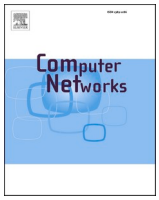
Computer Networks 284 (2026) 112289

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet



Practical challenges of implementing a hybrid ISS-QoSS RAN slicing in SDN WLAN networks

Ángela Hernández-Solana *, José Ruiz-Mas, María Canales, Julian Fernández-Navajas, José Ramón Gállego, Sara Ibáñez-Alloza

University of Zaragoza, Zaragoza, Spain

ARTICLE INFO

Keywords:
IEEE 802.11 networks
Wi-Fi Slicing
Airtime management
Adaptive slice management
SDN
User role-based slicing
QoS-driven slicing

ABSTRACT

Network slicing is a key feature in 4G and 5G mobile networks and, more recently, IEEE 802.11 networks. This paper proposes a hybrid Infrastructure Sharing Slicing (ISS) and Quality of Service Sharing Slicing (QoSS) solution, adapting the Airtime Deficit Weighted Round Robin (ADWRR) scheduling. The approach ensures isolation and resource segmentation among tenants or client categories, along with fine-grained traffic differentiation, prioritization or resource isolation for various service classes within slices. It includes local adaptations of allocated partitions/portions of airtime, a.k.a. quantum, at the two levels (ISS-QoSS) to improve specific QoS requirements by modifying the natural ADWRR airtime reallocation. The RAN slicing functions are implemented and tested on a Software Defined Networking (SDN) experimental prototype using an open-source framework combining centralized and distributed components. The technical focus is on practical challenges affecting local-level implementation efficiency and feasibility.

1. Introduction

Network slicing is one of the key enabling features for wireless networks, including 3rd Generation Partnership Project (3GPP) standardized wireless cellular technologies as 4G and 5G New Radio (5G NR) and more recently in Wi-Fi networks. Network slicing allows to create multiple virtual end-to-end networks on top of a shared physical infrastructure or to divide the networks into multiple logical pieces to deal with different applications or use cases.

The concept is especially important in 4G or 5G NR mobile networks. In this context, infrastructure providers can lend their network resources to other network players (tenants) as Virtual Mobile Network Operators (VMNOs) or Over-The-Top (OTT) applications, giving them control over leased resources. But, in any case, slicing enables the network operator to better manage multiple services with conflicting requirements, linked to specific use cases, such as Massive Machine Type Communications (mMTC), Ultra-Reliable and Low-Latency Communications (URLLC), or Enhanced Mobile Broadband (eMBB). However, despite the importance of slicing and the significant advancements in slice management and orchestration, proposals for slicing in the Radio Access Network (RAN) are less numerous compared to core and transport slicing, primarily due to the inherent variability of radio environments. Unlike the relatively stable conditions in the core and transport networks, where resources can be reserved with higher predictability, the RAN must contend with dynamic factors such as fluctuating signal strength, link adaptation schemes, interference, user mobility, and varying traffic demands. All of these factors create a dynamic and unpredictable landscape in the RAN that complicates the allocation and isolation of resources and makes it more difficult to guarantee strict Service Level Agreements (SLAs) for latency, throughput, or reliability.

The same difficulties arise in WLAN, where, in addition, proposals addressing end-to-end (E2E) slicing for WLAN are relatively scarce. Furthermore, in terms of RAN slicing, the differences of WLAN from 5G NR are obvious. Unlike 4G and 5G NR, which operate mainly in licensed spectrum with stricter interference management, IEEE 802.11 standards operate exclusively in unlicensed spectrum, making Wi-Fi networks more vulnerable to external interference from other devices and networks. In addition, IEEE 802.11 relies on Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA), a contention-based protocol in which devices compete for access to the medium. This can result in collisions and retransmissions, leading to limited control over latency and throughput. Unlike the centralized scheduling employed in 4G and 5G NR, which allocates resources deterministically and facilitates fine-grained Quality of Service (QoS) management, CSMA/CA inherently

* Corresponding author.
E-mail address: anhersol@unizar.es (Á. Hernández-Solana).

