Elastic Multi-resource Network Slicing: Can Protection Lead to Improved Performance?

Jiaxiao Zheng, Gustavo de Veciana
Dept. of Electrical and Computer Engineering, The University of Texas at Austin, TX
gustavo@ece.utexas.edu

***Abstract*—In order to meet the performance/privacy requirements of future data-intensive mobile applications, e.g., self-driving cars, mobile data analytics, and AR/VR, service providers are expected to draw on shared storage/computation/connectivity resources at the network "edge". To be cost-effective, a key functional requirement for such infrastructure is enabling the sharing of heterogeneous resources amongst tenants/service providers supporting spatially varying and dynamic user demands. This paper proposes a resource allocation criterion, namely, Share Constrained Slicing (SCS), for slices allocated predefined shares of the network's resources, which extends traditional $\alpha-$fairness criterion, by striking a balance among inter- and intra-slice fairness vs. overall efficiency. We show that SCS has several desirable properties including slice-level *protection*, *envyfreeness*, and *load-driven elasticity*. In practice, mobile users' dynamics could make the cost of implementing SCS high, so we discuss the feasibility of using a simpler (dynamically) weighted max-min as a surrogate resource allocation scheme. For a setting with stochastic loads and elastic user requirements, we establish a sufficient condition for the stability of the associated coupled network system. Finally, and perhaps surprisingly, we show via extensive simulations that while SCS (and/or the surrogate weighted max-min allocation) provides inter-slice protection, they can achieve improved job delay and/or perceived throughput, as compared to other weighted max-min based allocation schemes whose intra-slice weight allocation is not share-constrained, e.g., traditional max-min or discriminatory processor sharing.**

I. Introduction

Next generation networks face the challenge of supporting data-intensive services and applications, such as self-driving cars, infotainment, augmented/virtual reality [26], Internet of things [6], [26], and mobile data analytics [1], [11]. In order to accommodate the performance and privacy requirements of such services/applications, providers are expected to draw on shared storage/computation/connectivity resources at the network "edge". Such network systems can take advantage of Software-Defined Networking and Network Function Virtualization technologies to provision slices of shared heterogeneous resources and network functions which are customized to service providers'/tenants' requirements.

The ability to support slice-based provisioning is central to enabling service providers to take control of managing performance of their own dynamic and mobile user populations. This also improves the scalability by reducing the complexity of performance management on multi-service platforms. The ability to efficiently share network/compute resources is also key to reducing the cost of deploying such services. By contrast with today's cloud computing platforms, our focus in this paper is on provisioning slices of edge resources to meet mobile users/devices requirements. In general, shared edge resources will have smaller overall capacity resulting in reduced statistical multiplexing and making efficiency critical. Perhaps as is the case with cloud computing platforms, providers/tenants will want to make long-term provisioning commitments enabling predictable costs and resource availability, yet benefit, when possible, from elastic resource allocations aligned with spatial variations in their mobile workloads but not at the expense of other slices. Thus a particularly desirable feature is to enable slice-level provisioning agreements which achieve inter-slice protection, load-driven elasticity and network efficiency.

These challenges distinguish our work from previous research in areas including engineering, computer science and economics. The standard framework used in communication networks is utility maximization (see e.g., [28] and references therein), which has led to the design of several transport and scheduling mechanisms and criteria, e.g., the widely discussed proportional fairness. When considering dynamic/stochastic networks, e.g., [4], [16], researchers have studied networks where users are allocated resources based on utility maximization and studied requirements for network stability for 'elastic' user demands, e.g., file transfers. This body of work emphasizes user-level resource allocation, without specifically accounting for interactions among slices. Thus, it does not directly address the requirements of network slicing.

Instead in this paper, we propose an approach, namely, *Share Constrained Slicing (SCS)*, wherein each slice is assigned a share of the overall resources, and in turn, distributes its share among its users. Then the user level resource allocation is determined by maximizing a sharing criterion. When SCS is applied to a setting where each user only demands one resource, for example, slices sharing wireless resources in cellular networks [8], [9], [32], the scheme can be viewed as a Fisher market where agents (slices), which are share (budget) constrained, bid on network resources, see, e.g., [24], and for applications [3], [8], [18]. However, those works do not deal with settings where users require heterogeneous resources, and how to orchestrate slice-level interactions on different resources is not clear yet.

When it comes to sharing heterogeneous resources, a simple solution is static partitioning all resources according to a service-level agreement, see, e.g., [20]. This offers each slice a guaranteed allocation of the network resources thus in principle provides ideal protection among slices. However, it falls short from the perspective of providing load-driven elasticity to a slice's users, possibly resulting in either resource under-utilization or over-booking. Other natural approaches include full sharing [2], where users from all slices are served based on some prioritizing disciplines without prior resource reservation. Such schemes may not achieve slice-level protection and are vulnerable to surging user traffic across slices.

Additionally, many resource sharing schemes have been proposed for cluster computing where heterogeneous resources are

involved, including Dominant Resource Fairness (DRF) [19], Competitive Equilibrium from Equal Income (CEEI) [23] [30] [31], Bottleneck Max Fairness (BMF) [5], etc. These allocation schemes are usually based on modelling joint resource demands of individual users, but lack of the notion of slicing, thus it is not clear how to incorporate the need to enable slice-level long-term commitments. In these works, inter-slice protection and elasticity of allocations have not been characterized. Furthermore, most of these works are developed under the assumption that users are sharing a centralized pool of resources. In this paper we focus on a settings where resources are distributed, and mobile users are restricted to be served by proximal edge resources.

Contributions: The novelty of our proposed approach lies in maintaining slice-level *long-term commitments* defined by *shares*, which can be seen as a service-level agreement, while enabling user-level resource provisioning which is driven by dynamic user loads. We consider a model where users possibly require heterogeneous resources in different proportions, and the processing rate of a user scales linearly in the amount of resources it is allocated. Such a model captures tasks/services which speed up in the allocated resources. This is discussed further in the sequel.

We show that SCS can capture inter- and intra-slice fairness separately. When viewed as a resource sharing criterion, SCS is shown to satisfy a set of axiomatically desirable properties akin to those in [21], and can be interpreted as achieving a tunable trade-off among inter-slice fairness (which can be seen as a proxy of protection), intra-slice fairness, and overall utilization. Fairness is connected to load-driven elasticity through share constrained weight allocation. The merits of SCS are demonstrated in both static and dynamic settings. In static settings, we prove a set of desirable properties of SCS as a sharing criterion, including slice-level *protection* and *envyfreeness*, and we demonstrate the feasibility of using a computationally simpler (dynamically) weighted max-min as a surrogate resource allocation scheme for the cases where the cost of implementing SCS is excessive. In dynamic settings, we consider the elastic traffic model with each user as a service requirement, and leaves the system once it is processed. We model such system as a stochastic queuing network, and establish its stability condition. Due to lack of space, detailed proofs are included in [33].

Finally, and perhaps surprisingly, we show via extensive simulations that while SCS provides inter-slice protection, it need not compromise performance. We achieve improved average job delay and/or perceived throughput, as compared with multiple variations of traditional (weighted) max-min fair allocations but without share-constrained weight allocation. Roughly speaking, SCS can separate the busy-periods of different slices, thus reduces inter-slice contention, leading to improved performance. We validate this insight through simulations.

