Channel-Aware 5G RAN Slicing with Customizable Schedulers

Yongzhou Chen[1], Ruihao Yao[1], Haitham Hassanieh[2], Radhika Mittal[1]
[1]UIUC, [2]EPFL

Abstract. This paper focuses on 5G RAN slicing, where the 5G radio resources must be divided across slices (or enterprises) so as to achieve high spectrum efficiency, fairness and isolation across slices, and the ability for each slice to customize how the radio resources are divided across its own users. Realizing these goals requires accounting for the channel quality for each user (that varies over time and frequency domain) at both levels – inter-slice scheduling (i.e. dividing resources across slices) and enterprise scheduling (i.e. dividing resources within a slice). However, a cyclic dependency between the inter-slice and enterprise schedulers makes it difficult to incorporate channel awareness at both levels. We observe that the cyclic dependency can be broken if both the inter-slice and enterprise schedulers are greedy. Armed with this insight, we design RadioSaber, the first RAN slicing mechanism to do channel-aware inter-slice and enterprise scheduling. We implement RadioSaber on an open-source RAN simulator, and our evaluation shows how RadioSaber can achieve 17%-72% better throughput than the state-of-the-art RAN slicing technique (that performs channel-agnostic inter-slice scheduling), while meeting the primary goals of fairness across slices and the ability to support a wide variety of customizable enterprise scheduling policies.

1 Introduction

Network slicing is one of the key new features introduced in the 5G standards [3, 9, 37]. It refers to dividing network resources among different services or groups of users to create virtual customizable networks. Such virtualization enables cellular networks to expand beyond the classical "individual mobile user" use-case to a more general "groups of users" use-case which can support new applications with different requirements. These groups of users (typically referred to as enterprises in 5G) enter into service level agreements (SLA) with the network operator, which provides two features: (1) It governs the type of service and the total amount of resources allocated to each slice. For example, it can provide an ultra-reliable low-latency communication for first responders, connected vehicles, or hospitals performing remote surgeries [20]. It can provide cheap and scalable IoT connectivity for farmers using sensors to monitor crops and cities deploying sensors for traffic or air quality monitoring [18]. It can also provide high throughput connectivity for companies with multiple users as well as educational and training institutions using VR/AR [36]. (2) It allows each slice (or enterprise) to customize their virtual networks and dynamically manage their resources [22, 41]. For example, a farming enterprise might want to give higher bandwidth to drones collecting aerial images of crops as opposed to soil moisture sensors, or a hospital might want to prioritize traffic for critical remote surgeries at different times of the day.

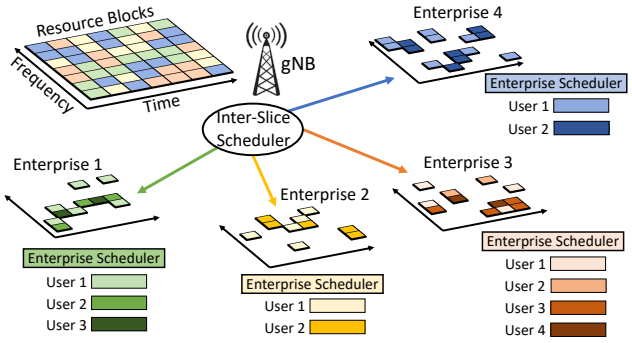


Figure 1: RAN Slicing of resources across 4 enterprises.

Network slicing has two components: 5G RAN (Radio Access Network) slicing and 5G core slicing [11, 29, 30, 44]. This paper focuses on RAN slicing as the RAN is typically the bottleneck in cellular networks [11, 27]. The goal is to divide physical layer resources at the base station (referred to as gNB in 5G) among different enterprises with devices connected to that gNB. These resources include time slots as well as frequency sub-bands used for transmission as shown in Fig. 1. Ideally, the RAN should schedule these resources in a manner that:

(1) Achieves high spectrum efficiency.
(2) Ensures fairness among enterprises subject to their SLAs.
(3) Allows enterprises to customize their scheduling policies.

Realizing the above goals, however, can be challenging in wireless networks because the throughput achieved by using a certain resource block is highly dependent on which user gets that block. In particular, the quality of the wireless channel can change drastically between frequency bands, between users, and over time. This well known phenomena is called frequency selective fading and is shown in Fig. 2 where the capacity can vary by up to 9× across 100 resource blocks (frequency sub-bands) for two users in a 20 MHz LTE bandwidth. Frequency selective fading is even more prominent in 5G where the total bandwidth is expanded to 100 MHz and up to 400 MHz [6]. Allocating resources in a channel agnostic manner can lead to inefficient spectrum usage and unfairness among different slices (enterprises) [10, 22].

Past work has considered the problem of channel aware spectrum allocation [2, 4, 38, 45], channel aware hierarchical resource scheduling in WiMax [21], and RAN virtualization in the context of MVNOs (Mobile Virtual Network

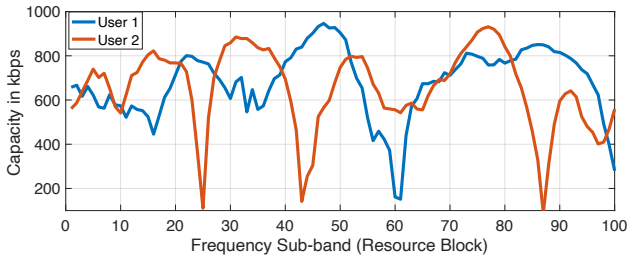


Figure 2: The channel quality across 100 RBs for two users in a 20MHz LTE downlink.

Operators)[1] [9, 22]. However, state-of-the-art techniques can only account for the channel quality between users within the same slice (as detailed in § 2). In other words, only after the inter-slice scheduler allocates resources to different slices irrespective of channel quality, each enterprise can run a channel aware scheduler. Hence, a slice can end up with resources that have bad channel quality for its users which significantly degrades the throughput as we demonstrate in §3.1.

Enabling channel aware scheduling at both the inter-slice scheduler and enterprise scheduler leads to a chicken and egg problem. The enterprise scheduler cannot allocate resources in a channel aware manner before it knows all the resources the inter-slice scheduler will give it, and the inter-slice scheduler cannot allocate resources among slices in a channel aware manner if it does not know to which user in the slice the enterprise scheduler will give a certain resource. One way to break this deadlock is to enumerate through all possible resource allocations and run the enterprise scheduler for each one. However, enumerating all possibilities leads to exponential complexity and is intractable. As a result, state-of-the-art work only uses a channel agnostic inter-slice scheduler [9, 21, 22].

In this paper, we present RadioSaber, a RAN slicing policy that enables channel aware resource allocation at both the inter-slice scheduler and the enterprise scheduler while allowing each enterprise to customize their scheduling policy. RadioSaber's design is based on a simple idea. Since both the inter-slice scheduler and the enterprise scheduler must run at the base station for real-time scheduling, the inter-slice scheduler can use the enterprise scheduling algorithms as subroutines in its algorithm. Both the inter-slice and enterprise scheduler can be channel aware if the inter-slice scheduler can call the enterprise scheduling algorithm with the following query: "If we give resource X to slice A, which user in slice A will get resource X?" If the enterprise scheduler is able to reply to this query independent of what other resources the inter-slice scheduler will allocate to its slice, the inter-slice scheduler can be channel aware. Specifically, the inter-slice scheduler can query the enterprise scheduler of each slice, find the user to which the resource X will be allocated and determine the slice in which X will deliver the best channel quality.

Being able to answer this query, however, limits the space of scheduling policies that an enterprise can run. In particular, the enterprise scheduler must decide how to allocate a resource X independent of the remaining resources that have not yet been allocated to it as we show in § 4.1. Hence, its algorithm must greedily allocate one resource at a time. Most common practical policies, however, like Max. throughput [4], proportional fairness [45], QoS-aware scheduling [2, 4, 38, 39] etc., tend to use greedy algorithms that satisfy the above requirement. It is worth noting here that when allocating a resource X, the algorithms can still account for resources that have already been allocated to the slice in the past (e.g. in order to provide some notion of fairness or demand-awareness), but not account for resources that are yet to be allocated in the future. Hence, RadioSaber is able to accommodate policies that are both channel aware and flexible to the needs of different slices.

Realizing RadioSaber in practice, however, requires addressing algorithmic questions like how do we incorporate the SLA between the operator and the enterprise into the inter-slice scheduler? In what order does the inter-slice scheduler query the resource blocks to decide which slice gets the block? How do we support customizable enterprise scheduling, while restricting the schedulers to be greedy? How does the enterprise scheduler balance between channel quality and other metrics such as flow priorities? We also have to address several system level questions like what is a good and simple interface that we can provide to the enterprises to set their scheduling policies? How do we incorporate RadioSaber into the current 5G standards? We address the above questions in detail in §4.2 and § 4.3.

We have implemented RadioSaber and evaluated it using trace driven simulations. We used an open-source 5G core network [1] and a popular RAN simulator [34] that is capable of simulating both the physical layer and higher layers at the RAN. We evaluate our system using traces of cellular signals [46], that were collected using software defined radios. We compare with:(1) NVS [9, 21, 22], a popular algorithm which enables RAN slicing with channel aware enterprise scheduler and (2) a global channel aware scheduler that schedules all users without slicing.

Our results reveal that RadioSaber is able to outperform NVS, achieving 17%-72% better throughput for backlogged flows, 2× to 4× lower FCT (flow completion time) for non-backlogged flows, and 24× lower packet delays for real-time flows with constant bitrates. Unlike the global channel aware (but slicing unaware) scheduler that fails to provide any isolation across slices, RadioSaber is able to achieve desired isolation and fairness. Finally, RadioSaber is able to accommodate enterprise schedulers with various policies and number of users. Hence, RadioSaber is able to achieve the three goals of spectrum efficiency, fairness, and customizability.

[1]Examples of MVNOs include Straight Talk, Virgin Mobile, & Xfinity mobile, which do not operate their own networks but rather run their traffic through Verizon, AT&T, & T-mobile networks.

The paper makes the following contributions:

- We present the first RAN slicing that is channel-aware both at the inter-slice scheduler and enterprise scheduler.
- We present a new framework for RAN virtualization that abstracts physical layer scheduling and provides an interface for enterprises to set their own schedulers.
- We implement our techniques and demonstrate significant improvement in efficiency and fairness.