https://doi.org/10.1016/j.comnet.2026.112289
Received 10 September 2025; Received in revised form 22 March 2026; Accepted 29 March 2026
Available online 3 April 2026

restricts such control. Consequently, ensuring QoS for different "slices" in Wi-Fi becomes considerably more challenging. The most commonly proposed approach for enabling network slicing in Wi-Fi involves airtime allocation—defined as the portion of time a transmission from an Access Point (AP) to a Station (STA) occupies the shared wireless medium. After establishing the basis for resource allocation, the next step is to determine which flows, tenants, or services should be mapped to individual slices, guiding how resources will be distributed across them. In the literature, two strategies represent the main perspectives on resource segmentation. In this work, we adopt the terminology introduced by Richart et al. [1] to refer to these perspectives in the context of RAN slicing in Wi-Fi.

- The Infrastructure-Sharing Slicing (ISS) corresponds to a user-oriented approach. It focuses on distinguishing between categories of users, or partitioning network resources based on agreements—such as subcontracting to third-party entities—thereby ensuring isolated and dedicated resource allocation for each user group or tenant.
- The Quality-of-Service Slicing (QoSS) is a service-oriented approach, where slices are defined by specific QoS requirements (e.g., latency, bandwidth, reliability). It aims to separate traffic according to service or application types, such as real-time communications, bulk data transfers, or background services. The infrastructure dynamically adapts resource allocation to meet QoS parameters, typically expressed as SLAs, regardless of the user category. A major challenge with QoSS lies in managing multiple concurrent slices, while preventing conflicts in resource demands that could compromise QoS guarantees. It is therefore essential to ensure that slices do not interfere with each other.

In this work, we propose a hybrid slicing approach that integrates these perspectives in a complementary way. First, ISS is used to segment network resources into multiple slices, each tailored to specific user categories—for example, faculty (professors and staff), students, and guests in a campus network environment. Building on this, a second layer of QoSS is embedded within the ISS framework. Within each ISS slice, applications and services are further differentiated based on their QoS requirements—such as latency sensitivity, bandwidth demand, and reliability—and are provisioned with dedicated resources accordingly. The proposed ISS-QoSS slicing framework is implemented on a Software Defined Networking (SDN) based experimental prototype, featuring global and distributed slicing components located at a central controller and distributed agents (APs). The focus is on the downlink, with particular attention to the design of the local slicing component. Here, incoming data flows are classified into different queues based on their assigned slice and service class. Through a hierarchical scheduling approach (inter-intra slice scheduling), based Airtime Deficit Weighted Round Robin (ADWRR) scheduling principles, the specific practical implementation ensures isolation and resource segmentation among tenants or client categories, in addition to fine-grained traffic differentiation, prioritization or resource isolation for various service classes within slices. The proposal includes local adaptations of allocated partitions or portions of airtime—known as quantum—at both the ISS and QoSS levels. These local quantum adaptations modify the standard ADWRR reallocation to better meet specific QoS requirements, enhancing overall network performance and user experience.

The work focuses on practical implementation issues affecting complexity and feasibility in real conditions. The main contributions include a lightweight ADWRR-based method for airtime quantum distribution, supporting dynamic queue traffic activation/deactivation without relying on complex optimizations. In addition, unlike traditional incremental approaches, the proposed method enables coordinated quantum redistribution across slices, allowing for better fairness, predictability, and control over inter-slice interactions. Furthermore, accurate airtime estimation is based on actual physical rates and retransmissions, improving realism over theoretical models. The procedure is designed to be aware of IEEE 802.11 hardware and software characteristics while maintaining independence from specific implementations.

The rest of the paper is organized as follows. Section 2 reviews the state of the art in RAN slicing at WLAN. Section 3 presents the SDN system architecture, providing an overview of the RAN slicing framework, while Section 4 includes a full description of the proposed slice design at local agents, including a detailed description and discussion of the issues concerning its practical implementation. Section 5 describes the testbed, and presents the performance analysis of the proposal. Section 6 describes the design of dynamic adaptation proposal and presents and discusses the corresponding performance results. Finally, Section 7 draws the conclusions.

