A practical approach to slicing Wi-Fi RANs in future 5G networks

Joan Josep Aleixendri*, August Betzler*, and Daniel Camps-Mur*
*i2CAT Foundation
joan.aleixendri@i2cat.net, august.betzler@i2cat.net, daniel.camps@i2cat.net

***Abstract*—To deliver on the vision of network slicing, future 5G networks will have to provide virtualization and isolation for a variety of Radio Access Network (RAN) technologies. In this paper we study how to implement RAN slicing in Wi-Fi networks. In particular, we present a scheduling algorithm that allocates airtime through a set of virtual Wi-Fi interfaces executing on the same or different physical radios, in order to fulfill each tenant's Service Level Agreement. We demonstrate a practical implementation of our scheduler on Linux systems based on the *mac80211* driver that contrary to other existing solutions is independent from underlying hardware drivers. Finally, we experimentally evaluate the performance of our scheme through a set of realistic experiments.**

***Index Terms*—Slicing, Wi-Fi, Airtime, SDN**

I. Introduction

Network slicing, introduced by the NGMN in [1], is one of the main principles guiding the design of 5G networks. Network slicing requires the instantiation of multiple virtual networks over a single shared physical infrastructure, which can then be controlled by different *tenants*. An important aspect of network slicing is its end to end nature, whereby a slice spans from the Radio Access Network (RAN) to the core network. Slicing the core network is addressed through the decomposition of core network functions and the execution of these functions inside virtual machines, and can leverage all the isolation mechanisms developed for IT virtualization. However, RAN slicing is in an earlier stage, and it is for example not yet clear how to isolate RAN virtual functions [2], [3]. In addition, 5G will integrate different types of radio technologies such as evolution of LTE, the 5G New Radio (NR) and Wi-Fi based technologies. Hence, RAN slicing needs to be supported for each technology, and in this context this paper focuses on implementing Wi-Fi RAN slicing.

The problem of virtualizing Wi-Fi networks has been extensively studied in the literature [4]. The main idea is to execute upper Media Access Layer (MAC) functions in software, running on the platform's CPU, while time sensitive operations are offloaded to the Wi-Fi modem. Upper MAC functions include, among others, the generation of Beacon frames, which convey the identity of the Wi-Fi network, or per-client state related to security operations. This approach is already supported in mainstream operative systems such as Linux, which allow to instantiate multiple virtual interfaces (*vifs*) over a single physical Wi-Fi radio [5]. Each *vif* is a separate process representing an upper MAC, which can be configured to operate in different modes, e.g. access point (AP) mode, or mesh point (MP) mode. Virtual interfaces have been leveraged in [6] to introduce the abstraction of per-client lightweight APs, which can be associated to a per-client slice and easily transferred between physical APs to aid mobility. However, neither Linux nor [6] provide any means to isolate the wireless resources consumed by each *vif*. This is the main problem addressed in this paper.

The problem of isolating *vifs* is related to the problem of resource allocation in Wi-Fi. An approach that has been extensively studied in the past is tuning the contention parameters of different traffic classes in order to provide prioritized QoS [7]. This approach however, has only been addressed for small number of traffic classes (IEEE 802.11e supports four traffic classes) and for a single AP, and fails to guarantee that a tenant can, regardless of the network conditions, hold a given percentage of the available resources. On the other hand, Wi-Fi networks are known to suffer from the rate anomaly problem [8], whereby slow clients can capture an excessive amount of network resources, i.e. *airtime*. Therefore, solutions are required that can properly allocate the airtime resource among different *vifs* in order to provide hard guarantees to a given slice. In [9] a variant of Deficit Round Robin (DRR) is proposed that can allocate airtime to different classes in Wi-Fi. However, [9] does not discuss practical implementation aspects, and the proposed solution fails to orchestrate resources among *vifs* that operate on specific physical interfaces. The implementation in [8] proposes a solution supporting multiple *vifs* over a single AP, but it only works for a specific driver.

The main contribution of this paper is an architecture and practical, hardware-independent implementation, of an airtime based scheduler implementing isolation between *vifs* belonging to different network slices, and contemplating the case of having *vifs* running over different physical interfaces. To the best of our knowledge, ours is the first contribution where Wi-Fi RAN slicing over a set of APs is experimentally demonstrated.