2 Background & Related Work

In this section, we provide a brief background on the radio access network (RAN) in cellular networks as well as the related work for channel aware scheduling and RAN slicing.

1. Resource Blocks: In 5G RANs, the user or device is referred to as UE (User Equipment) and the base station is referred to as gNB (next generation Node B). The gNB uses OFDMA (Orthogonal Frequency Division Multiple Access) at its PHY and MAC layers in order to divide radio resources across UEs. In OFDMA, the frequency bandwidth is divided into sub-carrier frequencies that are orthogonal (i.e. do not interfere) and time is divided into equal slots called TTIs (Transmission Time Interval). A resource block (RB), which is the smallest resource unit that can be allocated to a UE, is formed of 12 frequency sub-carriers and 1 TTI slot. Hence, the RBs are organized into a 2D grid as shown in Fig. 1. In practice, however, network operators typically schedule resources in the granularity of resource block groups (RBGs) to minimize control overhead. Each RBG contains a fixed number of consecutive RBs ranging from 1 to 4 [8, 42]. In 4G, each TTI has a fixed length of 1ms and each sub-carrier has a fixed width of 15 kHz. Thus, the RB spans $12 \times 15 = 180$ kHz. 5G, on the other hand, supports 5 configurable TTI and sub-carrier intervals such that the TTI is $2^{-\mu}$ ms and the sub-carrier interval is $2^{\mu} \times 15$ kHz. μ is commonly referred to as the numerology and chosen from the values $0, 1, 2, 3, 4$ depending on the band of operation [7, 35]. For example, the 5G sub-6GHz band, supports sub-carrier width of 60 KHz with a TTI of 0.25 ms for $\mu = 2$ [33, 43]. In this case, the RB spans 720 kHz.

2. Data Rate: The data rate at which a UE can transmit in a given RB depends on the channel quality which is typically defined by the SNR (signal-to-noise-ratio). The SNR determines the capacity of the wireless link and varies across time, RBs, and users as shown in Fig. 2. The SNR can be computed at the UE for each OFDM sub-carrier. For a RB or RBG made of many sub-carriers, the "effective SNR" is typically computed[2] as described in [16, 25, 31]. The effective SNR is then mapped to a discrete value called channel quality indicator (CQI) and periodically reported to the gNB [8]. The CQI is then used to determine the modulation and coding scheme (MCS) for the UE which in turn determines the data rate of the UE. The higher the SNR and channel quality, the higher order MCS can be used to increase the data rate.

[2]The effective SNR is not the average across sub-carriers but rather a weighted exponential average that typically gives a value close to the minimum SNR across sub-carriers. This ensures that the chosen data transmission rate does not exceed the capacity of the wireless link at any sub-carrier which would otherwise result in a very high packet loss rate.

UE	User Equipment	RAN	Radio Access Network
gNB	5G Base station	TTI	Transmission Time Interval
RB	Resource Block	RBG	Resource Block Group
SNR	Signal-to-Noise Ratio	SLA	Service Level Agreement
PF	Proportional Fair	CQI	Channel Quality Indicator
MT	Max. Throughput	MCS	Modulation & Coding Scheme

Table 1: Terms used in 5G networks

3. Channel Aware Scheduling: The need for channel-aware scheduling in wireless networks has led to a number of techniques that propose allocating resources across individual users in a channel-aware manner (e.g. [2, 4, 13, 14, 38, 39, 45]). In most cases, the scheduling problem is NP-Hard and a greedy heuristic is adopted whereby RBs are allocated one at a time to the UE that scores highest on some given metric [4]. These strategies ensure low scheduling overhead and enabling fast decisions at timescales of a single TTI. Some of the most common techniques include:

• *Maximum Throughput (MT):* Assigns the RB to the UE with the largest CQI to maximize data rate irrespective of fairness.
• *Proportional Fair (PF):* Extends MT to incorporate fairness across users by weighing the CQI with the UE's historical allocation. This is a very popular strategy in cellular networks that aims to optimize the sum of the log of throughput of UEs [17, 24, 45]. The PF metric can also be parameterized to vary the relative weights of the CQI versus historical allocation [45].
• *Incorporating QoS and delay:* The PF metric can be further extended to incorporate (i) QoS values that increase the weights of higher priority UEs [4, 13, 14, 39], or (ii) packet delays that increase weights of UEs that have been waiting longer [2, 38, 39].

Non-greedy heuristics have also been proposed for optimizing proportional fairness [17, 24]. The scheduling problem is abstracted as an NP-hard integer linear program and solved using a sub-optimal non-greedy algorithm. Such algorithms, however, are computationally very expensive and must use GPUs to compute their solutions [17].

4. RAN Slicing: The inter-slice scheduler provides each enterprise with a slice of the RAN by allocating a set of virtual RBs which it maps to physical RBs. In order to support channel aware scheduling, it also provides the enterprise scheduler with the CQIs of the UEs of this enterprise for each virtual RB. Each enterprise is then able to customize how it allocates the virtual RBs to its UEs by using virtual control functions at the gNB to specify its own scheduling policy [9, 10]. Note that both the inter-slice scheduler and enterprise scheduler run on the gNB which enables RadioSaber to expose the enterprise scheduling algorithms to the inter-slice scheduler as described in more details in §4.1.

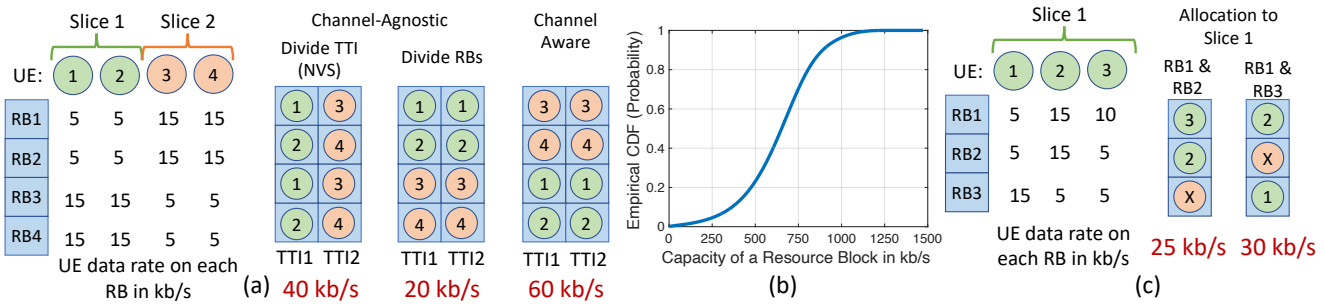


Figure 3: (a) Illustrative example showing the importance of a channel aware inter-slice scheduler. (b) Distribution of the capacity of RBs from real traces. (c) Example showing the challenge in enabling a channel aware inter-slice scheduler.

The closest to our work is NVS (Network Virtualization Substrate) [21], a popular inter-slice scheduler that is used by multiple RAN slicing schemes [9, 21–23, 26]. NVS allocates all RBs in a given TTI to a single slice independent of channel quality. It then rotates between slices in a weighted round-robin manner to satisfy the target throughput of each slice as specified by SLA. While NVS does allow the enterprise scheduler to run a channel aware policy, the inter-slice scheduler remains agnostic to channel quality which changes over time and resource blocks. We compare with NVS in §6 and show that our channel aware inter-slice scheduler can significantly improve performance.

Past work on slicing radio resources also explores supporting dynamic demands across slices [40, 47] and deadlines across slices [12, 15]. All past work, however, constraints the inter-slice scheduler to allocate virtual RBs to each slice in a channel-unaware manner. Our goal, in contrast, is to enable channel-aware scheduling at both inter-slice and enterprise levels, while still giving each enterprise enough flexibility to allocate RBs across its users.

3 Motivation and Challenges

In this section, we will explain, using illustrative examples, the importance of channel aware resource allocation at both the inter-slice scheduler and enterprise scheduler as well as the challenge in making the inter-slice scheduler channel aware.

3.1 Need for Channel-Aware Slicing

To best illustrate the need for channel-aware slicing at both the inter-slice scheduler and enterprise scheduler, consider the toy example in Fig. 3(a). In this example, there are 4 RBs $\{R_1, R_2, R_3, R_4\}$, 2 slices $\{S_1, S_2\}$, and each slice has 2 UEs: $\{u_1, u_2\} \in S_1$ and $\{u_3, u_4\} \in S_2$. The channel quality is shown in the left 2D grid in terms of the data rate each UE can achieve on each RB in kb/s. The enterprise scheduler of both slices are channel-aware and run a proportional fairness (PF) policy in order to maximize throughput while ensuring fairness between UEs.

Consider a channel unaware inter-slice scheduler. The scheduler could allocate all RBs in the first TTI to S_1 and all RBs in the second TTI to S_2 similar to NVS [21]. In this case, the enterprise scheduler of S_1 would allocate $\{R_1, R_3\}$ to u_1 and $\{R_2, R_4\}$ to u_2. Similarly, the enterprise scheduler of S_2 would allocate $\{R_1, R_3\}$ to u_3 and $\{R_2, R_4\}$ to u_4. This allows each UE to achieve 10 kb/s over the two TTIs for a total of 40 kb/s. An alternative channel unaware inter-slice scheduler could have also divided the RBs between the two slices by allocating $\{R_1, R_2\}$ to S_1 and $\{R_3, R_4\}$ to S_2 for all TTIs. In this case, the enterprise scheduler of S_1 would allocate $\{R_1\}$ to u_1 and $\{R_2\}$ to u_2 and that of S_2 would allocate $\{R_3\}$ to u_3 and $\{R_4\}$ to u_4. This allows each UE to achieve a data rate of 5 kb/s for a total of 20 kb/s. Since the inter-slice scheduler is channel-unaware, it has no way of figuring out that such an allocation is very inefficient.

Consider, on the other hand, a channel-aware inter-slice scheduler. This scheduler would allocate $\{R_3, R_4\}$ to S_1 and $\{R_1, R_2\}$ to S_2 for all TTIs. In this case, the enterprise scheduler of S_1 would allocate $\{R_3\}$ to u_1 and $\{R_4\}$ to u_2 and that of S_2 would allocate $\{R_1\}$ to u_3 and $\{R_2\}$ to u_4. This allows each UE to achieve a data rate of 15 kb/s for a total of 60 kb/s. Hence, the channel-aware inter-slice scheduler enables $1.5\times$ to $3\times$ higher throughput.