II. Resource sharing in network slicing

In this section we will briefly introduce our overall framework for resource allocation to network slices, namely, Share Constrained Slicing (SCS) where each slice manages a possibly dynamic set of users. Specifically, we consider resource allocation driven by the maximization of an objective function geared at achieving a trade-off between overall efficiency and fairness [21].

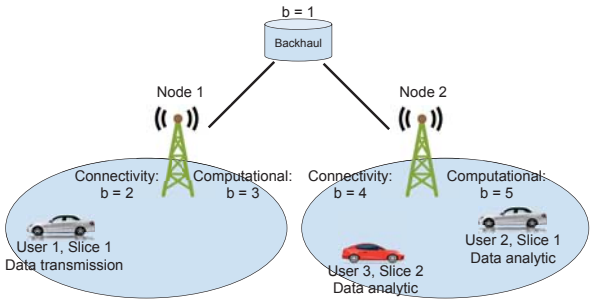


Fig. 1: Example: network slicing in edge computing with autonomous cars.

To begin, we consider the set of active users on each slice to be fixed. Let us denote the set of slices by $\mathcal{V}$, the set of resources by $\mathcal{R}$, each with a capacity normalized to 1. Each slice v supports a set of user classes, denoted by $\mathcal{C}^v$, and the total set of user classes is defined as $\mathcal{C} := \cup_{v\in\mathcal{V}}\mathcal{C}^v$. For simplicity, we let $v(c)$ denote the slice which supports class c. We let $\mathcal{U}_c$ denote the set of users of class c, and the users on slice v is denoted by $\mathcal{U}^v := \cup_{c\in\mathcal{C}^v}\mathcal{U}_c$. Also, the overall set of users is $\mathcal{U} := \cup_{v\in\mathcal{V}}\mathcal{U}^v$. For each user, possibly heterogeneous resources are required to achieve a certain processing rate. Let us denote the processing rate seen by user u by λ_u. We also define the resource demand vector of user class c as $\mathbf{d}_c := (d_c^r : r \in \mathcal{R})$, where d_c^r is the fraction of resource r required by user $u \in \mathcal{U}_c$ for a unit processing rate, i.e., to achieve $\lambda_u = 1$, we need to allocate fraction d_c^1 of the total amount of resource 1 to u, d_c^2 of the total amount of resource 2 to u, and so on. If a class c user does not need a given resource r then $d_c^r = 0$. Note that if two slices support users with the same requirements, we will distinguish them by defining two distinct user classes one for each slice. In other words, it is possible to have more than one user classes with exactly the same vector $\mathbf{d}_c$. Also, we let $\mathcal{R}_c$ denote the set of resources required by users of class c, and in turn, let the set $\mathcal{C}_r$ denote user classes using resource r. Among $\mathcal{C}_r$, the set of classes on slice v is $\mathcal{C}_r^v := \mathcal{C}_r \cap \mathcal{C}^v$. The number of active users of class c at time t is denoted by a random variable $N_c(t)$, and that on slice v by $N^v(t)$. $N^v(t) = \sum_{c\in\mathcal{C}^v} N_c(t)$. Realizations of these are denoted by lower case variables n_c and n^v, respectively.

This model captures the services/applications where tasks speed up with more allocated resources, e.g., a file download is faster when allocated more communication resources, or a computational task that can be parallelized, e.g., typical MapReduce jobs [17], and mobile data analytics when additional compute resources are available [11]. For more complex applications involving different types of stages, the stages associated with massive data processing might be parallelizable, making it possible to accelerate processing by allocating more resources. For example, in mobile cloud gaming [27], the most time-consuming and resource-demanding stage is usually the cloud rendering where a computing cluster renders the frames of the game. The rendering procedure can be accelerated by allocating more GPUs, and thus, can be viewed as a quantized version of our model.

Example: Let us consider an example where there are two autonomous vehicle service operators, say Slice 1 and Slice 2, coexisting in the same area, and supported by two edge computing nodes equipped with fronthaul connectivity

User \ Resource	1	2	3	4	5	Rate
user 1	0.4	0.4	0	0	0	$\lambda_1 = 0.4$
user 2	0.3	0	0	0.5	0.5	$\lambda_2 = 0.5$
user 3	0.3	0	0	0.5	0.5	$\lambda_3 = 0.5$

TABLE I: Example resource allocation

and computational resources (e.g., edge GPUs), as shown in Fig. 1. Both nodes are connected to the same backhaul node. Different resources at different locations are indexed as in the figure. There are 3 vehicles (users) in this area, each of which corresponds to a user class. Users 1 and 2 are on Slice 1, and User 3 is on Slice 2, respectively. Each autonomous vehicle can run either of two applications. User 1 is conducting simple file transfer, with the resource demand vector $\mathbf{d}_1 = (1, 1, 0, 0, 0)$, meaning that User 1's application involves only connectivity resources, and to achieve a unit transmission rate for User 1, the system needs to allocate all the connectivity resources at both Node 1 and the backhaul. Meanwhile, Users 2 and 3 are performing mobile data analytics, with demand vectors $\mathbf{d}_2 = \mathbf{d}_3 = (0.6, 0, 0, 1, 1)$, meaning that to achieve a unit processing rate for Users 2 or 3, the system needs to allocate 60% of the backhaul resource, all the fronthaul resource, together with all the computational resource at Node 2. Then, for example, if the resource allocation is as given in Table I, the system can achieve user service/processing rates given by $\lambda_1 = 0.4, \lambda_2 = \lambda_3 = 0.5$.

Next, we introduce the concept of *network share* and how it impact the weights across users.

Definition 1. Share-constrained weight allocation (SCWA): *Given a vector of slice shares $\mathbf{s} := (s_v \geq 0 : v \in \mathcal{V})$, each slice v is assigned a positive share s_v, representing the fraction of overall resources to be committed to slice v. In turn, it distributes its share s_v across its users $\mathcal{U}^v$, yielding a weight allocation to users $\mathbf{w} := (w_u \geq 0 : u \in \mathcal{U})$ such that for each slice v,*

$$\sum_{u \in \mathcal{U}^v} w_u = s_v. \tag{1}$$

Without loss of generality, we assume $\sum_{v \in \mathcal{V}} s_v = 1$. If we denote the aggregate weight of the users in each class c by $q_c := \sum_{u \in \mathcal{U}_c} w_u$, and the weight allocation across user classes by $\mathbf{q} = (q_c : c \in \mathcal{C})$, thus Eq. (1) implies $\sum_{c \in \mathcal{C}^v} q_c = s_v$. Furthermore, we denote the intra-slice weight allocation (across user classes) by $\mathbf{q}^v := (q_c : c \in \mathcal{C}^v)$. As a result, a slice can increase its users' weights by increasing its share. Also, note that if the number of users on a slice surges without increasing the associated share, on average each of its users will be given less weight. Two examples of SCWA are as follows.