This paper is organized as follows. Section II describes our Wi-Fi RAN slicing solution. Section III contains an experimental evaluation demonstrating the performance of our solution. Finally, Section IV summarizes and concludes the paper.

II. System Design

In this section we present our Software-Defined Networking (SDN)-based Wi-Fi RAN slicing solution that combines dis-

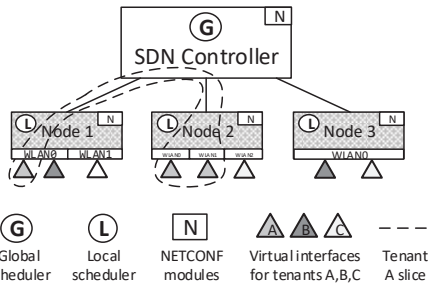

Fig. 1: Our network architecture is composed of APs with physical interfaces, on top of which virtual interfaces (*vifs*) are instantiated for different tenants. The SDN controller manages the *vifs* and configures the APs to apply the Wi-Fi slicing.

tributed and centralized mechanisms to control slicing over a typical dense Wi-Fi deployment. Our network architecture is composed of network nodes equipped with wireless transceivers to provide radio access, on top of which *vifs* for different tenants can be instantiated. The core element of our slicing solution is what we refer to as a *local scheduler*, built on top of the IEEE 802.11 Media Access Layer (MAC) of the wireless transceivers and responsible to allocate airtime between the active *vifs* running on the same physical radio. The instantiation and management of *vifs* on the nodes and each node's configuration of the local scheduler is performed by an SDN controller that is connected to the APs via a backhaul infrastructure. The SDN controller takes over the task of the *global scheduler* that also is responsible for monitoring the status of the RAN and assuring that the slicing is performed correctly. A slice is defined by a tenant selecting the physical APs where it wants to provide connectivity via a *vif* implementing a virtual AP (*vap*), assigning SSIDs and security and associating an SLA to the slice, consisting of the resources ratio assigned to this tenant (defined later in detail). The SDN controller uses NETCONF to instantiate *vifs* and to configure local schedulers according to the slice. Fig. 1 shows the basic elements of our architecture and their relations.

A. Local Scheduler Component

The local scheduler component needs to fulfill the following two basic requirements: First, it needs to be work conserving, i.e., the amount of unused airtime shall be minimized by reassigning unused resources to *vifs* in need of airtime for packet transmissions. Second, it needs to be independent from any underlying Wi-Fi drivers and hardware in order to achieve a high degree of interoperability and portability.

Slicing is performed in the time domain: each user, e.g. a tenant or wireless operator represented as a *vif* on top of a physical radio, is assigned a specific ratio of the available airtime. This share represents a guaranteed minimum of the available airtime during a certain time interval, but not a maximum. In our slicing solution the real share of the airtime assigned to each user is variable during operation and it can be adapted in order to fully use the available airtime. We define A to be the set of available APs, where $a \in A$ is a physical AP, the set of network slices S, where $s \in S$ is a tenant's network slice. Further, we define μ to indicate the utilization of a radio channel in % and ρ to indicate the airtime.

Our scheduler at the core operates on a traditional credit-based system: We define a periodically repeating, fixed time interval T, during which a specific airtime $\rho_{a,s}^{max}$ is calculated for *vif* s in AP a. $\rho_{a,s}^{max}$ is initialized as $T_{a,s}$, which is calculated from the desired share according to the SLA, $\mu_{a,s}^{\text{SLA}}$ assigned a priori to slice s, as $T_{a,s} = T \times \mu_{a,s}^{\text{SLA}}$, and where $\sum_{s=1}^{S} \mu_{a,s} \leq 1$ applies for all APs. A credit represents a unit of time within the interval T and it can adopt any granularity supported by the operating system, e.g., microseconds. Every *vif* is therefore assigned a specific number of credits $\rho_{a,s}^{max}(t)$ it can spend during T. The operation of the scheduler can be divided into three phases: when a packet is handed over to our module to be transmitted, after the packet transmission and at the end of an interval T. In the following, the scheduler actions during each of these phases is explained.