There are two factors in this example that enable a channel-aware scheduler to achieve better performance than a channel-agnostic scheduler: (i) The channel quality differs across RBs for a given slice, and (ii) The two slices have complementary channel quality distribution across RBs (i.e. slice 1 has a better channel quality for the first two RBs, and slice 2 has better channel quality for the last two RBs). It is the combination of these factors that enables a smarter (channel-aware) resource allocation policy to achieve better performance. Moreover, such factors are quite common in practice as can be seen from the real traces shown in Fig. 2.

While this toy example illustrates the insight behind RadioSaber, the variation in channel quality in real systems can be quite significant as was shown in Fig. 2. Fig. 3(b) plots the cumulative distribution function (CDF) of capacity of all the RBs in a real trace collected from 4G measurements (obtained from [46]). The figure shows that the channel in real traces can vary significantly leading to a capacity that can be as high as $2.2\times$ the median value and as low as $1/20$ of the median value. This diversity which is a result of frequency

selective fading in the wireless channel is expected to further increase in 5G as the bandwidth increases from 10-20 MHz to 100-400 MHz [6].

Our evaluation in §6 shows that due to this diversity, our simple insight from the toy example generalizes to more complex scenarios.

3.2 Challenge in Channel-Aware Slicing

In order to allocate RBs across slices in a channel-aware manner, we first need to know what channel quality each slice would achieve for each RB. While in the above toy example it is easy to see what the best channel aware allocation is, the problem is challenging in more general settings. In particular, if the inter-slice scheduler gives RB R_i to slice S_j, then the channel quality of R_i will depend on which UE belonging to S_j will use R_i which in turn depends on the enterprise scheduling policy. However, the enterprise scheduling policy itself could be channel-aware, in which case, the allocation of R_i to a given UE will depend on which RBs the inter-slice scheduler has allocated to S_j. This creates a deadlock as the inter-slice scheduler needs to know how the enterprise scheduler will allocate R_i to determine its channel quality and whether to give R_i to this slice. At the same time, the enterprise scheduler needs to know what RBs the inter-slice scheduler will give it so it can allocate them in a channel-aware manner.

Fig. 3(c) illustrates this through a toy example. Consider a slice with three UEs $\{u_1, u_2, u_3\}$. Suppose the inter-slice scheduler must allocate two RBs to this slice (as per its weighted share) and suppose the enterprise scheduling policy assigns a RB to the user which has maximum data rate for the RB, while limiting each user's allocation to a single RB. If the inter-slice scheduler allocates $\{R_1, R_2\}$ to the slice, then the enterprise scheduler would first allocate R_2 to u_2 and then allocate R_1 to u_3. However, if the inter-slice scheduler allocates $\{R_1, R_3\}$ to the slice, the enterprise scheduler would allocate R_1 to u_2 and R_3 to u_1. The data rate associated with R_1 for this slice is 10 kbps in the first case and 15 kbps in the second case. Hence, while determining whether to allocate R_1 to this slice or not, the inter-slice scheduler does not know the data rate that R_1 will deliver as it depend on whether the other RB allocated to the slice is R_2 or R_3.

One way out is to enumerate through all possible combinations of inter-slice allocations, and run the enterprise scheduler for each slice for each allocation. However, this is clearly intractable with the number of possible allocations increasing exponentially with the number of slices and resource blocks.

3.3 RadioSaber's Approach

In order to break the deadlock challenge outlined in §3.2, we leverage the following insights. First, both the inter-slice and enterprise scheduler must run on the base station (gNB) to guarantee real-time scheduling. Hence, the inter-slice scheduling algorithm can use the enterprise scheduling algorithm as a subroutine and query it to figure out how it will allocate a certain RB. Second, we can break the deadlock if the enterprise scheduler is able to reply to the following query from the inter-slice scheduler: "If I give resource R_i to slice S_j, which UE in slice S_j will get resource R_i?".

For the enterprise scheduler to be able to reply to this query, its algorithm should determine how to allocate a RB independent of other RBs that the inter-slice scheduler might allocate to it. Restricting the enterprise scheduling algorithm to greedily allocate one RB at a time makes it independent of the remaining RBs that will be allocated to it. It may still need to account for the historical allocation of RBs (i.e. RBs already assigned to the slice) to correctly estimate the remaining demand of a user or to account for fairness. A greedy inter-slice scheduler, that assigns RBs to slices one at a time, enables the enterprise scheduler to update its scheduling state based on its allocation so far. Restricting both scheduler to be greedy thus enables the enterprise scheduler to effectively answer the query while still accounting for historical allocation, and for the inter-slice scheduler to use the result of the query to assign the RB to the slice with the best channel quality.

4 RadioSaber's Design

Objectives. RadioSaber tackles the problem of dividing N RBs (over one or more TTIs) across K slices, and then dividing the RBs allocated to each slice across the UEs within that slice, such that the following objectives are met:

(i) Weighted fairness across slices. Each slice must get its weighted fair share of resource blocks. We assume that the weights are known and are proportional to each slice's demand (based on the SLA between the slice owner and the cellular network operator). Prior work on RAN slicing [22] shows how SLAs can be specified either in terms of number of RBs or overall throughput, and how both can be translated to dynamic per-slice weights. We use these weights to compute a quota of RBs for each slice, as detailed in §4.1.

(ii) High spectrum efficiency. The system must allocate RBs across slices so as to use the spectrum efficiently and achieve high overall throughput. For this, RadioSaber uses the approach outlined in §3.3 to greedily determine which RBs are allocated to a slice to fulfill its quota in a channel-aware manner. We detail RadioSaber's greedy channel-aware inter-slice scheduling algorithm in §4.1.

(iii) Customizable enterprise scheduler. Each slice can have a different policy for dividing the RBs allocated to it across its UEs and flows. The system should provide an expressive interface for slice operators to specify their desired policies, and should be able to enforce them. §4.2 describes RadioSaber's framework for supporting a variety of greedy enterprise schedulers.

After describing individual components of RadioSaber's design, we end this section with describing our overall workflow for RAN slicing in §4.3.

4.1 Inter-slice Scheduler

We divide inter-slice scheduling logic into two steps: (i) computing the quota of RBs for each slice in a TTI, and (ii) greedily allocating RBs to slices as per their quota in a channel-aware manner. [3] We detail these steps below.

Computing Per-slice Quotas. In each TTI, RadioSaber first computes the quota of RBGs (the granularity at which RBs are allocated) for each slice, based on per-slice weights. Let the number of RBs and the number of RBGs in each TTI be $|\mathcal{RB}|$ and $|\mathcal{RBG}|$ respectively. A naive strategy is to simply compute the quota of slice s as $w_s|\mathcal{RBG}|$, where w_s is the normalized weight of the slice. However,there are a few practical considerations: (i) The quota for a slice computed in this manner could be non-integral (and possibly less than 1, depending on the number of other slices and their weights). We cannot have non-integral allocation of RBGs. (ii) If $|\mathcal{RB}|$ is not a perfect multiple of the default number of RBs in each group (say k), then the last RBG will have fewer RBs.

RadioSaber accounts for these aspects by maintaining an offset from the (ideal) target share of RBs for each slice, that rolls over to the next TTI. Algorithm1 presents the pseudo-code. We first compute the target share of each slice (in number of RBs) as its absolute weighted share in a TTI (given by $w_s|\mathcal{RB}|$) subtracted by its rolling offset from the previous TTIs (initialized to zero for a new slice). We then set the quota for the slice (in number of RBGs) as its target share divided by k, and round down the result. Because of the rounding down, the sum of quota across all slices would be less than the available number of RBGs. We then increment the quota of a random set of slices by one, so as to account for all of the extra RBGs. This can result in a few slices getting less than their fair share of RBs, and a few slices getting more. We capture this by updating the offset for each slice. This offset is then taken into account when computing the quotas in the next TTI – the slices that get more than their fair share of RBs in the current TTI will have a positive offset and a lower share in the next TTI, while the slices that get less than their fair share of RBs in the current TTI will have a negative offset and a higher share in the next TTI.

As mentioned above, one of the RBGs allocated to a slice may have fewer than k RBs. We account for this by adjusting the offset of the slice that is allocated the aberrant RBG when assigning RBGs to slices (we skip mentioning this step when discussing our assignment algorithm below).

Assigning RBGs to Slices. Given per-slice quotas, we next need to assign RBGs to slices in a channel-aware manner, so as to maximize spectrum efficiency. Even if we assume that the channel-quality (or the data-rate) associated with each slice for each RBG is known in advance (which, as illustrated in §3, is not the case), computing the optimal assignment of RBGs that maximizes the total data-rate, while adhering to the quota on RBGs for each slice is an NP-hard problem. [4] A greedy heuristic is therefore a natural choice for finding (a potentially sub-optimal) solution to this problem. But more importantly, as discussed in §3.3, a greedy approach allows the inter-slice scheduler to effectively query the enterprise scheduler and determine the channel quality for each RBG.

The basic allocation logic then becomes relative straightforward: In each TTI, RadioSaber greedily picks a RBG and assigns it to the slice that achieves the maximum channel quality for that RBG. Once a slice has been allocated its quota of RBGs, it is no longer considered for subsequent RBGs allocation in that TTI.

The order in which the inter-slice scheduler allocates

Algorithm 1 Calculating RBGs quota for slices

```
1:  variable rbs_offset_
2:  procedure SLICEQUOTA
3:      rbs_share = []
4:      rbgs_quota = []
5:      k ← rbs_per_rbg()
6:      for s in S do
7:          rbs_share[s] ← |RB| × w_s - rbs_offset_[s]
8:          rbgs_quota[s] ← ⌊ rbs_share[s] / k ⌋
9:      end for
10:     extra_rbgs = |RBG| - sum(rbgs_quota)
11:     while extra_rbgs > 0 do
12:         rbgs_quota[S.rand()] += 1
13:         extra_rbgs -= 1
14:     end while
15:     for s in S do
16:         rbs_offset_[s] = rbgs_quota[s] × k - rbs_share[s]
17:     end for
18:     return rbgs_quota
19: end procedure
```

Figure 4: Allocating three RBs to three slices with same weights using different strategies.

[3] RadioSaber follows the standard practice of making radio resource allocation decisions at timescales of a TTI (§2). Consequently, any temporal variations in a UEs CQI is naturally accounted for when recomputing the schedule over each subsequent TTIs.

[4] It reduces to an Integer Linear Programming problem [45].