1) **Equal intra-class weight allocation**, where user weights are the same within a user class, i.e., $w_u = \frac{q_c}{n_c}$, for $u \in \mathcal{U}_c$ with $\sum_{c \in \mathcal{C}^v} q_c = s_v$. In this paper, we assume this always holds true.
2) **Equal intra-slice weight allocation**, where user weights are the same across the slice, i.e., $w_u = \frac{s_v}{n^v}$, for $u \in \mathcal{U}^v$. As a result, $q_c = \frac{s_{v(c)} n_c}{n^{v(c)}}$. One can see that equal intra-slice allocation is a further special case of equal intra-class allocation. When each user only demands one resource, such an allocation emerges naturally as the social optimal of a Fisher market, and Nash equilibrium when slices exhibit (price-taking) strategic behavior in optimizing their own utility, see [7].

Next we discuss how to map weights to rate allocations. Let us denote the user rate allocation by $\boldsymbol{\lambda} := (\lambda_u : u \in \mathcal{U})$. The aggregated rate allocation across user classes is denoted by $\boldsymbol{\phi} = (\phi_c : c \in \mathcal{C})$, where $\phi_c := n_c \lambda_u, u \in \mathcal{U}_c$, due to equal intra-class weight allocation. For each slice v, the intra-slice rate allocation is $\boldsymbol{\phi}^v := (\phi_c : c \in \mathcal{C}^v)$. The rate allocation across slices is $\boldsymbol{\gamma} := (\gamma_v := \sum_{c \in \mathcal{C}^v} \phi_c : v \in \mathcal{V})$. The overall rate across the system is $\lambda := \|\boldsymbol{\lambda}\|_1 = \|\boldsymbol{\phi}\|_1 = \|\boldsymbol{\gamma}\|_1$, where $\|\cdot\|_1$ is the L1-norm. SCS is thus defined as follows.

Definition 2. $\alpha-$***Share Constrained Slicing*** **($\alpha-$SCS):** *Under equal intra-class weight allocation with class weights $\mathbf{q}$, we say a class-level rate allocation $\boldsymbol{\phi}$ corresponds to $\alpha-$SCS if it is the solution to the following optimization problem*

$$\max_{\boldsymbol{\phi}} \ \{U_\alpha(\boldsymbol{\phi}; \mathbf{q}) : \sum_{c \in \mathcal{C}_r} d_c^r \phi_c \leq 1, \forall r \in \mathcal{R}\}, \tag{2}$$

where $\alpha > 0$ is a pre-defined parameter,

$$U_\alpha(\boldsymbol{\phi}; \mathbf{q}) := \begin{cases} e^{\sum_{v \in \mathcal{V}} U_\alpha^v(\boldsymbol{\phi}^v; \mathbf{q}^v)} & \alpha = 1, \\ \sum_{v \in \mathcal{V}} U_\alpha^v(\boldsymbol{\phi}^v; \mathbf{q}^v) & \alpha > 0 \ \text{ and } \ \alpha \neq 1, \end{cases}$$

and $U_\alpha^v(\boldsymbol{\phi}^v; \mathbf{q}^v)$ is the utility function of slice v, given by

$$U_\alpha^v(\boldsymbol{\phi}^v; \mathbf{q}^v) := \begin{cases} \sum_{c \in \mathcal{C}^v} q_c \log\left(\frac{\phi_c}{q_c}\right) & \alpha = 1, \\ \sum_{c \in \mathcal{C}^v} q_c \frac{(\phi_c / q_c)^{1-\alpha}}{1-\alpha} & \alpha > 0 \ \text{ and } \ \alpha \neq 1. \end{cases}$$

The criterion underlying SCS is different from the traditional class-level (weighted) $\alpha-$fairness criterion proposed in [22] and [4], which is defined as follows.

Definition 3. Class-level $\alpha-$fairness: *Under equal intra-class weight allocation, with class-level weights $\mathbf{q}$, a class-level rate allocation $\boldsymbol{\phi}$ corresponds to (weighted) $\alpha-$fairness if it is the solution to Problem (2) where the utility function of slice v is given by*

$$U_\alpha^v(\boldsymbol{\phi}^v; \mathbf{q}^v) := \begin{cases} \sum_{c \in \mathcal{C}^v} q_c \log(\phi_c) & \alpha = 1, \\ \sum_{c \in \mathcal{C}^v} q_c \frac{(\phi_c)^{1-\alpha}}{1-\alpha} & \alpha > 0 \ \text{ and } \ \alpha \neq 1. \end{cases}$$

As shown in [22], $\alpha-$fairness is equivalent to (weighted) proportional fairness when $\alpha = 1$ and unweighted maxmin fairness when $\alpha \to \infty$. By contrast, a characterization of $\alpha-$SCS resource allocation is given as follows.

Corollary 1. *$\alpha-$SCS is equivalent to (weighted) proportional fairness when $\alpha = 1$, and weighted max-min fairness when $\alpha \to \infty$.*

Under equal intra-class weight allocation, weighted proportional fairness is defined as the solution to the following problem:

$$\max_{\boldsymbol{\phi}} \ \left\{ \sum_{c \in \mathcal{C}} q_c \log \phi_c : \sum_{c \in \mathcal{C}_r} d_c^r \phi_c \leq 1, \forall r \in \mathcal{R} \right\}, \tag{3}$$

while weighted max-min fairness is defined as the solution to the following problem:

$$\max_{\boldsymbol{\phi}} \ \left\{ \min_{c \in \mathcal{C}} \frac{\phi_c}{q_c} : \sum_{c \in \mathcal{C}_r} d_c^r \phi_c \leq 1, \forall r \in \mathcal{R} \right\}. \tag{4}$$

In summary, under $\alpha-$SCS resource allocation the weight (and thus slice shares) have a consistent impact even when $\alpha \to \infty$, retaining a notion of inter-slice protection. To the

best of our knowledge, SCS is the first variation of $\alpha-$fairness incorporating such slice-based weighting in a consistent manner.

Let us define function $f_\alpha(\mathbf{x};\mathbf{y})$ of two positive vectors $\mathbf{x},\mathbf{y}\in\mathbb{R}_+^n$ such that $\|\mathbf{x}\|_1=\|\mathbf{y}\|_1=1$ as

$$f_\alpha(\mathbf{x};\mathbf{y})=\begin{cases} e^{-D_{KL}(\mathbf{y}\|\mathbf{x})} & \alpha=1,\\ \left(\sum_i y_i\left(\frac{x_i}{y_i}\right)^{1-\alpha}\right)^{\frac{1}{\alpha}} & \alpha>0,\alpha\neq 1,\end{cases} \tag{5}$$

where $D_{KL}(\cdot\|\cdot)$ represents the Kullback-Leibler (K-L) divergence. The function $f_\alpha(\mathbf{x};\mathbf{y})$ can be viewed as a measure of how close a *normalized* resource allocation $\mathbf{x}$ is to a *normalized* weight vector $\mathbf{y}$ in that, for example, when $\alpha=1$, it decreases with the K-L divergence between $\mathbf{x}$ and $\mathbf{y}$, thus assumes maximum when $\mathbf{x}=\mathbf{y}$, meaning that the rate allocation is aligned with the specified weights. For general α and a given $\mathbf{y}$, when no other constraint is imposed, $f_\alpha(\mathbf{x};\mathbf{y})$ achieves maximum when $\mathbf{x}=\mathbf{y}$. Thus $f_\alpha(\mathbf{x};\mathbf{y})$ can be interpreted as a measure of $\mathbf{y}-$weighted fairness of the allocation $\mathbf{x}$.