Credit Consumption: Whenever a packet $P_{i,s}$ is being transmitted, it traverses our scheduler, before being handed over to the wireless driver. To determine whether the packet can be transmitted, its expected airtime $\rho_{P_{i,s}}^{exp}$ is calculated

$$\rho_{P_{i,s}}^{exp} = \rho_{P_{i,s}}^{ideal} \times F_c(t), \tag{1}$$

where $\rho_{P_{i,s}}^{ideal}$ is the airtime of the packet in a non-congested channel and $F_c(t)$ is a congestion factor that indicates how busy the medium is. Notice that accounting for congestion allows to distribute the available bandwidth between local *vifs*, in case other transmitters are contending for the channel. For the calculation of $\rho_{P_{i,s}}^{ideal}$, the packet length, the current Modulation Coding Scheme (MCS), and additional timings are taken into account when calculating the duration of a packet transmission. However, the real duration of a transmission may vary because of backoffs, collisions and retransmissions. In our scheduler, we keep track of the congestion by estimating the available bandwidth and adjusting $F_c(t)$ (which is updated after the packet transmission, see below). The expected airtime $\rho_{P_{i,s}}^{exp}$ is checked against the credits available for *vif* s during the current time interval T. If enough credits are available for the transmission, the packet is handed over to the MAC layer to be transmitted and the available credits are reduced by $\rho_{P_{i,s}}^{exp}$. Otherwise, the packet is enqueued again.

Tracking Medium Congestion: After a transmission, the MAC layer generates a transmission report with the packet's MCS, which will be used for subsequent airtime estimations. The transmission report also is used to calculate the actual transmission time ($\rho_{P_{i,s}}^{measured}$) by comparing timestamps when handing over a packet to the MAC (t_{sent}) and receiving the correspondent report ($t_{received}$). This allows us to calculate $F_c(t)$ (with $F_c(0) = 1$) using a filter based on weighted moving averages that allows us to quickly follow channel congestion. First, $\rho_{P_{i,s}}^{measured}$ is computed as

$$\rho_{P_{i,s}}^{measured} = (t_{received} - t_{sent}) * F_{process} \tag{2}$$

$F_{process}$ is a constant < 1 used to compensate for the duration of internal packet processing in the lower layers of

the stack which are not part of the actual packet transmission. For our configuration, we determine $F_{process}$ to be 0.8.

Once $\rho_{P_{i,s}}^{measured}$ is calculated, the ratio $R_{P_{i,s}}$ between the real and the ideal airtime is derived $R_{P_{i,s}} = \frac{\rho_{P_{i,s}}^{measured}}{\rho_{P_{i,s}}^{ideal}}$. Depending on $R_{P_{i,s}}$, there are two ways to update F_c:

$$F_c(t+1) = \begin{cases} W_{inc}(F_c(t)), & \text{if } R_{P_{i,s}} \geq (1 + \frac{\theta}{100}) \\ max(W_{dec}(F_c(t)), 1), & \text{otherwise} \end{cases}$$

The parameter θ, is a configurable threshold in percent, above which the wireless medium is assumed to be congested. In the first case of the equation, if $R_{P_{i,s}}$ is exceeded by at least θ, F_c is updated with a moving average

$$W_{inc}(F_c(t)) = F_c(t) \times \alpha + R_{P_{i,s}} \times (1 - \alpha) \tag{3}$$

This represents the case where the channel is found to be congested, i.e., the transmission of packets takes considerably longer than the ideal time. Otherwise when the differences between real and ideal airtime are below the threshold, the degree of congestion is considered to be small and F_c is decremented moving it towards a minimum of 1 with

$$W_{dec}(F_c(t)) = F_c(t) \times (1 - \alpha) + R_{P_{i,s}} \times \alpha \tag{4}$$

A complementary weight (α) is used in the increment and decrement filters, to allow for different speeds in estimating increase and decrease of channel congestion. The impact of α is evaluated in subsection II-E

Redistributing Credits: At the end of an interval T the local scheduler in each AP calculates the usage of each *vif* s

$$\mu_{a,s}^{used}(t) = \frac{\sum_{P_{i,s}=1} \rho_{P_{i,s}}^{exp}}{\rho_{a,s}^{max}(t)} \tag{5}$$

and determines whether the assigned limit needs to be modified. If $\mu_{a,s}^{used}$ of the least active *vif* is less than 10%, we consider the *vif* to be idle and $\rho_{a,s}^{max}(t)$ for this *vif* is reduced:

$$\rho_{a,s}^{max}(t) = max(\rho_{a,s}^{max}(t) \times \mu_{a,s}^{used}(t), \rho_{a,s}^{min}) \tag{6}$$