2. Related work: RAN slicing at WLAN

Airtime allocation is the most common approach to network slicing in Wi-Fi. Most existing solutions focus on downlink resource allocation. To the best of our knowledge, only few works address uplink slicing, all relying on mechanisms introduced in IEEE 802.11ah and 802.11ax to reduce STA contention under AP control. In particular, [2] proposes a scheduler for dynamic slicing in IEEE 802.11ah focused on uplink traffic, leveraging the Restricted Access Window (RAW) mechanism. In this standard, STAs can access the channel only through RAWs, which are divided into equal-duration slots. These slots are managed by the AP using RAW Parameter Set (RPS) beacons that announce the number of RAWs, slot configurations, and STA assignments. The slicing algorithm in [2] follows four main steps: (1) assigning one STA per slot to avoid collisions; (2) defining the airtime required for each slice based on traffic and modulation and coding scheme (MCS); (3) configuring slot and RAW sizes according to each slice; and (4) dynamically adjusting the number and size of RAWs based on the number of STAs and their traffic patterns.

Regarding IEEE 802.11ax, it introduces Orthogonal Frequency-Division Multiple Access (OFDMA), which enables uplink and downlink slicing by dividing the spectrum into resource units (RUs). The AP assigns RUs via trigger frames (TFs) that specify selected STAs and their transmission parameters (e.g., RU, power, MCS), enabling collision-free uplink scheduling. This mechanism facilitates slicing by controlling the number of RUs allocated to each one with the appropriate scheduling algorithms. However, only very recent simulation-based works explore it so far: [3] uses an Actor–Critic algorithm for RAN slicing, while [4] focuses mainly on networking of 5G slicing with Wi-Fi 6, excluding details about RAN slicing.

Excluding the features of 802.11ah/ax, uplink slicing would rely on the Point Coordination Function (PCF), whose polling limitations are well known. Consequently, existing proposals focus on downlink slicing in 802.11n/ac, where the problem remains challenging. While downlink airtime scheduling principles can be adapted for the uplink, the main challenge is accurately obtaining the STAs' uplink requirements.

One of the earliest and most cited works on end-to-end (E2E) network slicing in real WLANs is [5]. It proposes a slicing framework using the 5G-EmPOWER platform [6], enabling centralized network management via Python-based applications on a controller. Each Wi-Fi interface at an AP (defined as a Resource Block [7]) classifies frames into queues per slice according to traffic rules (e.g., OpenFlow). Frames are dequeued using the ADWRR scheduler [8–10], while STAs within each slice are served via simple Round Robin. This proposal is evaluated using only static quantum allocation, which aligns more with ISS than QoS slicing. Although experiments were limited—for example, assigning equal quantum to all slices—results indicate that ADWRR can provide effective resource isolation. However, the evaluation relies on comparing achieved rates per STA and slice, which is imprecise and not generalizable. ADWRR is intended to guarantee airtime allocation proportional to the assigned quantum, not throughput. Rate-based

comparisons are unreliable in real systems, as airtime consumption varies with packet sizes, MCS, and error rates. Further study is required to address and evaluate all factors affecting practical operation, including detailed aspects of airtime estimation.

Authors in [11] use the same 5G-EmPOWER framework and extend ADWRR to adopt a QoS slicing approach, creating slices for QoS and Best Effort (BE) flows. Delay-sensitive slices are assigned larger quantum values, giving higher priority through potential access to a larger portion of the radio resources. However, the relationship between quantum allocation and achievable QoS is uncertain due the variability of the environment, MAC layer parameters of individual transmissions, and radio conditions. Quantum value selection alone cannot ensure QoS control. To address this, in [11], the authors propose a heuristic quantum adaptation algorithm that triggers adjustments to slice quantum values whenever the application's QoS requirements are not met. The problem is that independently adapting each slice—using increment or decrement factors—limits control over resource partitioning and the overall QoS impact on other slices, which is a common drawback in adaptive quantum approaches. In [7], the same authors optimize quantum allocation using a Quadratically Constrained Quadratic Program (QCQP) in order to minimize the overall queueing delay of the entire system. The proposal was validated through experimentation with real hardware, but a key drawback is the limited applicability of these techniques in real time. Similarly, authors in [12] propose a Deep Reinforcement Learning (DRL) in a 5G-enabled SDN to adjust slice quantum based exclusively on throughput satisfaction. AI and ML approaches can improve network resource utilization if network dynamics are accurately captured. ML-based quantum tuning, as in [12], is effective at certain time scales and controller levels but does not replace the need for efficient heuristic local scheduling at APs. Effective local scheduling requires understanding the combined impact of many RAN stack parameters, particularities of MAC, physical and hardware behavior, and configuration, such as driver's queue sizes, discard policies, frame sizes, aggregation options [16], ACK policies, and many more.