Algorithm 2 Assigning RBGs to slices

```
1:  procedure RBALLOCATION
2:      rbgs_quota ← sliceQuota()
3:      rbgs_assignment = []
4:      slice_user = []        ▷ Users that each slice decides to schedule for all RBs
5:      slice_cqi = [] ▷ Channel qualities of each slice(based on scheduled users) for all RBs
6:      slice_rbgs = []
7:      for i in range(|RBG|) do
8:          for s in S do
9:              u* ← s.enterpriseScheduler(i)
10:             slice_user[i][s] ← u*
11:             slice_cqi[i][s] ← u*.channelQuality(i)
12:         end for
13:     end for
14:     while rbs_assignment.size() < |RBG| do
15:         i*, s* ← argmax_{i,s} slice_cqi and i not in rbgs_assignment and slice_rbgs[s] < rbgs_quota[s]
16:         u* ← slice_user[s*]
17:         u*.allocRB(i*)
18:         rbgs_assignment[i*] = u*
19:         slice_rbgs[s*] += 1
20:         if slice_rbgs[s*] >= rbgs_quota[s*] then
21:             Continue
22:         end if
23:         for i in range(|RBG|) do
24:             u' ← s*.enterpriseScheduler(i)
25:             slice_user[i][s*] ← u'
26:             slice_cqi[i][s*] ← u'.channelQuality(i)
27:         end for
28:     end while
29: end procedure
```

the RBGs can impact spectrum efficiency. Fig. 4 illustrates this with an example. It shows the channel quality (datarate) associated with three slices $\{s_1, s_2, s_3\}$ for three RBGs $\{R_1, R_2, R_3\}$. Each slice has a quota of 1 RBG. Assigning RBGs sequentially in the order $\{R_1, R_2, R_3\}$ to the slice that has maximum datarate (until it exhausts its quota) results in a total datarate of 50Kb/s. In comparison, allocating RBGs in decreasing order of channel quality achieves a higher datarate of 60Kb/s as it greedily selects the best RBG-slice mapping (in terms of datarate) in each iteration. RadioSaber adopts this strategy for inter-slice scheduling.

While this greedy strategy results in a better outcome than the sequential allocation, it does not produce the optimal result. In fact, the optimal allocation for the example in Fig. 4 would have resulted in a higher datarate of 65kb/s. This suboptimality is expected from any polynomial time algorithm, given the NP-hardness of our resource allocation problem.

Algorithm2 presents the pseudo-code for RadioSaber's inter-slice scheduler. It computes maximum channel quality across slices for each unallocated RBG (lines 7-13). It allocates the RBG with the highest channel quality (say i^*) to the corresponding slice (s^*) that has the highest channel quality for it (line 15). If the quota for slice s^* is not exhausted, it recomputes the channel quality for the remaining RBGs for that s^* (lines 20-24). This recomputation is needed because the previous allocation may influence how the enterprise scheduler in s^* schedules UEs for subsequent RBGs allocated to it (e.g. if a UE has met its demand or to ensure fairness across UEs). The above steps are then repeated to proceed with the allocation of the next RBG.

This algorithm has a polynomial time complexity of $O(|\mathcal{RBG}|^2(|\mathcal{S}|+T)+|\mathcal{RBG}||\mathcal{S}|T)$ in the worst case, where T is the time complexity of the enterprise scheduler for assigning an RBG to a UE (this is typically O($|\mathcal{U}|$) for $|\mathcal{U}|$ UEs in a slice for the greedy enterprise scheduler described in §4.2).

4.2 Customizable Enterprise Scheduling

We now describe RadioSaber's framework for supporting different enterprise scheduling policies. Restricted by the need for the enterprise scheduler to be greedy (§3), and inspired by existing channel-aware wireless schedulers (described in §2), we adopt a metric-based allocation strategy. The enterprise scheduler computes a metric for each UE for the RBG it is assigned (or queried about by the inter-slice scheduler) and selects the UE that scores highest on that metric. These metrics are parameterized – RadioSaber exposes these parameters to the slice operators, allowing them to tune the parameters in order to express different scheduling policies. [5]

At a high-level RadioSaber allows the slice operator to specify a variety of policies based on flow priorities, fairness across UEs, channel-quality, and packet delays. It supports two scheduling paradigms:

Paradigm 1: Select User First. This paradigm assigns a given RBG i to a UE that scores the highest on the following metric (corresponding to the generalized PF [45]), and schedules the highest priority flow belonging to that UE.

$$metric(u,i) = d_{u,i}^{\varepsilon}/R_u^{\psi}$$

Here, $d_{u,i}$ is the instantaneous data rate of UE u for RBG i, and R_u captures the historical RBG allocation for the UE u as an exponential weighted moving average of the user's throughput, based on its datarate for the RBGs it has been assigned so far. The parameters, ε and ψ, are integers that determine the relative weightage of channel quality and historical allocation. For example, setting $\varepsilon = \psi = 1$ instantiates the default

[5] RadioSaber makes an implicit assumption that each slice operator wishes to implement a customized scheduler, and accordingly specifies the scheduling parameters that allow RadioSaber's enterprise scheduling framework to answer the inter-slice scheduler's query and to allocate resources. For slices that do not wish to implement a customized policy, RadioSaber can initialize the scheduling parameters to reflect the default scheduling policy decided by the network operator.

PF(proportional fairness) scheduler, or setting $\varepsilon = 1$ and $\psi = 0$ instantiates the MT(maximum throughput) scheduler.

Paradigm 2: Select Highest Priority First. This paradigm always assigns a given RBG i to the flow with highest priority. When the highest priority level (p) has multiple flows (across multiple UEs), it selects a flow (or a UE) [6] based on the following metric (from [2,4]):

$$metric(u,p,i) = (\beta D_{u,p} + (1-\beta))(d_{u,i}^{\varepsilon}/R_u^{\psi})$$

Here, $D_{u,p}$ is the queuing delay experienced by the packet at the head of the queue corresponding to UE u and priority p. β is a binary parameter that determines whether the head packet delay ($D_{u,p}$) influences the choice of UE. $d_{u,i}$, R_u, ε and ψ are as described above. Setting $\beta = \varepsilon = \psi = 1$ instantiates the M-LWDF(Modified-Largest Weighted Delay First) Scheduler [2].

A binary parameter α allows a slice operator to pick between the two paradigms ($\alpha = 0$ picks the first paradigm and $\alpha = 1$ picks the second paradigm).

In summary, there are four parameters that RadioSaber exposes for tuning the enterprise schedulers:
(1) α that determines whether a UE is selected first or the priority level.
(2) ε that determines the weightage of channel quality.
(3) ψ that determines the weightage of historical allocation.
(4) β that determines whether the head packet delay is a factor.
A slice operator can tune these parameters differently to express different policies, as per their requirements and workload. These parameters are initialized when the slice operator creates the slice, and are stored in the data repository in the 5G core (as detailed in §4.3). [7]

While our enterprise scheduling approach is heavily inspired from existing wireless scheduling techniques, we introduce some new aspects. In particular, cellular schedulers typically do not consider different priority levels for a given UE. Instead, they map all flows belonging to a UE to a single queue (the default radio bearer). We introduce the notion of different priority levels for each UE, as we believe that is an important requirement for emerging applications where each UE may have different types of flows.

4.3 RAN Slicing Workflow

We now explain the RAN slicing workflow for RadioSaber.

[6] We assume each UE has a single flow at each priority level. Even if a UE has multiple flows for a given priority level, they are served in a FIFO order with respect to one another, and we can logically treat them as a single flow.

[7] We assume that the slice operators are fine with sharing their coarse-grained scheduling policies with the network operators (in the form of these parameters). Extending our system to provide a secure environment for the slice operators to schedule their UEs and to answer the inter-slice scheduler's query in manner that does not reveal the scheduling policies is an interesting future direction, that is beyond the scope of this paper.

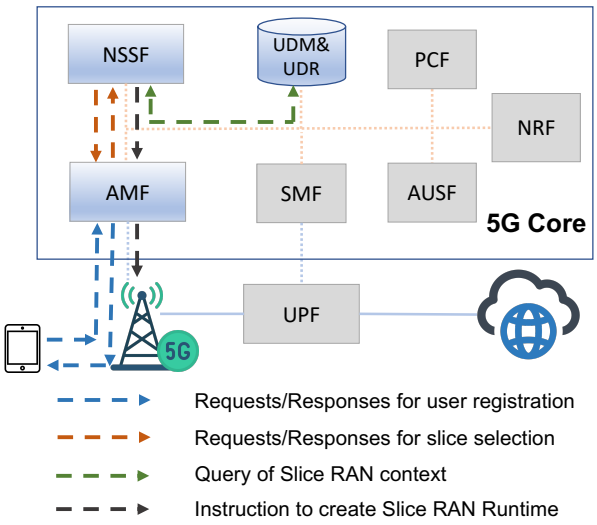


Figure 5: 5G Core network architecture, and the workflow for relaying slice context from 5G core to gNB.

4.3.1 Relaying Slice Context to gNB

The relevant slice context for RadioSaber includes the slice ID, a list of its users, the slice SLA (or weight), and the enterprise scheduling parameters. When the slice operator registers a new slice, this context is initialized and stored in the 5G core. When a user belonging to this slice attaches to a gNB, the relevant context must be relayed from the 5G core to the gNB for RAN slicing. We now describe the workflow for it.

Fig. 5 shows different modules of 5G core. The modules relevant for RadioSaber's workflow are shaded in blue. UDM & UDR (Universal Data Management & Repository) manages and stores all user-related data including slice contexts. AMF (Access and Mobility Function) manages different aspects of a UE (other than data forwarding) – these include connection, reachability, mobility, authentication, authorization, and location services. NSSF (Network Slicing Selector Function) handles slice selection request and manages the life cycle of a network slice instance.