The number of credits that are subtracted in this way are split among the remaining *vifs* according to their shares and added to their $\rho_{a,s}^{max}(t)$ for the next period. $\rho_{a,s}^{min}(t)$ assures that a *vif* is allowed to transmit a minimum of packets during an interval. This local optimization redistributes unused airtime among active *vifs* in a work conserving manner, as long as there are inactive *vifs* not requiring any airtime. If $\mu_{a,s}^{used}$ ever exceeds 30% for a *vif* with a reduced $\rho_{a,s}^{max}(t)$, the *vif* is considered to be active. In this case, to assure that an active *vif* can quickly obtain its full share of the network slice, a reset is performed and for all *vifs* their original values of $\rho_{a,s}^{max}(t) = T_{a,s}$ are restored.

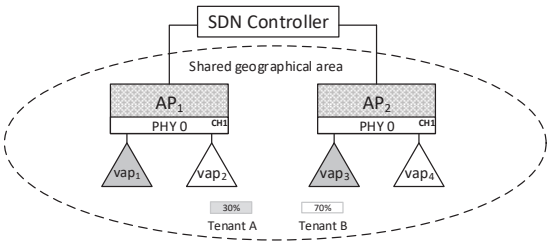

Fig. 2: Two APs sharing the wireless medium due to close vicinity. The APs provide *vaps* for two tenants (A, B) that have been assigned shares of 30% and 70%, respectively.

B. Global Scheduler Component

The local schedulers guarantee that $\mu_{a,s}^{\text{SLA}}$ is maintained for each *vif* running on the same physical radio. However, this condition is not sustainable if multiple devices with independently operating local schedulers are competing for the wireless medium, which is typical for high density deployments. In the example shown in Fig. 2, two APs with virtual APs (*vaps*) for two tenants A and B are configured to have a share of 30% and 70%, according to their respective SLAs. Both APs operate on the same channel. Locally, on each of the devices, the local scheduler maintains the ratios for each tenant when operating alone. However, if both devices are operating at the same time, the local share optimization can lead to violations of the globally assigned shares. In this example, if *vap*$_1$ corresponding to tenant A on AP_1 is transmitting alone, the scheduler will reassign the unused airtime of vap_2 belonging to tenant B to vap_1. Accordingly, if *vap*$_4$ of tenant B on AP_2 is the only one transmitting, the unused airtime of *vap*$_3$ is assigned to *vap*$_4$. As a result, *vap*$_1$ and *vap*$_4$ are locally assigned 100% of the airtime, i.e., they are only limited by the capacity of the radio channel. Since *vap*$_1$ and *vap*$_4$ are on the same radio channel and within transmission range, this results in them sharing the medium equally at a ratio of 50%/50%, which violates the tenants' SLAs.

It is the task of the global scheduler running in the SDN controller to periodically monitor $\mu_{a,s}^{used}$ of each *vif* in each AP, and to adjust their local scheduler when SLA ratios are violated. In this paper we propose a simple heuristic in the global scheduler, which applies when all the APs can see each other, as in a dense wireless deployment. At the end of each period T, $\mu_{a,s}^{used}$ is collected from all APs and the global scheduler computes the corrected values as $\mu_{a,s}^{corrected} = \sum_{a \in A} \mu_{a,s}^{used}(t)$. In our implementation, the periodic checks are performed every second and the corrected values are updated before the local scheduler redistributes credits at the end of each period T. In Section III we analyze the performance of this heuristic.

C. Uplink Traffic

While downlink traffic can be fully controlled by the local scheduler component as described in Section II-A, uplink traffic generated by user equipment needs to be handled in a different way. First, any packets received by a *vap* are counted towards the usage $\mu_{a,s}^{used}(t)$ reducing the available

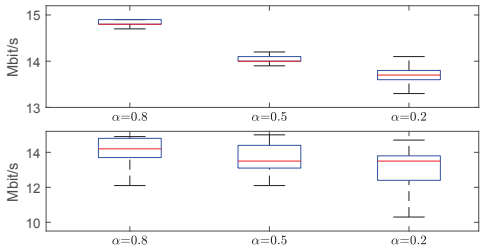

Fig. 3: Box plots showing the throughput, with the median, 1st and 3rd quartiles (lower/upper box limits), and min. and max. values (lower/upper whiskers) for the continuous case (top) and with interference (bottom).

airtime for this particular slice. Second, the uplink traffic is controlled indirectly, as the rate of downlink replies in any bidirectional traffic (e.g. TCP that requires acknowledgement or HTTP response packets) is controlled by the local scheduler.

