin{aligned} & \rho \langle \Delta_*^{t+1} \boldsymbol{\lambda}, \Delta_t^{t+1} \boldsymbol{\lambda} \rangle + \rho \langle \mathbf{A}\Delta_t^{t+1} \mathbf{z}, \mathbf{A}\Delta_*^{t+1} \mathbf{z} \rangle + \rho \langle \mathbf{B}\Delta_t^{t+1} \mathbf{z}, \mathbf{B}\Delta_*^{t+1} \mathbf{z} \rangle \\ & \leq \langle \Delta_*^{t+1} \nabla g(\mathbf{x}), \Delta_*^{t+1} \mathbf{x} \rangle - \rho \langle \mathbf{A}\Delta_t^{t+1} \mathbf{z}, \Delta_t^{t+1} \boldsymbol{\lambda}_S \rangle - \rho \langle \mathbf{B}\Delta_t^{t+1} \mathbf{z}, \Delta_t^{t+1} \boldsymbol{\lambda}_N \rangle. \end{aligned}$$

By the optimality of $\mathbf{z}^t$ and $\mathbf{z}^{t+1}$ for (22), we also have

$$\rho \langle \mathbf{r}^t + \boldsymbol{\lambda}_S^{t-1}, \mathbf{A}\Delta_t^{t+1} \mathbf{z} \rangle + \rho \langle \mathbf{r}^t + \boldsymbol{\lambda}_N^{t-1}, \mathbf{B}\Delta_t^{t+1} \mathbf{z} \rangle \leq 0, \tag{29}$$

$$\rho \langle \mathbf{r}^{t+1} + \boldsymbol{\lambda}_S^t, \mathbf{A}\Delta_{t+1}^t \mathbf{z} \rangle + \rho \langle \mathbf{r}^{t+1} + \boldsymbol{\lambda}_N^t, \mathbf{B}\Delta_{t+1}^t \mathbf{z} \rangle \leq 0. \tag{30}$$

Using $\mathbf{r}^t + \boldsymbol{\lambda}^{t-1} = \boldsymbol{\lambda}^t$ and $\mathbf{r}^{t+1} + \boldsymbol{\lambda}^t = \boldsymbol{\lambda}^{t+1}$, and summing up the two equations yields directly the announced inequality. Our assertion then follows by noting that $2\langle \Delta_t^{t+1} \mathbf{p}, \Delta_*^{t+1} \mathbf{p} \rangle = \|\Delta_t^{t+1} \mathbf{p}\|^2 + \Delta_t^{t+1} \|\Delta_* \mathbf{p}\|^2$ for any vector $\mathbf{p}$. ∎

Proof of Theorem 1: We explain now how Theorem 1 can be deduced from Lemma 1. Due to the capacity constraints (13a) and (13b), optimal values $\mathbf{u}^*$ are necessarily finite (though not necessarily unique). By the concavity of g, the first term on the right-hand side is non-positive. As all other terms are non-positive as well, it follows that the left-hand side is non-positive. This means that the distances between $\mathbf{A}\mathbf{z}^t$ and $\mathbf{A}\mathbf{z}^* \equiv \mathbf{x}^*$, between $\mathbf{B}\mathbf{z}^t$ and $\mathbf{B}\mathbf{z}^* = \mathbf{y}^*$, and between $\boldsymbol{\lambda}^t$ and $\boldsymbol{\lambda}^*$ only decrease with each iteration, so they must converge to some finite limit. It follows that the right-hand side must actually tend to 0 and, hence, $\Delta_t^{t+1} \boldsymbol{\lambda} \equiv \mathbf{r}^{t+1} \to 0$ as $t \to \infty$, implying in turn that the consistency constraints (13c)–(13d) are met as $t \to \infty$.

Focusing again on the left-hand side, since the optimal value $\mathbf{u}^*$ is bounded and the distance to it converges, we can always extract converging subsequences from $(\mathbf{u}^t)_{t \in \mathbb{N}}$. Thus, letting $\bar{\mathbf{u}}$ be a limit-point of such a subsequence, (26) readily yields $\langle \nabla g(\bar{\mathbf{x}}), \Delta^* \mathbf{x} \rangle \leq 0$. By concavity, this implies $g(\bar{\mathbf{x}}) \leq g(\mathbf{x}^*)$. As a result, asymptotically as $t \to \infty$, the allocations $(\mathbf{x}^t, \mathbf{y}^t, \mathbf{z}^t)$ obtained by Algorithm 1 satisfy (13c)–(13d), and achieve the optimum of (Opt). Furthermore, for all t, $\mathbf{z}^t$ and $\mathbf{y}^t$ satisfy (13b)–(13a) by (17b)–(17c), so our proof is complete. ∎

References

[1] NGMN Alliance, "5G white paper," Tech. Rep., 2015.