Standard implementations for 5G core provide a framework to support core slicing. We extend it for RAN slicing with RadioSaber by adding the following workflow:
(1) When a UE tries to connect, it issues a registration request to the gNB, which then forwards the request to the AMF.
(2) The AMF in turn issues a request to the NSSF to fetch the slice context associated with the UE.
(3) If the corresponding slice instance is not available at the NSSF, it means that the UE is the first to register for the slice. The NSSF then constructs the slice instance based on the slice context stored in the UDM, and passes the context to the AMF in response. If an instance for the slice is already available at the NSSF, it directly responds to AMF.
(4) The AMF then relays the slice context obtained from NSSF to gNB through the NGAP(NG Application Protocol).
(5) The gNB uses the information received from the AMF

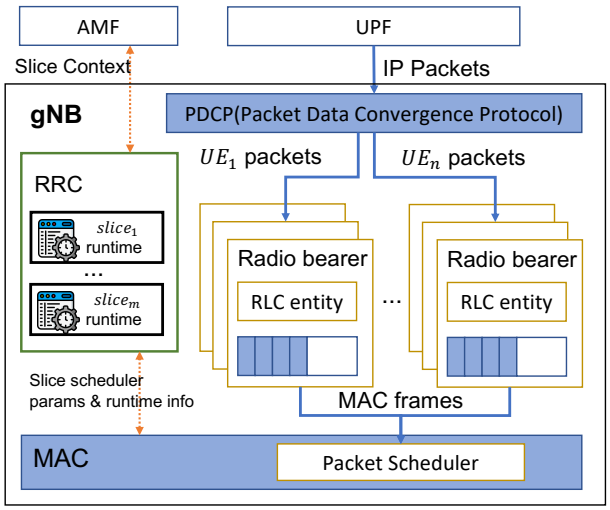


Figure 6: 5G gNB architecture: it shows how RRC maintains slice runtimes, and controls the MAC scheduler behavior.

to create a RAN slice runtime instance if that UE is the first to register for that slice at the gNB. Otherwise, it uses the information to determine the slice ID for the UE, and map the UE to the corresponding RAN slice instance.

4.3.2 Scheduling at gNB

The gNB stores the slice context obtained from the 5G core at the RRC (Radio Resource Control). The IP packets arrive at the gNB from the UPF (User Plane Function), and are intercepted by the PDCP (Packet Data Convergence Protocol). PDCP is responsible for compressing IP headers and ciphering data for integrity protection. For RadioSaber, it parses the packet priority from the DSCP field in the IP header.

Typically, a gNB maintains a single queue for each UE associated with a single QoS level (radio bearer) [32]. To support multiple priority levels within a UE, we simply allow multiple discrete priority queues (radio bearers) for each UE. [8]

Upon parsing the priority level from the packet header, the PDCP can then forward the packets to the radio bearer with the corresponding UE and priority level.

The packet scheduler in the gNB uses the slice context stored in the RRC (that includes slice weights and enterprise scheduling parameters) and the per-UE context (including its periodically updated CQI and historical allocations) to run both the inter-slice and the enterprise schedulers.

5 Implementation

5G Core Support. We extend Open5GS (an open-source 5G core) [1] to add support for the RadioSaber workflow described in §4.3.1. Open5GS already provides a framework for core slicing. We add the serialization and deserialization procedure of our RAN slice context, and the communication between the AMF and gNB for the construction and destruction of RAN slice runtime. This implementation comprises 530 lines of code in total.

RAN Slicing and Scheduling Logic. Due to high overhead and massive cost of deploying a base station and scaling to large number of users, we evaluate our system using trace-driven simulations. We implement RadioSaber's RAN slicing and scheduling logic in an open-source RAN simulator [34]. By default, the simulator was configured to use LTE settings. We extend it to support 5G configuration. In particular, we configure the downlink bandwidth to 100MHz and the TTI to 250μs. We set the guard band to 3920KHz so there are 128 RBs and 32 RBGs in total. We also add support for some practical considerations (e.g. CQI reporting intervals). We ensure that UEs report subband CQI every 40ms, at a granularity that covers 4 RBs. We extend the simulator to support multiple priority levels for each UE. Moreover, we implement multiple scheduling policies in the simulator, including RadioSaber's inter-slice scheduler, the inter-slice scheduling logic in NVS [22], and the enterprise scheduling policies.

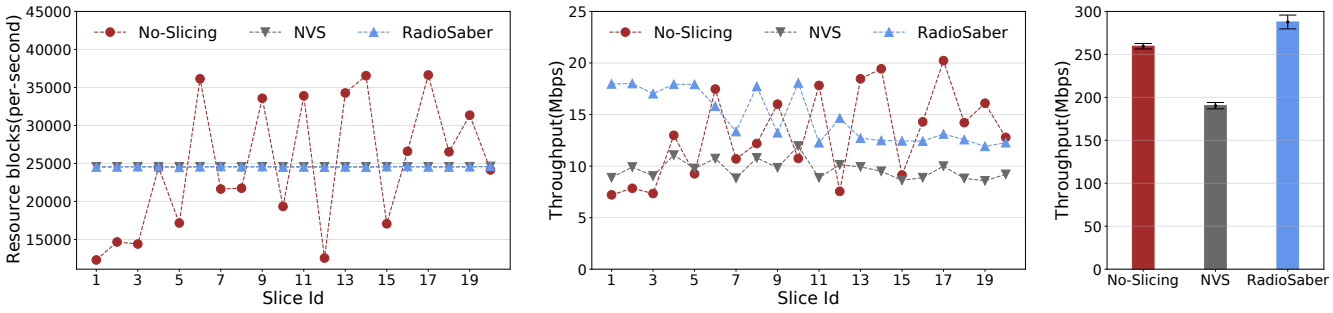
As detailed in §6, we use this simulator to do trace driven simulations to evaluate RadioSaber, using traces from LTScope [46]. These traces measure LTE signal strength (SNR over time and sub-carriers) for major US mobile carriers. We extend those measurements to 5G by concatenating five randomly selected distinct measurements of the 20 MHz LTE bandwidth to generate measurements over a 100 MHz 5G bandwidth.

6 Evaluation

We use the simulator described in §5 to evaluate RadioSaber. Our evaluation results in the following key takeaways:

- RadioSaber achieves higher overall throughput than the state-of-the-art (channel-unaware) inter-slice scheduler, NVS [9, 21, 22] (§6.1).
- RadioSaber can correctly enforce isolation and weighted fairness across slices (§6.1).
- RadioSaber is able to support diverse and customizable enterprise scheduling policies. The throughput wins with RadioSaber over NVS translate to better performance on the corresponding slice-specific metrics (§6.2).
- Complementary CQI distribution across slices is required for channel awareness to produce throughput gains (§6.3).
- RadioSaber's performance benefits hold as we vary the number of slices and UEs/slice (§6.4).
- RadioSaber performs better than NVS with non-greedy PF enterprise schedulers(§6.5).
- RadioSaber performance is very close to our (contrived) upperbound (§6.6).
- The scheduling latency of RadioSaber is within a TTI and the runtime overhead is low (§6.7).

[8] We expect a typical deployment to support 4-8 priority levels.



(a) per-slice RBs per second (b) per-slice throughput (c) sum total throughput

Figure 7: Comparing RadioSaber with NVS and no slicing. Number of slices is fixed to 20, with 5-15 UEs in each slice. The first 10 slices use MT and the next 10 use PF enterprise schedulers (except for no slicing, which uses a global PF scheduler).

6.1 Spectrum Efficiency and Fairness

We first evaluate how RadioSaber can achieve both high spectrum efficiency and fairness between slices. We configure our experiment to use 20 slices, and randomly assign between 5 to 15 UEs to each slice. While we expect there to be hundreds of slices, and hundreds of UEs associated with each slice, only a few of them will be active (and in the geographical vicinity) of a given gNB.

We configure all slices to have the same weight. To evaluate total spectrum efficiency of RadioSaber, we instantiate each UE with a single backlogged flow. The enterprise scheduler-ing policy is configured to follow MT scheduling for the first 10 slices (with $\beta = 0$, $\varepsilon = 1$, and $\psi = 0$), and PF scheduling for the last 10 slices (with $\beta = 0$, $\varepsilon = 1$, and $\psi = 1$).

We compare RadioSaber with two baselines in this experiment: (i) NVS, with a channel-agnostic inter-slice scheduler described in §2, and the same (channel-aware) enterprise schedulering policy for each slice as described above; (ii) No-Slicing, which uses a global PF scheduler to schedule UEs without any notion of slicing.

Fig. 7(a) shows the number of resource blocks allocated to each slice over intervals of 1 second. Both RadioSaber and NVS allocate same shares of radio resources to slices since each slice is configured to have the same weight. In contrast, No-Slicing cannot guarantee the same level of fairness across slices. With No-Slicing, the allocated RBs for a slice is roughly proportional to the number of UEs in the slice.

Fig. 7(b) shows the aggregate average throughput achieved by each slice. We see that, even though NVS and RadioSaber allocate the same number of RBs to each slice, RadioSaber achieves better throughput than NVS for each slice by enabling a better (channel-aware) assignment of RBs to slices. As expected, for both RadioSaber and NVS, the first 10 slices (that use MT schedulers) achieve higher average throughput than the last 10.

Fig. 7(c) shows the overall throughput computed as the total number of bytes transmitted across all UEs and over all TTIs and divided by the total number of TTIs in the experiment run. RadioSaber achieves 51% higher throughput than NVS and 11% higher throughput than No-Slicing. The higher throughput compared to NVS comes from channel-aware inter-slice scheduling with RadioSaber. The higher throughput compared with No-Slicing stems from the first 10 slices applying MT enterprise schedulers as compared to all UEs being scheduled as per the PF policy with No-Slicing (that does not have any notion of customizable per-slice scheduling).

We also evaluated a setting where slices differ in their weights. Both NVS and RadioSaber adhered to the specified per-slice weights when allocating RBs, while No-Slicing could not, and RadioSaber achieved better throughput than NVS. We present these results in Appendix§A.1.

If No-Slicing used MT instead of PF, it would have necessarily achieved a better overall throughput than RadioSaber, but would have failed to provide weighted fairness across users. In contrast, RadioSaber strives to maximize throughput while meeting this basic objective of isolation and fairness across RAN slices.

Slices	Scheduler $(\alpha, \beta, \varepsilon, \psi)$	Traffic generation	Metrics
1-5	PF(0,0,1,1)	a backlogged flow	average throughput
6-10	PF(1,0,1,1)	heavy-tail distributed flows	FCT(Flow Completion Time)
11-15	PF(1,0,1,1)	heavy-tail distributed flows(25% prioritized)	FCT of prioritized flows
16-20	M-LWDF (1,1,1,1)	a 1280kbps real-time video flow	average queueing delay

Table 2: Scheduling configuration, workloads per user, and metrics to evaluate in different slices.