D. Implementation of the local scheduler

The dynamically loadable Linux kernel module that implements our scheduling algorithm is written in C and it is placed in between the IP module and the *mac80211* module. Since the module is very lightweight, it can even run on constrained devices, e.g. a Raspberry Pi. The local scheduler module uses the *nftables* API [10] to intercept all the outgoing traffic and it can be loaded, unloaded and reconfigured dynamically, which allows reconfiguration by the SDN controller. Our local scheduler module is linked to the mac80211 module, and hence can work with any Wi-Fi chipset, including 11n or 11ac, supported in Linux. This in contrast to [11] that relies on low-level drivers (ath9k) to implement airtime based scheduling.

E. Evaluation of the α parameter

The parameter α determines how quickly the local scheduler adapts the congestion factor F_c when detecting an increase or decrease of congestion. We determine an optimal α (Eqs. 3 and 4) from a set of options (0.8, 0.5, or 0.2). The setting $\alpha = 0.8$ makes F_c slowly increase when detecting congestion but quickly decrease, as soon as the channel is detected to be free again. The opposite behavior is achieved by setting $\alpha = 0.2$, whereas the setting $\alpha = 0.5$ represents a balanced behavior that equally weights increases and decreases of congestion. We perform two experiments in an indoor environment and evaluate the parameter's impact on the performance.

In the first experiment, we launch a UDP iperf transmission with a data rate of 15 Mbit/s from an AP to a client device, while allowing the devices to consume 100% of the airtime. Apart from sporadic transmissions on the radio channel, it can be assumed to be free of interference. In the second experiment we set up the same transmission between AP and client, while interfering the connection with another, independent transmission on the same channel with a load of 3 Mbit/s that turns on and off every 2 s. This generates cycles of network congestion where the scheduler needs to adapt the airtime cost estimation. In both experiments we measure the throughput between AP and client every 0.5 s. The experiments are performed for each setting of α with a duration of 10 minutes, setting an empirically determined θ to 30%.

Figure 3 reveals that in both the continuous and interfered traffic cases the median throughput achieved with $\alpha = 0.8$ is the highest. The larger variations observed in the experiment with interference are the result of the transmissions adapting to low and high congestion phases. If α is set to a lower value, i.e., the system reacts more quickly to larger observed packet transmission durations, we observe a tendency to overestimate congestion which reduces the overall data throughput when in fact the channel would allow for more data to be transmitted. This reflects in the results obtained with $\alpha = 0.5$ and even at a higher degree with $\alpha = 0.2$. Based on these results, we choose $\alpha = 0.8$ for the evaluations of our slicing solution.

III. Performance Evaluation

In order to validate the scheduler and to measure its performance, we define and execute a total of 4 experiments in a physical setup in our office premises. The setup consists of a PC running a Linux environment with our local scheduler kernel module and it is equipped with two IEEE 802.11 a/b/g/n transceivers. The SDN controller runs on a separate machine. During the setup phase, the two Wi-Fi interfaces are set up as APs on the least interfered channel 4 in the 2.4 GHz Band. On top of each Wi-Fi physical interface, 2 *vaps* are instantiated for two tenants A and B. The *vaps* are configured according to their Service Level Agreement (SLA) with predefined slicing ratios of 30% (tenant A) and 70% (tenant B). Further, one client is attached to each of the *vaps*. The parameter T is set to 250 ms and θ to 30%. During each experiment, a UDP-based data stream generated with iperf is launched from each of the *vaps* towards its attached client device. This type of traffic represents a typical use case with a downstream connection where the client device requested a stream of media content or a large file. The throughput and airtime measured for each of the tenants are used as performance metrics for these evaluations. The experiments are performed during the late evening or night in order to minimize external interference.

In the first 3 experiments only one wireless transceiver is used, whereas the 4th experiment evolves around the example introduced in Section II-B (Fig. 2) with two competing Wi-Fi APs. In the following subsections, a description of each experiment is given, along with the obtained results.