In addition, a key aspect of slicing—also in QoS slicing—is to ensure that slices remain independent, avoiding performance degradation or resource appropriation between them. Allocating specific airtime ratios by itself does not guarantee QoS, and maximizing throughput or minimizing delay does not ensure slice isolation. Slicing architectures should control traffic according to SLAs, detect and handle violations, and use airtime efficiently without restricting traffic when resources are available. The problem is often formulated as a stochastic optimization problem [13–14]. In [15], the authors propose a dynamic adaptation scheme to fulfill SLA contracts over multiple APs using quadratic programming. However, incorporating all required constraints is challenging, and most proposals are evaluated only by simulation.

Authors in [1] adopt an alternative ISS slicing approach called Adaptive Time-Excess Round Robin (ATERR), where each slice is guaranteed a portion of an AP's total airtime, while unused airtime is shared proportionally among other slices. ATERR is essentially an adaptation of ADWRR [10]. It uses time excess instead of deficit, but the overall concept is equivalent. In ATERR, each slice maintains a queue for each user's traffic flow, allowing users to participate in multiple slices. The quantum of each slice is equally distributed among its active queues (those with packets), to achieve fairness among users within the same slice, while each slice is allocated a specific ratio of the AP's airtime. By establishing the mathematical relationships between the quantum and a minimum allowable quantum for each queue, the work dynamically calculates the quantum values each time a queue becomes active or inactive. In our opinion, the same objectives can be achieved with a simpler implementation and extended to manage differentiated airtime shares among STAs.

Concerning the extension of approaches to multi-AP scenarios, although many SDN-based frameworks support programmable and dynamic network slicing in WLANs, most evaluations proposals and evaluations are truly limited to a single AP [7,11,12]. In [17], the authors of [7,11] propose a delay-aware slicing including an AP selection/association and load balancing algorithm based on Multi-Criteria Decision Analysis (MCDA), but in a scenario with two APs, which will be extended in [18]. A variant of the ADWRR policy at the AP level is proposed in [19], and the need for dynamic adjustment of the weights per AP is highlighted. Similarly, in [20], the authors introduce an airtime-based scheduling approach (not ADWRR) that implements slice isolation over a set of APs. Two heuristic schedulers—global and local—are proposed to manage resources and address QoS violations using a credit-based scheme. When a policy is violated, the global scheduler updates the airtime allocations in the local scheduler. APs are competing for the same channel, so the problem is similar to those for only one AP. Although it may be possible in high-density deployments, this scenario constrains the solution and is not the typical one to be addressed. In both cases [18,19], no concrete or clearly defined algorithm for network-wide weight adaptation is provided, leaving open questions about how to effectively manage airtime slicing across multiple APs under varying spatial and temporal load demands. In [15], the authors propose a Global-Airtime Deficit Round Robin (G-ADRR) scheme, which dynamically adapts a local set of airtime weights per-slice in each AP, in order to fulfill an SLA contract over a set of APs. The SLA is defined by the fraction of airtime reserved for a slice over a set of APs and the sliding window where this fraction of airtime needs to be met across the set of APs. The mechanism is formulated as quadratic programming problem but, as we mentioned before, a drawback of this optimization approach is the difficulty of including many additional real-world specific constraints or requirements that condition the problem.

In addition, concerning local schedulers, whereas many previous works are based on network simulations, several other works provide experimental demonstrations using commercial Wi-Fi modules. Among them, in [15], authors use custom implementations in the wireless drivers/kernel. In [21], a flexible airtime-based scheduler—named PoliFi—is presented that operates at the mac80211 level and can therefore work with different Wi-Fi module vendors. Others prefer schedulers that are not embedded into them [20] (user space).