6.2 Diverse Enterprise Schedulers

We now evaluate how RadioSaber supports diverse and customizable enterprise schedulers, and maintains isolation between slices. We compare against the same baselines: NVS and No-Slicing. We consider a total of 20 slices, and randomly assign between 5-15 users to each slice. We group the slices into 4 categories, with 5 slices in each category. Table 2 summarizes the configuration for slices in each category, including the enterprise scheduler, workloads per user, and

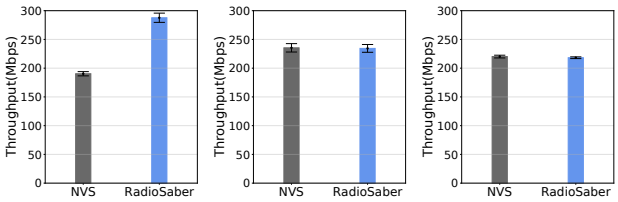
Slices	Metrics	RadioSaber	NVS	No-Slicing
1-5	throughput (Mbps)	13.02	8.45	**17.41**
6-10	average FCT(s)	**2.606**	5.708	5.714
11-15	average FCT(s)	**0.489**	1.686	2.988
16-20	average queueing delay(s)	**0.061**	1.493	0.696

Table 3: Experiment results w.r.t different metrics in all slices of RadioSaber and baselines.

metrics: (i) Slices 1-5 use PF based enterprise scheduling policy. Every user in a slice instantiates a backlogged flow, and we measure the average aggregate throughput as the metric. (ii) Slices 6-10 also use PF based enterprise scheduling policy. Each slice in this category generates flows with Poisson inter-arrival time, arriving at an average rate of 12Mbps, following a heavy-tailed Internet flow distribution [28]. It randomly and evenly assign these flows to its UEs. We measure the FCT(Flow Completion Time) of flows as the evaluation metric. (iii) Slices 11-15 use the same workload generator as the previous category, but assigns a higher priority to 25% of the flows (that are randomly selected). We measure the FCT of prioritized flows as the key metric. (iv) For slices 16-20, each user streams a real-time video flow with 1280kbps average bitrate. Each slice applies the M-LWDF policy for enterprise scheduling, to ensure low packet delays for the real-time streaming. We compute per-packet queuing delays as the key evaluation metric. We configure the weights of slices 16-20 to be 2× higher than the weights of the other slices.

Table 3 summarizes the results (aggregating the slice-specific metrics across all slices in each category). We also present the CDF of different metrics, to highlight the performance differences at different percentiles in the Appendix§A.2.

We find that RadioSaber consistently outperforms NVS across all slice-specific metrics, with 1.5× higher throughput for slices 1-5, 2-4× lower FCT for slices 6-15 and 24× lower packet delays for slices 16-20. This shows that the throughput wins of RadioSaber directly translate to other relevant metrics such as flow completion time and packet delays. No-Slicing achieves higher throughput than RadioSaber for slices 1-5 which have backlogged flows, but fares significantly worse in the performance metrics for other slices (with 2-7× higher FCT in slices 6-15 and 10× higher packet delays in slices 16-20). This is because: (i) No-Slicing cannot provide isolation across slices and correctly enforce weighted fairness. Instead, it ends up allocating a larger share of RBs to the first category of slices (1-5) which have more number of active users at any given time (as they are all backlogged). (ii) No-Slicing's PF policy is not compliant with the requirements of other slices (i.e. prioritization for slices 11-15 and low packet delays for slices 16-20)



(a) Real Trace (b) Synthetic: no variation in sub-band CQI (c) Synthetic: non-complementary subband CQI

Figure 8: Experiments with different types of CQI traces

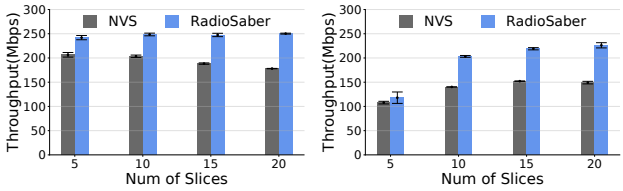
6.3 What Makes RadioSaber Win Over NVS?

We now validate the intuition discussed in §3, that the throughput gain from channel-aware inter-slice scheduling originates from the complementary channel quality distribution between slices. We use the same setting as that in §6.1, except that we generate synthetic CQI reports using the default channel propagation module in our RAN simulator that models losses due to multipath [19], path loss, penetration and shadowing [34]. Fig. 8(a) shows experiment results from real traces for comparison (these results are the same as those presented in §6.1).

In the first experiment, we exclude the multipath loss to make the channel quality same across all RBs (although the channel quality still varies across different UEs). Channel quality variations over time happen at granularity of 40ms (the CQI reporting interval), which translates to 160 TTIs. Fig. 8(b) shows that NVS and RadioSaber both achieve the same throughput. If we consider a 2D grid spanning 20 TTIs (number of slices in our setup), the channel quality for a UE neither varies in the frequency domain, nor in the time domain – so, as long as all slices are assigned RBs as per their quotas, it does not matter which RBs they get assigned.

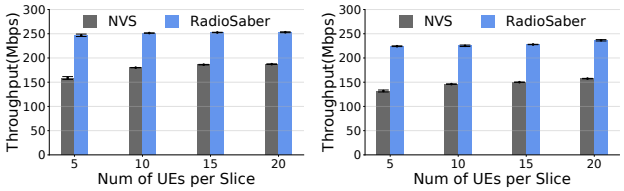
In the second experiment, we synthetically make the sub-band channel distribution non-complementary. For this, we exclude the multipath loss, and manually decrease SNR of the first 50MHz channel by 10dB and increase SNR of the lower 50MHz channel by 10dB. What this implies is that all UEs have equally high channel quality for the first half of the RBs in the frequency domain, and equally low channel quality for the second half. Fig. 8(c) shows that NVS and RadioSaber again achieve similar throughput with this "non-complementary" CQI distribution. This is because, with both NVS and RadioSaber, each RB will produce the same datarate (that is either high or low), no matter which slice/UE it gets assigned to.

These results highlight that RadioSaber wins over NVS when (i) the CQIs differ across UEs in the frequency domain, and (ii) the CQIs for a RB across different UEs complement one another (i.e. if some UEs have low CQIs for a given RB, there are other UEs having high CQIs for it). We found this to be the case for the LTE cellular traces used in our experiments, and expect these trends to be stronger for 5G with larger bandwidths.



(a) backlogged flows (b) heavy-tail distributed flows

Figure 9: Varying the number of slices (5-15 UEs per slice).



(a) backlogged flows (b) heavy-tail distributed flows

Figure 10: Varying the number of UEs per slice for 20 slices.

6.4 Varying Number of Slices and UEs per Slice

We next evaluate the robustness RadioSaber's throughput wins over NVS, as we vary the number of slices and the number of users in each slice. For simplicity of analyzing the results, we configure each slice to apply the PF enterprise scheduling policy. We experiment with two types of workloads: (i) each user maintains a backlogged flow; (ii) the heavy-tail distribution of flows across UEs (as described in §6.2), and an average load of 24Mbps per slice.

Varying number of slices. We increase the number of slices from 5 to 20 and randomly assign between 5-15 users in each slice. In Fig. 9(a), we find that with backlogged flows, the aggregate throughput remains almost unchanged in RadioSaber as we increase the number of slices, but the aggregate throughput keeps dropping in NVS. It shows that the spectrum efficiency of RadioSaber remains unaffected and scales well with the number of slices. RadioSaber achieves 17.2%-40.5% higher throughput than NVS.

We find some interesting trends in Fig. 9(b), with non-backlogged flows. RadioSaber is able to make better use of the spectrum as the number of slices increase, which increases the number of active users (with diverse and complementary subband CQI distributions) that can be scheduled per TTI. NVS does not experience a similar trend, and in fact, has a significantly lower aggregate throughput compared to the backlogged flows scenario in Fig. 9(a). This is because NVS allocates all RBs in a TTI to the same slice, and since there are fewer active users in a slice in this scenario, the multi-user diversity gain reduces, which reduces the throughput achieved with the PF scheduler. RadioSaber achieves 37.3%-50% higher throughput than NVS in this context for 10-20 slices. Note that when there are 5 slices, the throughput is limited by the total incoming network traffic.

Varying number of UEs. In the second experiment, we fix the number of slices to 20, and increase the number of UEs per slice. In Fig. 10(a), the aggregate throughput of NVS increases a little when the number of users per slice increases. This is reasonable since more users bring more diverse subband channel quality distributions for the PF enterprise scheduler in NVS to allocate RBs smartly. However, RadioSaber still outperforms NVS by 35.2%-56.8%. From Fig. 10(b), RadioSaber outperforms NVS by 49.7%-72.2% when the flows are heavy-tail distributed.

6.5 Non-greedy Enterprise Schedulers

Our evaluation so far only considered greedy enterprise schedulers. Indeed, most proposed (and potentially deployed) radio resource schedulers employ a greedy heuristic to achieve low runtime overheads [4]. We now compare how RadioSaber (using a greedy enterprise scheduler for PF) compares against NVS using a non-greedy heuristic for PF [17]. We briefly explain the non-greedy heuristic before presenting our results.

A UE can only use a single MCS(Modulation and Coding Scheme) across all RBs that are allocated to it. The typical practice is to first assign RBs to users, and then compute the MCS based on effective SNR across these RBs. The final data-rate so achieved is less than the sum of the data-rates achieved if one could use the optimal MCS value for each RB (our experimental results take this effect into account). Given this effect, GPF [17] uses a non-greedy heuristic to co-optimize the MCS assignment and the RB allocation – it samples 300 plausible mappings between each user and a corresponding MCS value (assuming that all RBs are available for a given user), and computes the best RB allocation for each of these MCS mappings. It then selects the MCS mapping and RB allocation that achieves the highest PF metric.

We evaluate how NVS using the above non-greedy PF scheduler for each slice compares against NVS and RadioSaber using greedy PF enterprise schedulers. We fix the number of slices to 20 and vary the number of UEs in each slice. Fig. 11 reports the overall throughput across the three schemes. We find that NVS with non-greedy PF achieves 13%-19% higher aggregate throughput than NVS with greedy PF. This better spectrum efficiency results from the tactical assignment of RBGs and MCS to users. However, the aggregate throughput of NVS (non-greedy PF) is still 14%-18% lower than RadioSaber since NVS is channel-unaware.