Local Scheduler: Constant Load and Dynamic SLA

In this experiment it is shown how the scheduler applies network slicing for two tenants and how it can update an SLA (change slicing ratios) during runtime. At the beginning of the experiment, the scheduler maintains the predefined ratios of 30%/70% for two constant, unbounded data streams generated by vap_1 and vap_2, respectively. After 30 s, the ratio is inverted, i.e., tenant A is assigned 70% of the airtime, whereas tenant B is assigned the remaining 30%. The scheduler is notified about the change and it readjusts the rate for each *vap* in order to fulfill the new network slicing ratios. Figure 4 shows how the

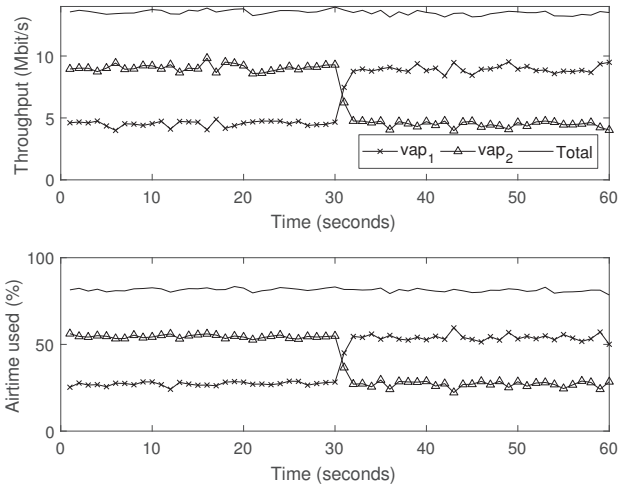

Fig. 4: Per-tenant throughput and airtime in the constant load and dynamic SLA experiment.

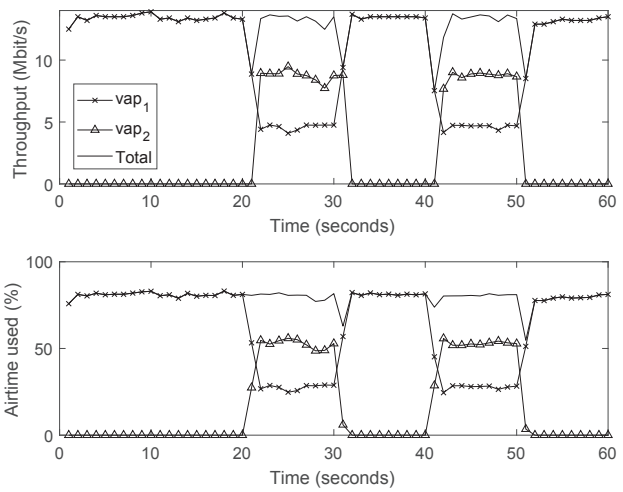

Fig. 5: Per-tenant throughput and airtime in the dynamic load experiment.

throughput and the (measured) busy airtime evolve for the two *vaps* during the experiment.

The MAC layer in average reports a MCS corresponding to 18 Mbit/s between the *vaps* and their clients. The real overall throughput of the *vaps* approximates 13.5 Mbit/s, split between tenants A and B in a ratio of 33% and 67%, respectively. The minor deviation of 3% from the ideal shares can be assigned to the non-ideal radio channel behavior. Small fluctuations in MCS or even minor external interference can lead to inaccuracies in the calculations performed by the local scheduler, resulting in non-ideal slicing. As soon as the ratios are inverted at second 30 of the experiment, the throughput inversion of the two *vaps* can be observed. There is a short transition phase during which the scheduler readjusts the shares of the two *vaps* that lasts approximately 1 to 2 s.

The overall busy airtime measured with the wireshark tool for both *vaps* is 80%. Packet drops, collisions, and inter frames spacing are not accounted for in the busy airtime. Taking the 80% busy time as a reference, tenant A is assigned 31% of the airtime in the first half of the experiment, whereas tenant B is assigned 69%. The shares are inverted in the second half of the experiment. These values are very close to the ideal values, showing that the scheduler works very accurately.

Local Scheduler: Dynamic Load

In this experiment we show how the local scheduler optimizes slicing by redistributing unused airtime. At the beginning of the experiment only vap_1 of tenant A is transmitting a continuous bulk of data as an unbounded UDP stream. After 20 s, vap_2 of tenant B transmits for a period of 10 s, after which it stops transmitting. This burst-type transmission is repeated from seconds 40 to 50.