[2] A. Fischer, J. F. Botero, M. T. Beck, H. de Meer, and X. Hesselbach, "Virtual network embedding: A survey," *IEEE Communications Surveys Tutorials*, pp. 1888–1906, 2013.

[3] D. Bhamare, R. Jain, M. Samaka, and A. Erbad, "A survey on service function chaining," *J. Netw. Comput. Appl.*, pp. 138–155, 2016.

[4] B. Addis, D. Belabed, M. Bouet, and S. Secci, "Virtual network functions placement and routing optimization," in *CloudNet*, 2015.

[5] M. Gao, B. Addis, M. Bouet, and S. Secci, "Optimal orchestration of virtual network functions," *arXiv:1706.04762*, 2017.

[6] A. Gupta, M. F. Habib, P. Chowdhury, M. Tornatore, and B. Mukherjee, "Joint virtual network function placement and routing of traffic in operator networks," in *NetSoft*, 2015.

[7] M. Mechtri, C. Ghribi, and D. Zeghlache, "A scalable algorithm for the placement of service function chains," *IEEE Transactions on Network and Service Management*, pp. 533–546, 2016.

[8] M. R. Rahman and R. Boutaba, "SVNE: Survivable virtual network embedding algorithms for network virtualization," *IEEE Transactions on Network and Service Management*, pp. 105–118, 2013.

[9] J. Erfanian, et al., "Network operator perspectives on NFV priorities for 5G," NFV#17 Plenary Meeting, 2017.

[10] G. Brown, "Designing cloud-native 5G core networks," Heavy Reading, Feb. 2017.

[11] Cloud Native Computing Foundation (CNCF), "Charter document," https://www.cncf.io/about/charter.

[12] Kubernetes Project, "Reference documentation," https://kubernetes.io/docs/reference/.

[13] S. Shakkottai and R. Srikant, "Network optimization and control," *Found. Trends Netw.*, pp. 271–379, 2007.

[14] H. Feng, J. Llorca, A. Tulino, D. Raz, and A. Molisch, "Approximation algorithms for the NFV service distribution problem," in *INFOCOM*, 2017.

[15] L. Tassiulas and A. Ephremides, "Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks," *IEEE Transactions on Automatic Control*, pp. 1936–1948, 1992.

[16] X. Lin and N. B. Shroff, "Utility maximization for communication networks with multipath routing," *IEEE Transactions on Automatic Control*, pp. 766–781, 2006.

[17] J. Elias, F. Martignon, S. Paris, and J. Wang, "Efficient orchestration mechanisms for congestion mitigation in nfv: Models and algorithms," *IEEE Transactions on Services Computing*, 2015.

[18] S. H. Low and D. E. Lapsley, "Optimization flow control. I. Basic algorithm and convergence," *IEEE/ACM Transactions on Networking*, pp. 861–874, 1999.

[19] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows*. Prentice Hall, 1993.

[20] S. Lee, S. Pack, M.-K. Shin, E. Paik, and R. Browne, "Resource management in service chaining," IETF Secretariat, Internet-Draft, 2016.

[21] "Cisco cloud services router 1000v," Data Sheet.

[22] P. Raghavan and C. D. Thompson, "Randomized rounding: A technique for provably good algorithms and algorithmic proofs," *Combinatorica*, 1987.

[23] F. P. Kelly, A. K. Maulloo, and D. K. H. Tan, "Rate control in communication networks: shadow prices, proportional fairness and stability," *Journal of the Operational Research Society*, pp. 237–252, 1998.

[24] J. Mo and J. Walrand, "Fair end-to-end window-based congestion control," *IEEE/ACM Transactions on Networking*, pp. 556–567, 2000.

[25] B. Radunovic and J. Y. L. Boudec, "A unified framework for max-min and min-max fairness with applications," *IEEE/ACM Transactions on Networking*, pp. 1073–1083, 2007.

[26] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Found. Trends Mach. Learn.*, pp. 1–122, 2011.

[27] M. Kozlov, S. Tarasov, and L. Khachiyan, "The polynomial solvability of convex quadratic programming," *Elsevier, USSR Computational Mathematics and Mathematical Physics*, pp. 223–228, 1980.

[28] S. Kapoor and P. Vaidya, "Fast algorithms for convex quadratic programming and multicommodity flows," in *STOC*, 1986.

[29] Y. Ye and E. Tse, "An extension of Karmarkar's projective algorithm for convex quadratic programming," *Springer, Mathematical Programming*, pp. 157–179, 1989.