6.6 Is There a Better Inter-Slice Scheduler?

We now evaluate two questions:

(i) What is the impact of assigning RBGs in the order of decreasing channel quality? For this, we modify RadioSaber's inter-slice scheduler to greedily assign RBGs to the slices with maximum channel quality in a sequential order (from top to bottom). We term this strategy as "Sequential". With this approach, the inter-slice scheduler can query enterprise schedulers to determine the channel quality when it

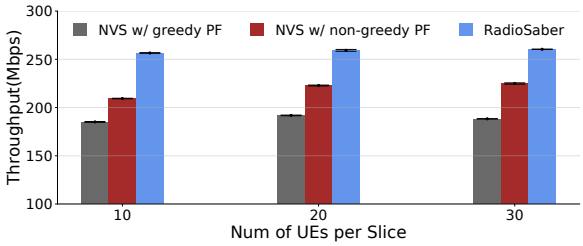


Figure 11: Comparing RadioSaber (using a greedy PF scheduler) with NVS (using a non-greedy PF scheduler). We fix number of slices to 20, and vary the number of UEs per slice.

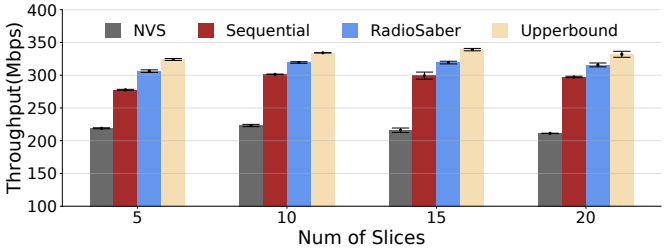


Figure 12: Comparing RadioSaber with a simple greedy inter-slice scheduler, and with a contrived upperbound. We vary the number of slices, with 5-15 random users in each slice, and use MT scheduling policy in each slice.

assigns RBGs, which avoids the need for recomputing channel quality after each allocation (as in the original inter-slice scheduling algorithm), resulting in a lower time complexity of $O(|\mathcal{RBG}||S|T)$.

(ii) Could we have achieved a better allocation? For this we compare RadioSaber with an impractical scheme that gives us an upper-bound on the spectrum efficiency that any inter-slice scheduler can achieve. In this scheme, we greedily allocate the RBG with the best channel quality to each slice until the slice's quota is exhausted. However, in doing so we allow a given RBG to be repeatedly allocated to multiple slices. As a result, each slice gets its best set of RBGs, independent of the allocation for other slices. Note that this upperbound still enforces fairness between slices, and allows customizable enterprise scheduling within each slice.

To make it easier to analyze the inter-slice schedulers we configure the enterprise scheduler in each slice to use the MT policy. We evaluate the overall throughput as we vary the number of slices, with 5-15 users in each slice. Fig. 12 shows that RadioSaber performs only 4%-6% worse than Upperbound and 6%-10% better than Sequential. We see similar trends across other settings (e.g. using PF policy in each slice, varying the number of users, etc).

The key takeaways are: (a) RadioSaber's inter-slice scheduling policy is close to optimal, and (b) Even the simple channel-aware greedy inter-slice scheduler achieves 25%-36% better throughput than channel-agnostic NVS, and is a good option if lower time complexity is required, at a cost of some throughput penalty compared to our current algorithm.

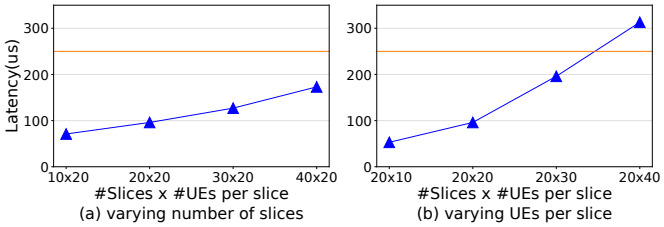


Figure 13: RadioSaber's scheduling latency

6.7 Scheduling Latency and Overhead

To evaluate RadioSaber's scheduling latency and runtime overhead, we implement its MAC scheduling logic on a system using a single Intel Xeon core. For simplicity of analysis, we configure each slice to apply the PF scheduling policy with backlogged users. Fig. 13(a) shows how the scheduling latency increases linearly with the number of slices when the number of UEs per slice is fixed to 20. Similarly, Fig. 13(b) shows how the scheduling latency increases linearly with the number of UEs per slice when the number of slices is fixed to 20. In both cases, the scheduling system can support as many as 600 users and make the scheduling decision within the stringent TTI constraint (250us).

7 Limitations and Future Work

• So far, we have only considered scheduling for downlink radio resources and not for uplink. The uplink channel applies SC-FDMA [5], which is similar to OFDMA and allows radio resources allocation in both time and frequency domain. This indicates that RadioSaber can be extended to uplink scheduling. However, it requires more complicated control information exchange between UEs and the base station. We leave a detailed exploration of this to future work.

• We only consider scheduling radio resources in a channel-aware manner for a single gNB. An interesting future direction is to extend our work in the context of multiple gNBs with small cells. The problem expands to a 3D allocation of frequency, time, and space resources in a channel aware manner.

Acknowledgement

We would like to thank our shepherd, Ganesh Ananthanarayanan, and the anonymous NSDI reviewers for their insightful comments and feedback. We're grateful to Ammar Tahir, Emerson Sie for their feedback in the camera-ready version. We would also like to thank Yaxiong Xie for sharing the LTE SNR traces with us. This work was supported by Intel, Facebook, AG NIFA under grant 2021-67021-34418, and UIUC's Smart Transport Infrastructure Initiative.

References

[1] Open5gs is a c-language open source implementation for 5g core and epc. https://github.com/open5gs, 2021.

[2] M. Andrews, K. Kumaran, K. Ramanan, A. Stolyar, P. Whiting, and R. Vijayakumar. Providing quality of service over a shared wireless link. *IEEE Communications Magazine*, 39(2):150–154, 2001.

[3] G. S. Association. 5g network slicing self-management white paper. https://www-file.huawei.com/-/media/corporate/pdf/news/5g-network-slicing-self-management-white-paper.pdf?la=en-us, 2020.

[4] F. Capozzi, G. Piro, L. Grieco, G. Boggia, and P. Camarda. Downlink packet scheduling in lte cellular networks: Key design issues and a survey. *IEEE Communications Surveys Tutorials*, 15(2):678–700, 2013.

[5] ETSI. 5g; 5g system; network slice selection services;. https://www.etsi.org/deliver/etsi_ts/129500_129599/129531/15.01.00_60/ts_129531v150100p.pdf, 2018.

[6] ETSI. 5g; nr; base station (bs) radio transmission and reception. https://www.etsi.org/deliver/etsi_ts/138100_138199/138104/15.02.00_60/ts_138104v150200p.pdf, 2018.

[7] ETSI. 5g nr: Physical channels and modulation(3gpp ts 38.211 version 16.2.0 release 16). https://www.etsi.org/deliver/etsi_ts/138200_138299/138211/16.02.00_60/ts_138211v160200p.pdf, 2020.

[8] ETSI. Evolved universal terrestrial radio access (e-utra); physical layer procedures. https://www.etsi.org/deliver/etsi_ts/136200_136299/136213/15.10.00_60/ts_136213v151000p.pdf, 2020.

[9] X. Foukas, M. K. Marina, and K. Kontovasilis. Orion: Ran slicing for a flexible and cost-effective multi-service mobile network architecture. In *Proceedings of the 23rd Annual International Conference on Mobile Computing and Networking*, MobiCom '17, page 127–140, New York, NY, USA, 2017. Association for Computing Machinery.

[10] X. Foukas, N. Nikaein, M. M. Kassem, M. K. Marina, and K. Kontovasilis. Flexran: A flexible and programmable platform for software-defined radio access networks. In *Proceedings of the 12th International on Conference on Emerging Networking EXperiments and Technologies*, CoNEXT '16, page 427–441, New York, NY, USA, 2016. Association for Computing Machinery.

[11] X. Foukas, G. Patounas, A. Elmokashfi, and M. K. Marina. Network slicing in 5g: Survey and challenges. *IEEE Communications Magazine*, 55(5):94–100, 2017.

[12] J. García-Morales, M. C. Lucas-Estañ, and J. Gozalvez. Latency-sensitive 5g ran slicing for industry 4.0. *IEEE Access*, 7:143139–143159, 2019.

[13] M. Gidlund and J.-C. Laneri. Scheduling algorithms for 3gpp long-term evolution systems: From a quality of service perspective. In *2008 IEEE 10th International Symposium on Spread Spectrum Techniques and Applications*, pages 118–123, 2008.

[14] M. K. Giluka, N. Rajoria, A. C. Kulkarni, V. Sathya, and B. R. Tamma. Class based dynamic priority scheduling for uplink to support m2m communications in lte. In *2014 IEEE World Forum on Internet of Things (WF-IoT)*, pages 313–317, 2014.

[15] T. Guo and A. Suárez. Enabling 5g ran slicing with edf slice scheduling. *IEEE Transactions on Vehicular Technology*, 68(3):2865–2877, 2019.

[16] M. B. Hcine and R. Bouallegue. Analytical downlink effective sinr evaluation in lte networks. In *2015 IEEE 29th International Conference on Advanced Information Networking and Applications Workshops*, pages 376–381, 2015.

[17] Y. Huang, S. Li, Y. T. Hou, and W. Lou. Gpf: A gpu-based design to achieve 100 us scheduling for 5g nr. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, MobiCom '18, page 207–222, New York, NY, USA, 2018. Association for Computing Machinery.

[18] H. Hui, Y. Ding, Q. Shi, F. Li, Y. Song, and J. Yan. 5g network-based internet of things for demand response in smart grid: A survey on application potential. *Applied Energy*, 257:113972, 2020.

[19] W. C. Jakes and D. C. Cox. *Microwave Mobile Communications*. Wiley-IEEE Press, 1994.

[20] E. Kahuha. 5 real life use cases of 5g ultra-reliable low-latency communication (urllc). https://www.section.io/engineering-education/five-real-life-use-cases-of-5g-ultra-reliable-low-latency-communication-urllc/, 2021.

[21] R. Kokku, R. Mahindra, H. Zhang, and S. Rangarajan. Nvs: A virtualization substrate for wimax networks. In *Proceedings of the Sixteenth Annual International Conference on Mobile Computing and Networking*, MobiCom '10, page 233–244, New York, NY, USA, 2010. Association for Computing Machinery.