In Fig. 5 it can be seen how vap_1, even though only assigned 30% of the airtime, uses the full capacity of the channel during the initial 20 s of the experiment. In this situation the scheduler behaves in a work conserving manner and redistributes the airtime from idle tenant B to tenant A. As soon as tenant B requires its share of 70% of the airtime (seconds 20 and 40), the scheduler detects the increment in the usage of vap_2 and resets the credits assigned to vap_1 to 30%. For the duration of the bursts, the assigned per-tenant ratios are maintained. The transition phases are clearly visible as a slight drop of the overall throughput before and after the burst. The busy airtime measured during the experiment shows how tenant A maintains a high usage, except for the durations of the bursts, where tenant B is assigned the greater part of the available airtime. The slicing ratios are applied correctly, except for the short transition phases, when the scheduler readjusts the slices.

Local Scheduler: Varying MCS

The last experiment with a single AP shows how the local scheduler is capable of maintaining airtime ratios in the case that the radio transmission rates vary during operation. This is a typical phenomenon observable in mobility use cases or in dynamic radio environments. For this experiment, we fix the MCS between each *vap* and their attached client to a transmission rate of 18 Mbit/s and generate unbounded data streams for tenants A and B at the same time. After 30 s, we fix the MCS of vap_2 to 12 Mbit/s, while maintaining the MCS with 18 Mbit/s for vap_1. The scheduler is notified by the driver that the MCS has changed and dynamically calculates different credit usage for the two *vaps*, while maintaining the assigned airtime slices for each tenant.

As can be seen in Fig. 6, the throughput reflects the correct application of the network slices for both tenants in the initial phase, as already observed in the previous experiments. At second 30, the MCS drop from vap_2 towards its attached client has a significant impact on its own and the overall throughput. The scheduler now adapts the credit estimation for vap_2 to these new circumstances. This results in vap_2 reducing its throughput, while vap_1 maintains it, proving the isolation properties of the scheduler. The busy airtime before and after the transmission rate change is constant for both *vaps*.

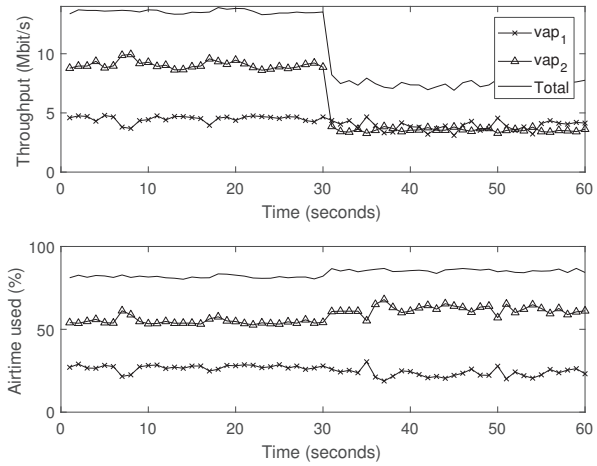

Fig. 6: Per-tenant throughput and airtime in the varying MCS experiment.

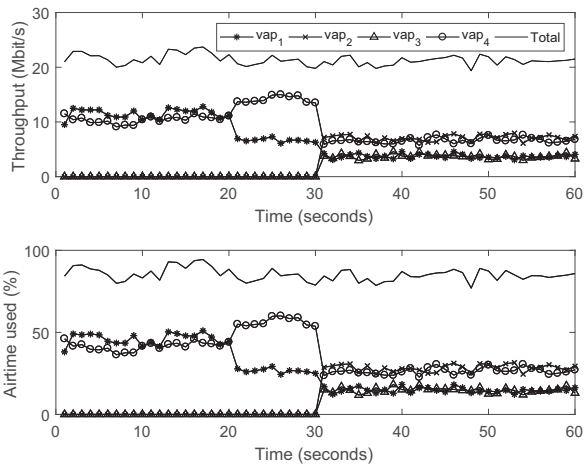

Fig. 7: Per-tenant throughput and airtime in the experiment showing interaction of the global scheduler.