Let us define the efficiency/utilization of a resource allocation as a concave non-decreasing function of the overall λ,

$$E_\alpha(\lambda):=\begin{cases}\lambda & \alpha=1,\\ \frac{\lambda^{1-\alpha}}{1-\alpha} & \alpha>0 \text{ and } \alpha\neq 1.\end{cases}$$

We define an inter-slice fairness function as

$$F_\alpha^{\text{inter}}(\boldsymbol{\gamma}):=f_\alpha(\tilde{\boldsymbol{\gamma}};\mathbf{s}),$$

where $\tilde{\boldsymbol{\gamma}}:=(\tilde{\gamma}^v:=\gamma^v/\lambda:v\in\mathcal{V})$ is the normalized aggregated rate allocation across slices. In view of Eq. (1), we shall also define the normalized weight allocation for slice v as $\tilde{\mathbf{q}}^v:=(\tilde{q}_c:=\frac{q_c}{s_v}:c\in\mathcal{C}^v)$, and the normalized rate allocation across user classes on slice v as $\tilde{\boldsymbol{\phi}}^v:=(\tilde{\phi}_c:=\frac{\phi_c}{\gamma^v}:c\in\mathcal{C}^v)$. Finally we define an intra-slice fairness measure as

$$F_\alpha^{\text{intra}}(\boldsymbol{\phi};\mathbf{q}):=\begin{cases} e^{\sum_{v\in\mathcal{V}} t_\alpha^v(\tilde{\boldsymbol{\gamma}};\mathbf{s})\log f_\alpha(\tilde{\boldsymbol{\phi}}^v;\tilde{\mathbf{q}}^v)} & \alpha=1,\\ \left(\sum_{v\in\mathcal{V}} t_\alpha^v(\tilde{\boldsymbol{\gamma}};\mathbf{s})(f_\alpha(\tilde{\boldsymbol{\phi}}^v;\tilde{\mathbf{q}}^v))^\alpha\right)^{\frac{1}{\alpha}} & \alpha\neq 1,\end{cases}$$

where $t_\alpha^v(\tilde{\boldsymbol{\gamma}};\mathbf{s})$ can be viewed as a weight impacting the importance of the fairness for each slice v:

$$t_\alpha^v(\tilde{\boldsymbol{\gamma}};\mathbf{s}):=\frac{s_v(\frac{\tilde{\gamma}^v}{s_v})^{1-\alpha}}{\sum_{v'\in\mathcal{V}} s_{v'}(\frac{\tilde{\gamma}^{v'}}{s_{v'}})^{1-\alpha}}. \tag{6}$$

With these definitions in place we can show that for a given α, the $\alpha-$SCS criterion can be expressed as follows.

Proposition 1. *Under the $\alpha-$SCS criterion, we have that*

$$U_\alpha(\boldsymbol{\phi};\mathbf{q})=E_\alpha(\lambda)\left(F_\alpha^{inter}(\boldsymbol{\gamma})F_\alpha^{intra}(\boldsymbol{\phi};\mathbf{q})\right)^\alpha, \tag{7}$$

where $E_\alpha(\lambda)$, $F_\alpha^{inter}(\boldsymbol{\gamma})$ and $F_\alpha^{intra}(\boldsymbol{\phi};\mathbf{q})$ can be interpreted as the overall network efficiency, inter-slice and intra-slice fairness, respectively.

Based on Eq. (7) it should be clear that $\alpha-$SCS embodies a trade-off among overall network efficiency, inter- and intra-slice fairness, which can be viewed as a proxy for the degree of protection offered to slices, where the significance of fairness grows as α increases. When $\alpha\to 0$, $\alpha-$SCS corresponds to maximizing the overall rate allocation, regardless of slice shares. In order to achieve flexible resource utilization, a sharing criterion should realize *load-driven elasticity*, i.e., the amount of resources provisioned to a user class would ideally increase in the number of users in the class. Under equal intra-slice weight allocation, from Eq. (7) one can observe that, due to the fairness terms, the relative resource allocation of a slice tends to be aligned with $\tilde{\mathbf{q}}^v=(\frac{n_c}{n^v}:c\in\mathcal{C}^v)$, i.e., its relative load distribution. Thus the load-driven elasticity of $\alpha-$SCS is achieved as a result of weighted fairness. Specifically, under SCS when users' requirements are decoupled, i.e., each user only uses one resource, $|\mathcal{R}_c|=1,\forall c\in\mathcal{C}$, one can show the following result.

Proposition 2. *Under equal intra-slice weight allocation, assuming $|\mathcal{R}_c|=1,\forall c\in\mathcal{C}$, $\alpha-$SCS is such that ϕ_c is a monotonically increasing function of n_c, when $n_{c'}$ is fixed for $c'\neq c$.*

Proof: Specifically in the setting of Proposition 2, solving the associated KKT condition, one can show that each resource r will provision its resource across user classes in proportion to $\frac{s_{v(c)}n_c}{n^{v(c)}}$. ∎

Such load and share-dependant elasticity is key to achieving a sharing scheme that embodies the inter-slice protection, while still improving the resource utilization by accommodating dynamic user loads on different slices.

III. Static Analysis

A. System model

In this section we will take a closer look at the characterization of $\alpha-$SCS slice level resource allocations. The $\alpha-$SCS criterion (Problem (2)) is equivalent to the solution to the following problem

$$\max_{\boldsymbol{\phi}}\ \left\{\sum_{v\in\mathcal{V}} U_\alpha^v(\boldsymbol{\phi}^v;\mathbf{q}^v):\sum_{c\in\mathcal{C}_r} d_c^r\phi_c\leq 1,\quad\forall r\in\mathcal{R}\right\}. \tag{8}$$

We shall explore two key desirable properties for a sharing criterion, namely, *protection* and *envyfreeness*. In our setting, protection means that no slice is penalized under $\alpha-$SCS sharing vs. static partitioning, where each resource is provisioned across slices in proportion to their shares. Envyfreeness means that no slice is motivated to swap its resource allocation with another slice with a smaller share. These two properties together motivate the choice of $\alpha-$SCS sharing, and at least partially purchasing a larger share in order to improve performance.

B. Protection

Formally, let us characterize protection among slices by how much performance deterioration is possible for a slice when switching from static partitioning to $\alpha-$SCS sharing. Note that under static partitioning, slices are decoupled, so inter-slice protection is achieved possibly at the cost of efficiency. To be specific, the rate allocation for slice v under static partitioning is given by the following problem.

$$\max_{\boldsymbol{\phi}^v}\ \left\{U_\alpha^v(\boldsymbol{\phi}^v;\mathbf{q}^v):\sum_{c\in\mathcal{C}_r^v} d_c^r\phi_c\leq s_v,\quad\forall r\in\mathcal{R}\right\}. \tag{9}$$

From now on, for a given α, let us denote the rate allocation for slice v under $\alpha-$SCS by $\boldsymbol{\phi}^{v,S}:=(\phi_c^S:c\in\mathcal{C}^v)$, and that under static partitioning by $\boldsymbol{\phi}^{v,P}:=(\phi_c^P:c\in\mathcal{C}^v)$. The parameter α is suppressed when there is no ambiguity. The following result demonstrates that $1-$SCS achieves inter-slice

protection in that each slice individually achieves a better utility under $\alpha-$SCS as compared to static slicing.