With a different objective, the authors in [22] focus on the STA association problem in Wi-Fi networks, considering IoT resource requirements and the 5G slicing concept. Three options are compared: formulation as a Mixed Integer Linear Program (MILP), a heuristic algorithm, and a Reinforcement Learning algorithm. However, this kind of study, theoretically formulated and validated only through simulation, does not delve into practical implementations with real hardware, thus avoiding real physical phenomena and some engineering challenges, such as modifications to the packet scheduler of a Wi-Fi AP. The same occurs with other works as presented in [23]. The authors proposed a 5G-EmPOWER based framework that dynamically creates and manages slices in an SDN-controlled network, but no details about radio scheduling are disclosed.

In our opinion, it is clear that effective slicing implementation in multi-AP environments remains an open issue, as does QoS control and resource orchestration between slices. Beyond global-level mechanisms such as dynamic quantum adaptation or STA association, any effective solution for real-world environments must first include, as a prerequisite, a precise and well-designed local scheduling at the AP level. Moreover, many of the proposals analyzed overlook numerous practical implementation aspects.

At this point, this work proposes a novel hybrid ISS-QoSS approach for the local slicing component. Through a hierarchical scheduling mechanism based on ADWRR principles, it ensures both isolation and resource segmentation among tenants or client categories, as well as fine-grained traffic differentiation, prioritization, and resource isolation for various service classes of traffic application types within slices. Additional, local quantum adaptations at two levels (ISS-QoSS), compatible with global-level orchestration algorithms, enhance specific QoS requirements by adjusting the natural ADWRR reallocation. The

approach focuses on practical implementation challenges affecting complexity and operational feasibility under real conditions, including:

1) Quantum distribution: Queues within a slice receive quantum via an ADWRR—based scheme that handles queue activation/deactivation without complex mathematical optimization, enhancing simplicity and robustness.
2) Quantum adaptation: Controlled redistribution among slices replaces incremental/decremental adjustments. This approach enables precise anticipation and management of inter-slice effects, ensuring effective resource redistribution while preserving fairness and predictable behavior.
3) Airtime estimation: Unlike many works (e.g., [5,7,11,12]) that rely on simplified estimations of the expected MCS—omitting details on the estimation procedure as averaging window, delay, or the effects of inaccuracies—this work uses the real MCS/physical rate. Deficits are then corrected based on the observed number of retransmissions.
4) Implementation awareness: The procedure is designed to be aware of IEEE 802.11 hardware and software characteristics while maintaining independence from specific implementations. This ensures compatibility across devices from different manufacturers.
5) Real-world testing: Methods are validated under fluctuating traffic and channel conditions, a variety of STA traffic patterns, rate control behavior, fragmentation, and other relevant factors.

3. Architecture implementation

The experimental prototype developed in this work integrates the proposed RAN slicing functionalities and is built upon the open-source framework described in [24], which is itself derived from [25]. The high-level architecture, illustrated in Fig. 1, comprises a central controller and a set of distributed agents (APs).

The central controller aggregates information from all APs and enables network applications to make intelligent and orchestrated decisions regarding client assignments to APs, load balancing (e.g., proactive and reactive handoffs), and coordination and resource management among APs, including QoS provisioning. These applications operate both reactively and proactively, leveraging measurements from multiple layers collected by the APs.

At the local level, software implemented on the agents associated with each AP handles non-time-critical MAC functionalities, operating strictly within the policies and settings defined by the central controller. This distributed setup optimizes QoS and resource utilization locally within each AP cell. In contrast, critical aspects of the Wi-Fi MAC protocol—such as Distributed Coordination Function (DCF), timing, and acknowledgments—are managed by the mac802.11 kernel, regardless of the specific driver used (see Fig. 2).

3.1. SDN-based Wi-Fi RAN slicing architecture overview

Based on the previously described architecture [24], the SDN-based Wi-Fi RAN slicing solution adopts a hierarchical design that integrates a centralized slicing control and local (distributed) components, to manage slicing in dense Wi-Fi deployments. Through the control plane (shown in Fig. 1), the controller and agents exchange messages via TCP and UDP, coordinating the centralized and distributed components.