Global Scheduler

To analyze the global scheduler we consider the case discussed in Section II-B and depicted in Fig. 2 with two APs providing *vaps* for tenants A and B operating on the same channel. This experiment shows how the SDN controller detects and addresses violations of slicing ratios in a geographical area. The experiment can be broken down into 3 phases. In the initial phase, vap_1 and vap_4 are transmitting an unbounded data stream, using a MCS with a transmission rate of 54 Mbit/s. Since the *vaps* are managed by different local schedulers, each local scheduler assigns the unused share of the inactive *vaps* (vap_2 on AP_1, vap_3 on AP_2) to the active ones. As a result, vap_1 and vap_4 are competing equally for the channel, splitting the available airtime equally among each other. This behavior can be seen in Fig. 7 from the start of the experiment to second 20, during which the throughput and airtime are the same for vap_1 and vap_4. This, however, violates the SLAs accorded by tenant A (30%) and tenant B (70%).

The global scheduler component in the SDN controller is activated at second 20 and it detects the violation of the assigned shares. The controller reacts by telling the scheduler running in AP_1 to locally reduce the allowed share for tenant A (vap_1). This initiates the second phase, during which the ratios are maintained for vap_1 and vap_4. In Fig. 7, this second phase corresponds to seconds 20 to 30. In the third and last phase of the experiment (second 30 to the end), vap_2 and vap_3 are activated, transmitting an unbounded stream of data. In this phase, the local schedulers of AP_1 and AP_2 manage the slicing locally and as a result, the correct shares are assigned to each tenant. This last phase shows that the shares defined for multiple tenants are applied correctly across several devices in a geographical area while all are transmitting at the same time. The throughput and the airtime ratios are maintained as defined in the SLAs, assigning the *vaps* of tenant A approximately 30% of airtime and the *vaps* of tenant B the remaining 70%.

IV. Conclusions

This paper presents a practical solution to slice Wi-Fi RANs in future 5G networks. This work provides in-depth details about its two main components, the centralized global and local scheduler mechanisms, and validates the correct operation of the slicing solution in a set of experiments. The designed solution fulfills 3 main requirements, namely being work conserving, having a high degree of portability, and enforcing the correct network slicing on a local level, but also in a geographical area with multiple Wi-Fi devices. As future work we consider porting our solution to IEEE 802.11ac compliant wireless radios, and to design global scheduler algorithms for situations where not all APs see each other.

V. Acknowledgements

The research leading to these results has received funding from the European Union under grant agreement 761508 (H2020 5GCITY), and from the Spanish Ministry of Economy and Competitiveness (MINECO), through project TEC2016-76795-C6-2-R and FEDER.

References

[1] NGMN Alliance, NGMN 5G White Paper, White paper, Feb. 2015.
[2] Marsch, P., et al. "5G Radio Access Network Architecture: Design Guidelines and Key Considerations." IEEE ComMag 54, no. 11 (2016).
[3] O. Sallent; J. Perez-Romero; R. Ferrus; R. Agusti, "On Radio Access Network Slicing from a Radio Resource Management Perspective" in IEEE Wireless Communications , vol.PP, no.99, (pp.2-10)
[4] Xia, L., et al. "Virtual WiFi: bring virtualization from wired to wireless." In ACM SIGPLAN Notices, vol. 46, no. 7, pp. 181-192. ACM, 2011.
[5] Vipin, M., and Srikanth, S. (2010, January). "Analysis of open source drivers for IEEE 802.11 WLANs." In ICWCSC, 2010. (pp. 1-5). IEEE.
[6] Schulz-Zander, J., et al. (2014, June). "Programmatic Orchestration of WiFi Networks." In USENIX Annual Technical Conference.
[7] Banchs, A., and Vollero, L. "Throughput analysis and optimal configuration of 802.11 e EDCA." Computer Networks 50, no. 11 (2006).
[8] Tan, G., and Guttag, J. V. (2004, June). "Time-based Fairness Improves Performance in Multi-Rate WLANs." In USENIX Annual Technical Conference, General Track (pp. 269-282).
[9] Riggio, R., Miorandi, D., and Chlamtac, I. (2008, September). "Airtime deficit round robin (ADRR) packet scheduling algorithm." MASS 2008. 5th IEEE International Conference on (pp. 647-652).
[10] Nftables project, https://netfilter.org/projects/nftables/
[11] Høiland-Jørgensen, T., et al. "Ending the Anomaly: Achieving Low Latency and Airtime Fairness in WiFi." USENIX 2017