Theorem 1. *For given* $\mathbf{q}$*, when the resource allocation is performed according to* $1-$*SCS, slice* v*'s utility exceeds that under static partitioning (Problem (9)), i.e.,*

$$U_1^v(\boldsymbol{\phi}^{v,P};\mathbf{q}^v) \leq U_1^v(\boldsymbol{\phi}^{v,S};\mathbf{q}^v). \tag{10}$$

Remark: It is a straightforward to see that under $1-$SCS, the global utility $\sum_{v\in\mathcal{V}} U_1^v(\boldsymbol{\phi}^v;\mathbf{q}^v)$ is improved since it can be viewed as relaxing the system constraints. However, Theorem 1 asserts that this holds on a per slice basis.

A similar result can be shown for general α and is provided in the extended version of this work [33]. Roughly speaking, for general α, the utility deterioration depends on the user distribution across classes. If a slice's user distribution is aligned with that of the overall system, it is guaranteed to see a utility gain under $\alpha-$SCS. Also, for general α the gap cannot be arbitrarily bad.

C. Envyfreeness

Formally, envyfreeness is defined under the assumption that, for two slices v and v', if they swap their allocated resources, slice v's associated utility will not be improved if $s_{v'} \leq s_v$. Before swapping, the rate allocation for slice v is given by $\boldsymbol{\phi}^{v,S}$, while after swapping with slice v', its rate allocation is determined by solving following problem:

$$\max_{\boldsymbol{\phi}^v} \ \{U_\alpha^v(\boldsymbol{\phi}^v;\mathbf{q}^v) : \sum_{c\in\mathcal{C}_r^v} d_c^r\phi_c \leq \sum_{c\in\mathcal{C}_r^{v'}} d_c^r\phi_c^S, \ \forall r\in\mathcal{R}\}.$$

Note that $\sum_{c\in\mathcal{C}_r^{v'}} d_c^r\phi_c^S$ corresponds to the fraction of resource r provisioned to slice v' under SCS. Let us denote the solution to such problem for slice v as $\boldsymbol{\phi}^{v\leftrightarrow v'}$. Then we have the following result for the case with $\alpha = 1$.

Theorem 2. *The difference between the utility obtained by slice* v *under* $1-$*SCS with SCWA, and that under static partitioning with the resource provisioned to another slice* v' *is upper-bounded by the difference between their shares, i.e.,*

$$U_1^v(\boldsymbol{\phi}^{v\leftrightarrow v'};\mathbf{q}^v) - U_1^v(\boldsymbol{\phi}^{v,S};\mathbf{q}^v) \leq s_{v'} - s_v.$$

Thus, a slice has no incentive to swap its allocation with another with a less or equal share, which implies our global sharing is envyfree. Envyfreeness indicates that $1-$SCS achieves desirable resource utilization in that the right portion of resources is provisioned to each slice.

D. Using $\infty-$SCS as a surrogate for $1-$SCS

From the previous discussions, one can see that it is of particular interest to use $1-$SCS as the fairness criterion, as it achieves both strict protection and envyfreeness. When $\alpha = 1$, $\alpha-$SCS becomes weighted proportional fairness, whose solution usually involves iterative methods, and the complexity increases rapidly with the number of user classes as well as the accuracy requirement, see, e.g., [25], making it hard to implement in large-scale. In comparison, weighted max-min is relatively easy to implement in distributed manner, see [19] for example. Specifically a progressive water-filling algorithm [25] has $O(|\mathcal{C}|\max_{c\in\mathcal{C}}|\mathcal{R}_c|)$ complexity. Thus, in our work we will discuss the feasibility of using $\infty-$SCS, which is equivalent to a (dynamically) weighted maxmin, as a surrogate to $1-$SCS. If the resulting utility function is not far from the optimum of the $1-$SCS criterion, we shall assert $\infty-$SCS achieves similar performance as $1-$SCS. We consider a global utility function as the sum of $1-$SCS utility functions across $v\in\mathcal{V}$, given by

$$\Psi(\boldsymbol{\phi};\mathbf{q}) := \sum_{v\in\mathcal{V}} U_1^v(\boldsymbol{\phi}^v;\mathbf{q}^v), \tag{11}$$

then for the overall utility achieved we have following theorem.

Theorem 3. *For a given class weight allocation* $\mathbf{q}$*, if* $d_c^r \geq 1, \forall r\in\mathcal{R}, c\in\mathcal{C}$*, we have*

$$\Psi(\boldsymbol{\phi}^{*,1};\mathbf{q}) - \Psi(\boldsymbol{\phi}^{*,\infty};\mathbf{q}) \leq \sum_{c\in\mathcal{C}} q_c D_c - 1, \tag{12}$$

where $\boldsymbol{\phi}^{*,\alpha} := (\phi_c^{*,\alpha} : c\in\mathcal{C})$ *is the optimal rate allocation under* $\alpha-$*SCS, and* $D_c := \sum_{r\in\mathcal{R}_c} d_c^r$.

Remark: First note that the condition $d_c^r \geq 1$ can be easily satisfied by rescaling the unit of rate without loss of generality. Also by rescaling, one can show that such bound vanishes when each user class is associated with only one resource, i.e., $|\mathcal{R}_c| = 1, \forall c\in\mathcal{C}$, and d_c^r are the same, e.g., $d_c^r = 1$. The bound implies that, the suboptimality due to using a surrogate solution to achieve weighted proportional fairness depends on the diversity in the users' requirements across resources. Also, this suboptimality gap cannot be arbitrarily bad because under SCWA, we have $\sum_c q_c = 1$, thus the right hand side of Eq. (12) is upper-bounded by $\max_c D_c - 1$.

Also, note that by averaging utility observed across time instants, one can easily extend the results in this section to a dynamic setting in the time average sense as long as the user dynamics are not impacted by the resource allocation, e.g. inelastic and/or rate-adaptive users seeing acceptable performance.

IV. Elastic Traffic Model

A. System model

In this section we switch gears to study a scenario where the user traffic is elastic, i.e., each user carries a certain amount of work and leaves the system once it is finished. Specifically, for a class-c user, we assume that its service requirement is drawn from an exponential distribution with mean $\frac{1}{\mu_c}$ independently, and its arrival follows a Poisson process with intensity ν_c. Then the traffic intensity associated with user class c is given by $\rho_c = \frac{\nu_c}{\mu_c}$.