The centralized slicing component periodically computes and adjusts the guaranteed fractions of radio resources intended to each RAN slice instantiated on every AP. Decisions are based on network players (tenants) requirements, STA distribution, traffic demands, and measured QoS metrics. Resource partitions are expressed as per-slice airtime fractions and enforced locally by the slicing components at each AP, ensuring isolation among coexisting slices.

It should be noted that, in this work, slicing is limited to the downlink, with slice scheduling policies based on a revised version of ADWRR/ADRR. Consequently, guaranteed resource partitions and slice isolation are managed indirectly through the selection of quantum values, while the scheduled airtime corresponds to the available downlink airtime, excluding periods when the channel is occupied by other transmissions.

The proposed solution is designed to operate in a controlled infrastructure deployment where Wi-Fi channels selection and configuration prevent channels reuse among APs with overlapping cells. This setup limits interference to that caused by external devices or networks. Under this assumption, each AP operates independently within the quantum values enforced by the centralized slicing component. Airtime is managed locally through the scheduling component, which allocates resources according to the connected STAs and traffic. As long as the centralized slicing component does not re-orchestrate local settings—for instance, in response to network dynamics such as inter-cell mobility—each AP preserves performance isolation with respect to the rest of the deployment. Consequently, experimental results obtained at a single

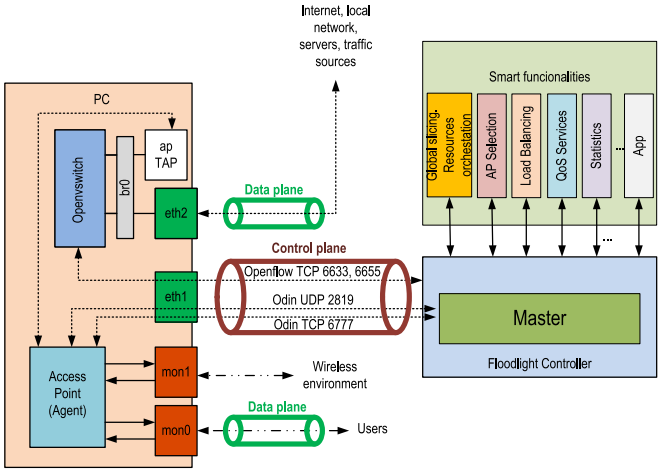


Fig. 1. High level architecture of the solution.

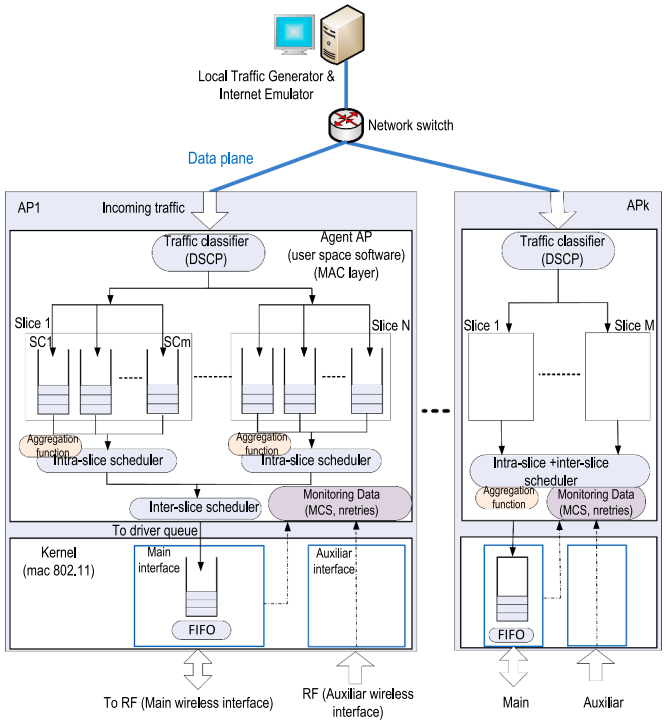


Fig. 2. Local implementation in the APs (data plane).

AP accurately characterize local slicing behavior. These results are therefore generalizable, independently of the presence or number of APs in the deployment.