[22] R. Kokku, R. Mahindra, H. Zhang, and S. Rangarajan. Nvs: A substrate for virtualizing wireless resources in cellular networks. *IEEE/ACM Trans. Network.*, 20(5), oct 2012.

[23] R. Kokku, R. Mahindra, H. Zhang, and S. Rangarajan. Cellslice: Cellular wireless resource slicing for active ran sharing. In *2013 Fifth International Conference on Communication Systems and Networks (COMSNETS)*, pages 1–10, 2013.

[24] R. Kwan, C. Leung, and J. Zhang. Proportional fair multiuser scheduling in lte. *IEEE Signal Processing Letters*, 16(6):461–464, 2009.

[25] H. Liu, L. Cai, H. Yang, and D. Li. Eesm based link error prediction for adaptive mimo-ofdm system. In *2007 IEEE 65th Vehicular Technology Conference - VTC2007-Spring*, pages 559–563, 2007.

[26] R. Mahindra, M. A. Khojastepour, H. Zhang, and S. Rangarajan. Radio access network sharing in cellular networks. In *2013 21st IEEE International Conference on Network Protocols (ICNP)*, pages 1–10, 2013.

[27] C. Marquez, M. Gramaglia, M. Fiore, A. Banchs, and X. Costa-Perez. How should i slice my network? a multi-service empirical evaluation of resource sharing efficiency. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, MobiCom '18, page 191–206, New York, NY, USA, 2018. Association for Computing Machinery.

[28] R. Mittal, J. Sherry, S. Ratnasamy, and S. Shenker. Recursively cautious congestion control. In *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*, NSDI'14, page 373–385, USA, 2014. USENIX Association.

[29] A. Nakao, P. Du, Y. Kiriha, F. Granelli, A. A. Gebremariam, T. Taleb, and M. Bagaa. End-to-end network slicing for 5g mobile networks. *Journal of Information Processing*, 25:153–163, 2017.

[30] N. Nikaein, E. Schiller, R. Favraud, K. Katsalis, D. Stavropoulos, I. Alyafawi, Z. Zhao, T. Braun, and T. Korakis. Network store: Exploring slicing in future 5g networks. MobiArch '15, page 8–13, New York, NY, USA, 2015. Association for Computing Machinery.

[31] A. Oborina, T. Henttonen, V. Koivunen, and M. Moisio. Efficient computation of effective sinr. In *2012 46th Annual Conference on Information Sciences and Systems (CISS)*, pages 1–6, 2012.

[32] M. Olsson, S. Sultana, S. Rommer, L. Frid, and C. Mulligan. Chapter 6 - session management and mobility. In M. Olsson, S. Sultana, S. Rommer, L. Frid, and C. Mulligan, editors, *SAE and the Evolved Packet Core*, pages 97–140. Academic Press, Oxford, 2010.

[33] L. Peterson and O. Sunay. *5G Mobile Networks: A Systems Approach*. USA, 2020.

[34] G. Piro, L. A. Grieco, G. Boggia, F. Capozzi, and P. Camarda. Simulating lte cellular systems: An open-source framework. *IEEE Transactions on Vehicular Technology*, 60(2):498–513, 2011.

[35] Qualcomm. Future of 5g: Building a unified, more capable 5g air interface for the next decade and beyond. https://www.qualcomm.com/media/documents/files/making-5g-nr-a-commercial-reality.pdf, 2020.

[36] I. Qualcomm Technologies. Vr and ar pushing connectivity limits. https://www.qualcomm.com/media/documents/files/vr-and-ar-pushing-connectivity-limits.pdf, 2018.

[37] A. Rao. 5g network slicing: crossdomain orchestration and management will drive commercialization. https://www.cisco.com/c/dam/en/us/products/collateral/cloud-systems-management/network-services-orchestrator/white-paper-sp-5g-network-slicing.pdf, 2020.

[38] J.-H. Rhee, J. Holtzman, and D.-K. Kim. Scheduling of real/non-real time services: adaptive exp/pf algorithm. In *The 57th IEEE Semiannual Vehicular Technology Conference, 2003. VTC 2003-Spring.*, volume 1, pages 462–466 vol.1, 2003.

[39] B. Sadiq, R. Madan, and A. Sampath. Downlink scheduling for multiclass traffic in lte. *EURASIP J. Wirel. Commun. Netw.*, 2009, mar 2009.

[40] J. X. Salvat, L. Zanzi, A. Garcia-Saavedra, V. Sciancalepore, and X. Costa-Perez. Overbooking network slices through yield-driven end-to-end orchestration. In *Proceedings of the 14th International Conference on Emerging Networking EXperiments and Technologies*, CoNEXT '18, page 353–365, New York, NY, USA, 2018. Association for Computing Machinery.

[41] K. Samdanis, X. Costa-Perez, and V. Sciancalepore. From network sharing to multi-tenancy: The 5g network slice broker. *IEEE Communications Magazine*, 54(7):32–39, 2016.

[42] sharetech. Resource allocation type. https://www.sharetechnote.com/html/Handbook_LTE_RAType.html, 2020.

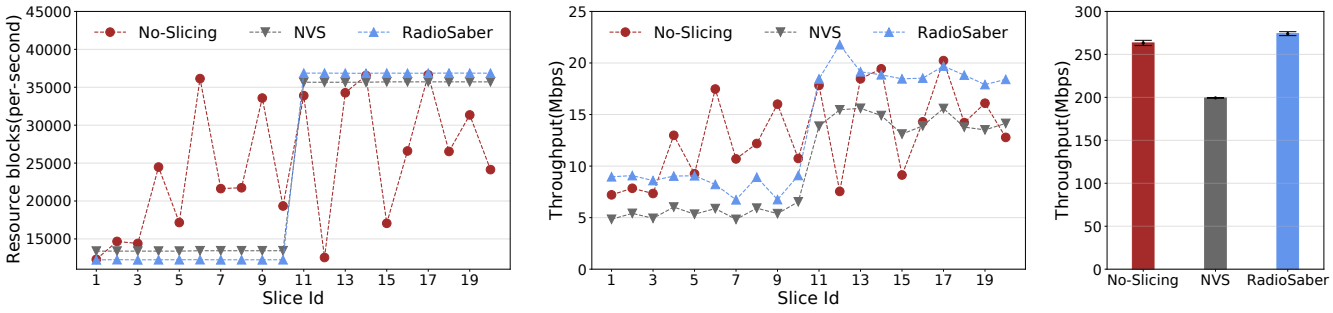
[43] Techplayon. 5g nr resource block definition and rbs calculation. https://www.techplayon.com/nr-resource-block-definition-and-rbs-calculation/, 2019.

[44] N. Van Giang and Y. H. Kim. Slicing the next mobile packet core network. In *2014 11th International Symposium on Wireless Communications Systems (ISWCS)*, pages 901–904, 2014.

[45] C. Wengerter, J. Ohlhorst, and A. von Elbwart. Fairness and throughput analysis for generalized proportional fair frequency scheduling in ofdma. In *2005 IEEE 61st Vehicular Technology Conference*, volume 3, pages 1903–1907 Vol. 3, 2005.

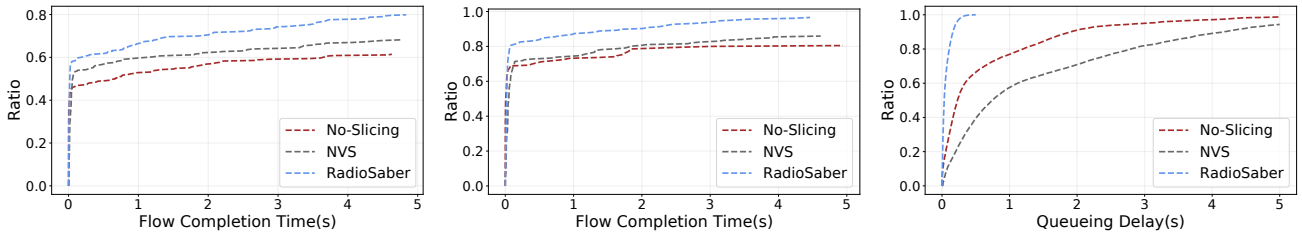
[46] Y. Xie, F. Yi, and K. Jamieson. Pbe-cc: Congestion control via endpoint-centric, physical-layer bandwidth measurements. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '20, page 451–464, New York, NY, USA, 2020. Association for Computing Machinery.

[47] M. Yan, G. Feng, J. Zhou, Y. Sun, and Y.-C. Liang. Intelligent resource scheduling for 5g radio access network slicing. *IEEE Transactions on Vehicular Technology*, 68(8):7691–7703, 2019.



(a) per-slice RBs per second (b) per-slice throughput (c) sum total throughput

Figure 14: (a) the average allocated RBs per second to slices; (b) the average aggregate throughput of slices; (c) the sum total throughput of three systems



(a) All flows in slice 6-10 (b) Prioritized flows in slice 11-15 (c) All real-time video flows

Figure 15: CDFs of (a) completion time of flows in slices 6-10, (b) completion time for the higher prioritized flows in slices 11-15, and (c) queuing delay of packets in slices 16-20. We cut the graphs at 5s to better highlight the trends.

A Supplemental Evaluation

A.1 Slices with Different Weights

§6.1 evaluated the scenario where every slice has the same weight. Here we do the same experiment but slightly modify SLAs of slices: slices with PF schedulers have 3X higher weights than slices with MT schedulers. Fig. 14 shows the experiment results. From Fig. 14(a) and Fig. 14(b), it's obvious that the last 10 slices get 3X more RBs than the first 10 slices, and approximately 3X higher throughput. Meanwhile, it's impossible for No-Slicing to meet the SLAs of slices since it doesn't support network slicing between groups of users at all.

A.2 CDF Graphs of FCT and Queueing Delay

We provide more evaluation results in §6.2. Fig. 15(a) shows the CDF of flow completion time in the slice 6-10, and Fig. 15(b) shows the CDF of flow completion time for the higher priority flows in slices 11-15, and Fig. 15(c) shows the CDF of queuing delay for slices 16-20. We cut the graphs at 5s to better highlight the trends. For flows in slices 6-15, the FCT in No-Slicing is the longest since No-Slicing cannot enforce fairness for slices in which flows arrive intermittently. NVS suffers from the longest queuing delay in slices 16-20 due to low spectrum efficiency.