Let us first consider a given time instant, when the cardinalities of $\mathcal{U}_c$ and $\mathcal{U}^v$ are given by n_c and n^v respectively. Also, for simplicity we assume equal intra-slice weight allocation, thus $q_c = \frac{s_{v(c)}n_c}{n^{v(c)}}$. Substituting q_c into Problem (2), the $\alpha-$SCS criterion can be rewritten as follows.

$$\begin{aligned} \max_{\boldsymbol{\phi}} \quad & \sum_{c\in\mathcal{C}} \left(\frac{s_{v(c)}n_c}{n^{v(c)}}\right)^\alpha \frac{(\phi_c)^{1-\alpha}}{1-\alpha}, \\ \text{such that} \quad & \sum_{c\in\mathcal{C}_r} \phi_c d_c^r \leq 1, \ \forall r\in\mathcal{R}. \end{aligned} \tag{13}$$

B. Stability

Problem (13) characterizes the rate allocation across classes when the number of users in the network is fixed. However, it is natural to study the evolution of the system under stochastic loads. Note that while [4] studied the stability condition for $\alpha-$fairness when weights are introduced, their weights do not depend on the dynamic number of users in the network. By

using the fluid models to study stability as established in [15], [14] and [13], one can show that $\alpha-$SCS stablizes the system as long as no resource is overloaded.

Theorem 4. *Consider a set of slices sharing resources under the $\alpha-$SCS criterion with equal intra-slice weight allocation, then when the following effective load conditions are satisfied:*

$$\sum_{c\in\mathcal{C}_r} \rho_c d_c^r < 1, \ \forall r \in \mathcal{R}, \tag{14}$$

the network is stable.

Remark: The result in [4] is similar to the above under the assumption that each user has a fixed weight. Thus the *overall* resources committed to a slice increase with the number of the active users on the slice, up to the full capacity of the resources it requires. In other words, it achieves full-level allocation elasticity, by possibly compromising inter-slice protection. Theorem 4 shows that even when inter-slice protection is maintained, $\alpha-$SCS will still stablize the system.

V. Simulation results

One might think that by introducing inter-slice protection, $\alpha-$SCS effectively imposes additional constraints to the service discipline, thus is compromised in users' performance. However, this need not to be true, as we will demonstrate in this section. We compare the performance of $\alpha-$SCS resource allocation versus several benchmarks, including:

1) Dominant Resource Fairness (DRF) [19], which is a variation of weighted maxmin fairness where users' weights are associated with their resource demands. Here to incorporate network slicing, we adopt a variation where a user's weight also reflects equal intra-slice weight allocations, i.e., $w_u = \frac{s_v}{n_v} \cdot \delta_u, u \in \mathcal{U}^v$, where δ_u is the inverse *dominant share* of user u given by $\delta_u := \frac{1}{\max_{r\in\mathcal{R}} d_c^r}, u \in \mathcal{U}_c$.
2) Discriminatory Processor Sharing (DPS) [2], [12]. To address the multi-resource setting, we adopt a version of DPS which again is a variation of maxmin fairness where user u's weight is $w_u = s_v, u \in \mathcal{U}^v$, without comprising a notion of per-slice share constraint and inter-slice protection.

DRF and DPS are two representatives of weighted max-min fair resource sharing policies, with different heuristics begin invoked to assign weights. Other policies, including BMF [5] and Proportionally Fair, might not be practical under multi-resource cases due to excessive computational complexity, Thus are not included in our comparison. Also note that because SCS might be hard to scalably compute for general α, we propose the use of $\infty-$SCS, as a surrogate resource allocation scheme.

In our simulations, we focus on two performance metrics: mean delay and mean throughput of a typical user. Let the random variable D denote the delay/sojourn time perceived by a typical user, and L its service requirement. The mean delay and mean throughput are defined as $E[D]$ and $E[\frac{L}{D}]$, respectively. The performance of different sharing schemes were compared in a range of settings, from a simple single-resource setting, to a more complex multi-resource network setting where different services/tasks are coupled together through shared resources.

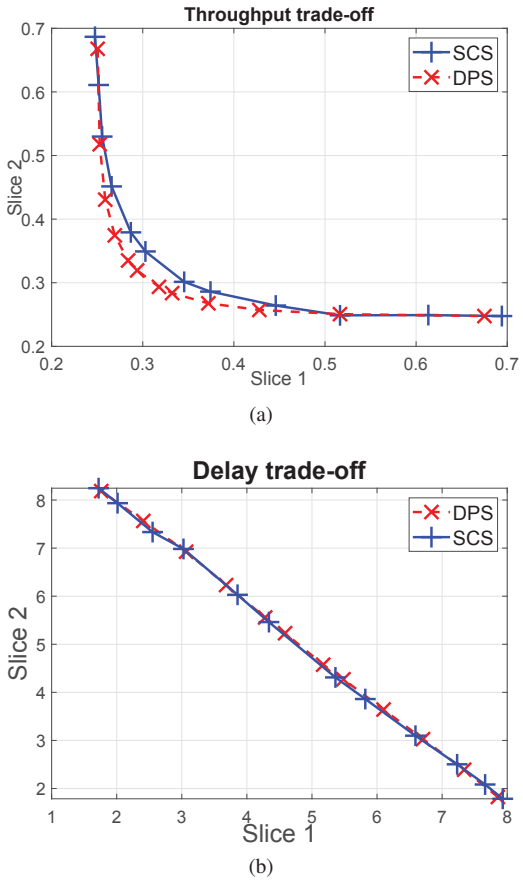


Fig. 2: Performance trade-offs of single-resource case

A. Single-resource case

Since for more complicated network setup, the system performance (for example, processing rate) is often determined by the resource allocation at 'bottleneck' resources, we first consider single-resource setting. Note that, under such circumstances, SCS coincides with General Processor Sharing (GPS) [29] as well as DRF because all classes of users $c \in \mathcal{C}$ are associated with the same resource, and have the same demand.

We consider a simple scenario where $|\mathcal{V}| = 2$, and each slice only supports one user class, so in this setting, a user class corresponds to a slice. Two slices share one resource, referred to as Resource 1 with capacity 1, and $d_1^1 = d_2^1 = 1$. Their traffic models are assumed to be symmetric, with mean arrival rates $\nu_1 = \nu_2 = 0.45$ and mean workloads $\frac{1}{\mu_1} = \frac{1}{\mu_2} = 1$. Their shares, however, are tuned to achieve different performance trade-offs. The share of Slice 1, s_1, ranges from 0.01 to 0.99, while $s_2 = 1 - s_1$. The achieved mean user perceived delay and throughput are illustrated in Fig. 2. One can see that the average delays are marginally better under $\infty-$SCS, and that it outperforms DPS in terms of the average throughput. For example, when two slices have the same share $s_1 = s_2 = 0.5$, SCS increases the throughput of users on both slices by $\sim$10%. This phenomenon was widely observed under different traffic assumptions, for example, when arrival processes are nonsymmetric, and/or workloads have general distributions. See the extended version [33] for details.