Should a deployment intentionally enable channel reuse—an option not excluded in future scenarios, particularly considering proposals for IEEE 802.11ax and beyond—the slicing framework would require partial adaptation at both global and local components. In such cases, resource allocation would be performed through a global scheduling mechanism at the SDN controller, while still accounting for local slicing components. In global scheduling, APs sharing the same channel would be treated as a single logical entity to mitigate inter-AP interference.

It should be noted that, in all the cases, occasional collisions caused by hidden node transmissions may result in transmission errors. These errors are handled in the same manner as those arising from channel conditions or rate adaptation mismatches.

In the current context, the focus of this work is placed on the local/distributed slicing component. The details of the local implementations at the APs supporting local slicing are presented in this section, while the slicing implementation itself is described in Section IV. The design of the centralized slicing component is beyond the scope of this study, with only its main functionalities briefly outlined in the next subsection.

3.2. Centralized slicing component and slice lifecycle

The centralized slicing component operates at the SDN controller and is primarily responsible for orchestrating all slicing functionalities in a WLAN environment, covering the entire slice lifecycle:

1) *Preparation:* defining ISS and QoSS slices resources and performance requirements (e.g., SLA), assigning per-user-class and per-service-class identifiers, and selecting which APs will host each slice.
2) *Instantiation*: creating and activating slices by configuring the chosen APs (e.g., provisioning user-class/service-class tags or identifiers and initial airtime quantum values).
3) *Configuration and Adaptation:* dynamically adjusting airtime quantum values and the selected scheduling policies to be applied on each AP (local agent) to accommodate evolving traffic patterns due to mobility, load balancing, new STA associations, and other factors. This functionality can be identified as the centralized component of the slicing referred above. Its main purpose is to orchestrate adaptively the slice's quantum allocation at each AP (we identified later as nominal quantum), considering the set of slices instantiated on the APs, the associated traffic distribution across the entire deployment, and monitoring information.
4) *Monitoring:* collecting real-time ISS and QoSS slices metrics—such as allocated resources per AP, slice isolation, throughput satisfaction, latency, and retransmissions—and triggering corrective actions centrally (global adaptations), which are then communicated to the local slicing components at the AP agents.
5) *Termination:* decommissioning slices by removing user-class and service-class identifiers and releasing all associated resources.

3.3. Local agent architecture supporting local slicing level

The local slicing component constitutes, as in other related proposals, an essential element for effective execution of the RAN slicing solution. It is responsible for enforcing slice-based resource allocation at the AP. It is implemented at the local agent operating in user space above of IEEE 802.11 MAC layer of the wireless cards and it controls packet transmission through airtime-aware scheduling mechanisms.

Fig. 2 provides an overview of the local data-plane operation at the AP and the interfaces involved. Traffic arriving from the wired interface is first inspected and classified by the local agent, which maps packets to a specific slice according to predefined policies. Within each slice, packets can be further differentiated based on service class, STA identity, or other criteria. Classified packets are then enqueued in a hierarchical queue structure that supports both inter-slice and intra-slice scheduling.

An asynchronous thread manages the scheduling of queues using a hierarchical approach that accounts for both inter-slice and intra-slice criteria. As mentioned above, the implemented slice scheduling policies are based on a modified version of the well-known ADWRR (or ADRR) algorithm. These policies aim to ensure slice isolation according to the nominal quantum values configured and orchestrated by the centralized slicing component, while allowing local, real-time adaptations around those values based on monitoring feedback. Nominal quantum values are assumed to be communicated semi-statically by the central controller and remain constant over time intervals significantly longer those associated with local adaptations.

Selected packets are transmitted through the main wireless interface, which operates in monitor mode with frame injection enabled. In parallel, a monitoring process collects real-time transmission-related information—such as buffer occupancy and retries—either from driver feedback or, when unavailable, through an auxiliary wireless interface operating in monitor mode. This monitoring information is used to support rate control, airtime estimation, queue management, and dynamic adjustments of the local scheduling behavior.

The following subsections detail the practical aspects that enable this local processing pipeline, including hardware compatibility constraints, required interfaces, monitoring mechanisms, and operational requirements of the wireless interfaces.