This observation, may be surprising, in the sense that enforcing protection might be seen as compromising performance. We note, however, that even when average delays are the same, perceived throughput can vary significantly under different

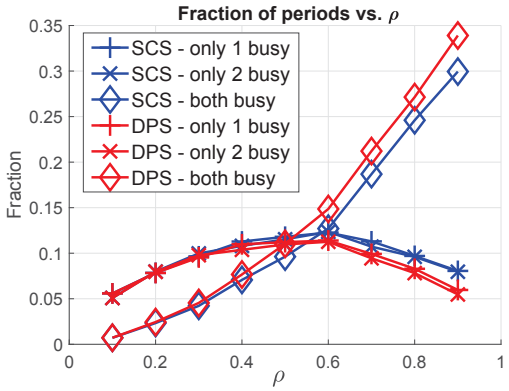


Fig. 3: Busy period of slice 1 and 2 vs. load intensity.

sharing criteria. For example, assuming there are Users 1, 2 and 3, with workloads $L_1 = 1, L_2 = 2$ and $L_3 = 3$, respectively. If a sharing criterion achieves $D_1 = 1, D_2 = 2$ and $D_3 = 3$, it yields mean throughput $E[\frac{L}{D}] = 1$ and mean delay $E[D] = 2$. If another criterion achieves $D_1 = 3, D_2 = 2, D_3 = 1$, it yields mean throughput $\frac{13}{9}$ with the same mean delay. Roughly speaking, accelerating jobs with short residual time would reflect an improvement in the mean throughput. We conjecture that due to the inter-slice protection built into SCS, under stochastic traffic, the slices with fewer customers, which are usually those with shorter residual times, will tend to see higher processing rate than other sharing criterion, as a result, the customers leave the system faster. Overall, SCS tends to "separate" the busy periods of slices, so they tend to be non-overlapping, and the level of inter-slice contention is reduced. We validated this conjecture by measuring the busy period under the symmetric traffic pattern, where the arrival rates of both slices are the same, and are tuned from 0.05 to 0.45, with $s_1 = s_2 = 0.5$. Other parameters are the same as in the setting in Fig. 2. We plot the fraction of times when there is only one busy slice and both slices are busy, vs. the overall traffic load $\rho = \frac{\nu_1}{\mu_1} + \frac{\nu_2}{\mu_2}$ in Fig. 3. One can see that, for both SCS and DPS, the time fraction when both slices are busy increases with ρ, and that when only one slice is busy first increases when ρ is low due to underutilization, but decreases when ρ is high because the inter-slice contention becomes inevitable. However the time fraction when both are busy is always smaller under SCS than that under DPS.

B. Multiple-resource case

We also test the performance of SCS under a more complex setting where a simple cellular networks with both fronthaul and backhaul resources are simulated.

Let us consider a setting with 6 fronthaul resources, 3 backhaul resources, and a cloud computing resource. This system supports two slices, each containing 3 user classes. Slice 1 includes Classes 1,2 and 3, while Slice 2 includes Classes 4, 5 and 6. The association between user classes and resources is demonstrated in Fig. 4, and the demand vectors, as well as the arrival rates and mean workloads, are given in Table II.

Slice 1's share ranges from 0.1 to 0.9, while $s_2 = 1 - s_1$. The achieved performance trade-offs under different sharing criteria are illustrated in Fig. 5. One can see that both SCS and DRF outperform DPS throughput, with similar mean delays under all 3 criteria. Similar results are observed in a range of settings with different traffic patterns and resource demands. For the results in Fig. 6, we adjust the weighting schemes used in DRF by

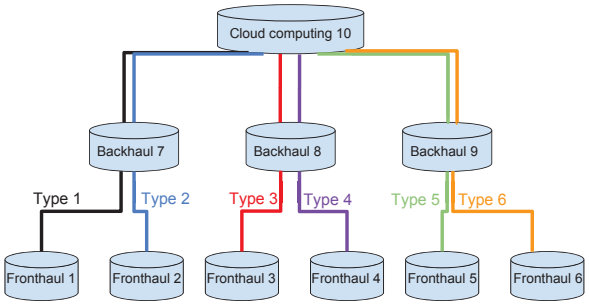
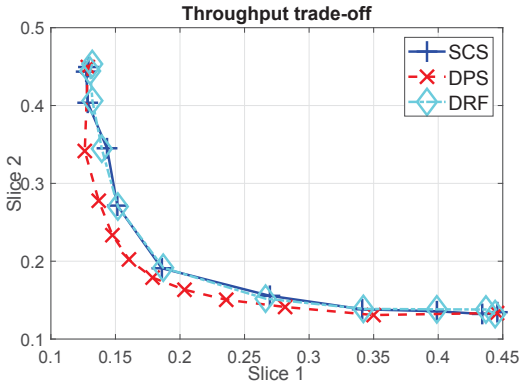
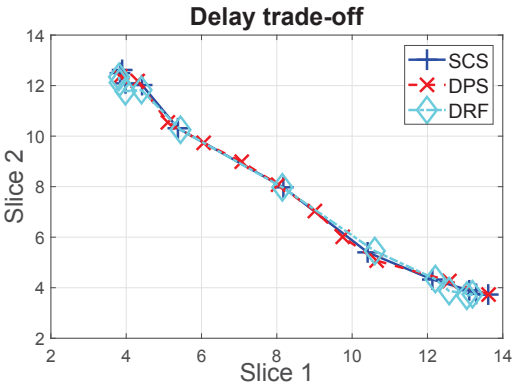


Fig. 4: Association between user classes and resources.



(a)



(b)

Fig. 5: Performance trade-offs of multi-resource case.

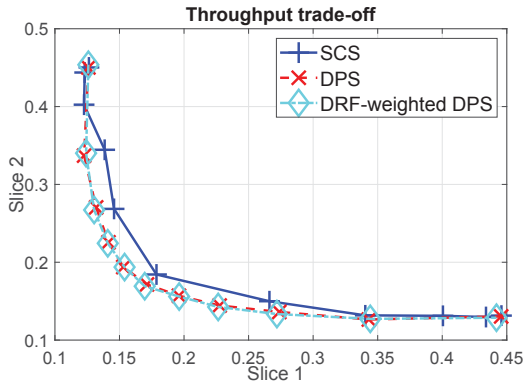
User class	Demand vector	Mean workload	Arrival rate
Class 1	$(\frac{5}{6}, 0, 0, 0, 0, 0, 0.5, 0, 0, 0.217)$	1	0.7
Class 2	$(0, \frac{5}{6}, 0, 0, 0, 0, 0.5, 0, 0, 0.217)$	1	0.7
Class 3	$(0, 0, 1, 0, 0, 0, 0, 0.625, 0, 0.217)$	1	0.7
Class 4	$(0, 0, 0, 1, 0, 0, 0, 0.625, 0, 0.217)$	1	0.7
Class 5	$(0, 0, 0, 0, 1, 0, 0, 0, 0.625, 0.217)$	1	0.7
Class 6	$(0, 0, 0, 0, 0, 1, 0, 0, 0.625, 0.217)$	1	0.7

TABLE II: Multiple-resource experiment setting.

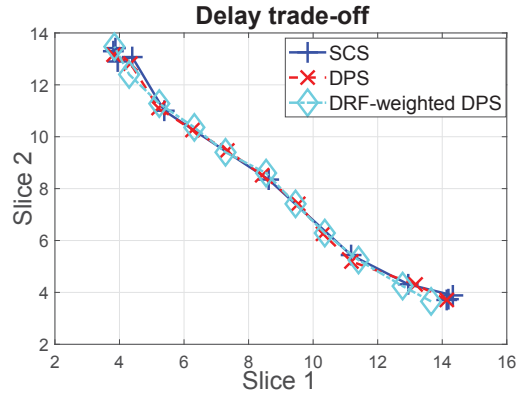
relaxing the share constraints in weight allocation. In particular we let $w_u = s_v \delta_u, u \in \mathcal{U}^v$, and the resources are provisioned according to DPS with weight w_u. The results show that without SCWA, DRF is similar to DPS in both throughput and delay. Therefore, we can conclude that SCWA is the root cause of the improved throughput, and SCS can even improve the system performance while providing inter-slice protection.

VI. Conclusions and Future Work

This paper has explored a new criterion to resource allocation in the context of network slicing for distributed coupled



(a)



(b)

Fig. 6: Performance trade-offs with DRF-weighted DPS.

resources–$\alpha-$SCS, which provides inter-slice protection, load-driven elasticity and good performance for elastic users at the same time. SCS can be viewed as a key to enabling low-complexity performance management in network slicing, by exposing *network shares* to slice operators/tenants, as a high-level resource management interface. Finally, the share abstraction provides a simple parametric "crude" model for slice-level resource allocation which needs to interact with an intra-slice performance management strategy. This approach can be further extended in two directions: i) if slices have highly imbalanced spatial user distributions, it might be useful to let slices specify different shares across different pools of resources, e.g., regions corresponding to downtown, stadium and/or rural area, see, e.g., [10]; and ii) slices may wish to request different shares across types of resources, e.g., a slice may specify a higher share of computational resource pool than that of the communicational resources. For example, mobile cloud gaming is computation intensive, thus the operator might want to reserve more computing resources than connectivity.

Acknowledgment

This work was partially supported by NSF grant CNS-1731658.

References

[1] A. Ahmed and E. Ahmed. A survey on mobile edge computing. In *2016 10th International Conference on Intelligent Systems and Control (ISCO)*, pages 1–8, Jan 2016.

[2] E. Altman, K. Avrachenkov, and U. Ayesta. A survey on discriminatory processor sharing. *Queueing systems*, 53(1-2):53–63, 2006.

[3] A. Banchs. User fair queuing: fair allocation of bandwidth for users. In *IEEE INFOCOM*, volume 3, pages 1668–1677. IEEE, 2002.

[4] T. Bonald and L. Massoulié. Impact of fairness on internet performance. In *ACM SIGMETRICS Performance Evaluation Review*, volume 29, pages 82–91. ACM, 2001.

[5] T. Bonald and J. Roberts. Multi-resource fairness: Objectives, algorithms and performance. In *ACM SIGMETRICS Performance Evaluation Review*, volume 43, pages 31–42. ACM, 2015.

[6] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, pages 13–16. ACM, 2012.

[7] S. Brânzei, Y. Chen, X. Deng, A. Filos-Ratsikas, S. K. S. Frederiksen, and J. Zhang. The fisher market game: Equilibrium and welfare. 2014.

[8] P. Caballero, A. Banchs, G. de Veciana, and X. Costa-Pêrez. Multi-tenant radio access network slicing: Statistical multiplexing of spatial loads. *IEEE/ACM Transactions on Networking*, PP(99):1–15, 2017.

[9] P. Caballero, A. Banchs, G. de Veciana, and X. Costa-Pérez. Multi-tenant radio access network slicing: Statistical multiplexing of spatial loads. *IEEE/ACM Transactions on Networking*, 25(5):3044–3058, Oct 2017.

[10] P. Caballero, G. de Veciana, A. Banchs, and X. Costa-Perez. Optimizing Network Slicing via Virtual Resource Pool Partitioning (Extended). https://www.dropbox.com/s/hso0yqupo5wfwwl/Extended_VRPP.pdf.

[11] Y. Cao, H. Song, O. Kaiwartya, B. Zhou, Y. Zhuang, Y. Cao, and X. Zhang. Mobile edge computing for big-data-enabled electric vehicle charging. *IEEE Communications Magazine*, 56(3):150–156, March 2018.

[12] N. Chen and S. Jordan. Throughput in processor-sharing queues. *IEEE Transactions on Automatic Control*, 52(2):299–305, 2007.

[13] J. Dai et al. A fluid limit model criterion for instability of multiclass queueing networks. *The Annals of Applied Probability*, 6(3):751–757, 1996.

[14] J. G. Dai. On positive harris recurrence of multiclass queueing networks: a unified approach via fluid limit models. *The Annals of Applied Probability*, pages 49–77, 1995.

[15] J. G. Dai and S. P. Meyn. Stability and convergence of moments for multiclass queueing networks via fluid limit models. *IEEE Transactions on Automatic Control*, 40(11):1889–1904, 1995.

[16] G. de Veciana, T.-J. Lee, and T. Konstantopoulos. Stability and performance analysis of networks supporting elastic services. *IEEE/ACM Transactions on Networking*, 9(1):2–14, 2001.

[17] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[18] M. Feldman et al. The proportional-share allocation market for computational resources. *IEEE TPDS*, 20(8):1075–1088, 2009.

[19] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica. Dominant resource fairness: Fair allocation of multiple resource types. In *Nsdi*, volume 11, pages 24–24, 2011.

[20] T. Guo and R. Arnott. Active lte ran sharing with partial resource reservation. In *Vehicular Technology Conference (VTC Fall), 2013 IEEE 78th*, pages 1–5. IEEE, 2013.

[21] T. Lan, D. Kao, M. Chiang, and A. Sabharwal. An axiomatic theory of fairness in network resource allocation. In *2010 Proceedings IEEE INFOCOM*, pages 1–9, March 2010.

[22] J. Mo and J. Walrand. Fair end-to-end window-based congestion control. *IEEE/ACM Transactions on Networking*, 8(5):556–567, Oct 2000.

[23] H. Moulin. *Fair division and collective welfare*. MIT press, 2004.

[24] N. Nisan, T. Roughgarden, E. Tardos, and V. V. Vazirani. *Algorithmic game theory*, volume 1. Cambridge University Press Cambridge, 2007.

[25] M. Pióro and D. Medhi. *Routing, flow, and capacity design in communication and computer networks*. Elsevier, 2004.

[26] M. Satyanarayanan. The emergence of edge computing. *Computer*, 50(1):30–39, 2017.

[27] O. Soliman, A. Rezgui, H. Soliman, and N. Manea. Mobile cloud gaming: Issues and challenges. In *International Conference on Mobile Web and Information Systems*, pages 121–128. Springer, 2013.

[28] R. Srikant and L. Ying. *Communication networks: an optimization, control, and stochastic networks perspective*. Cambridge University Press, 2013.

[29] M. J. van Uitert. *Generalized processor sharing queues*. Eindhoven University of Technology, 2003.

[30] H. R. Varian. Equity, envy, and efficiency. 1973.

[31] H. P. Young. *Equity: in theory and practice*. Princeton University Press, 1995.

[32] J. Zheng, P. Caballero, G. de Veciana, S. J. Baek, and A. Banchs. Statistical multiplexing and traffic shaping games for network slicing. *IEEE/ACM Transactions on Networking*, 26(6):2528–2541, Dec 2018.

[33] J. Zheng and G. de Veciana. Elastic multi-resource network slicing: Can protection lead to improved performance (extended). https://arxiv.org/abs/1901.07497, 2019.