3.3.1. Hardware and compatibility

The local agent is linked to the mac80211 module and works with any Linux-compatible Wi-Fi chipset that supports frame injection in monitor mode. Our approach excludes modifications to the low level drivers (e. g. in ath9k_htc, like in [26]) in order to implement the local airtime-based scheduler. Thus, a key characteristic of this work is the development of a practical and hardware-agnostic Wi-Fi RAN slicing approach, which does not rely on any specific 802.11 driver implementation. In addition, the user-space implementation, together with monitor mode and frame injection, facilitates the implementation and evaluation of the proposed approaches, making them well suited for experimental Wi-Fi environments.

Frame injection in monitor mode enables Wi-Fi cards to capture and transmit all frames (data, control and management) on a specific channel, regardless of the network they belong to. This mode optionally allows us to select a preloaded association of STAs with APs prior to network operation. Without loss of generality, promiscuous reception makes it possible to ignore the transmission and reception of management frames (e.g., beacons, probes, association, etc.) and focus exclusively on analyzing and testing traffic slicing metrics, thereby avoiding unnecessary overhead. The use of Radiotap, the de facto standard for IEEE 802.11 frame injection and reception [27], enables active transmission of crafted or modified frames, which is essential for testing scenarios such as network slicing. It supports all Wi-Fi frame transmission formats, including IEEE 802.11ax, and allows configuration of channel bandwidths, spatial streams, guard intervals, and other transmission parameters defined in the standards.

We emphasize that using monitor mode at the AP does not compromise the feasibility of deploying the approach in real-world scenarios. STAs (e.g., commercial phones) operating in managed mode can still interact with the SDN system, provided that the infrastructure agent software keeps the transmission and reception of all required Wi-Fi management frames (e.g., beacons, probes, associations) active.

3.3.2. Required interfaces

For the specific tests performed in this work, the prototype does not require the use of the SDN controller in the br0 interface for its traffic filtering function towards the connected STAs. However, it maintains both the wired and wireless interfaces, with the same functionalities that they already had.

- The wired interface (eth2 in Fig. 1) receives frames from the Internet, local servers, or traffic sources.
- The main wireless interface in Fig. 2 (mon0 in Fig. 1), configured to support frame injection in monitor mode, enables the AP to capture and transmit all frames (data, control and management) on a specific channel.
- The auxiliary wireless interface in Fig. 2 (mon1 in Fig. 1), also configured in monitor mode, can retain its specific monitoring functions [24] such as supporting proactive and reactive handovers or monitoring all frames transmitted and received by the main interface for feedback or statistical purposes.

3.3.3. Monitoring process

In this work, the auxiliary interface is used for monitoring purposes to obtain real-time information about: buffer driver occupation at the main interface, rate and retries when direct feedback from the network driver is not accessible or supported. The existence of an auxiliary interface in monitoring mode does not pose a limitation on the practical operation of the system in real-world scenarios. However, whenever it is possible to obtain feedback directly from the driver used in the AP deployment, this interface is not used.

Note that, the slicing scheduling proposals in this work are defined above the mac80211 level. The AP agent, defined in the user space, can work with different Wi-Fi module vendors. Some driver implementations (particularly those integrated into the Linux kernel) can, in principle, obtain and report about what happened during a transmission. Examples include the Radiotap TX-Flags and data retries fields, which provide feedback that eliminates the need for an additional interface. However, we have not been able to access commercial devices that faithfully implement this functionality as described. That is, the mac80211 stack does not natively include real-time functions for logging or updating the status of outgoing transmitted frames, such as retry counts or rate control metrics, in a way that is accessible by the agent through standardized interfaces. Thus, driver modifications are required in these cases if no auxiliary interface is used. It is possible to modify the driver to add such functionality in the transmission handling related functions such as ieee80211_tx() and ieee80211_tx_status(), but this approach has not been considered.

Furthermore, our testbed does not utilize driver-specific features such as rate control or aggregation due to the difficulty of managing them jointly with frame injection. Instead, these functionalities are implemented by the software agent. Specifically, a modified version of ONOE rate algorithm is considered although, the specific rate control mechanism (e.g., Minstrel or Minstrel-HT defined on *mac80211*) is not relevant from the perspective of the slicing scheduling. Feedback information required by the rate control algorithm could be derived, if possible, from the driver's tx_info or similar structures, which log retry attempts for each frame. However, in this case, it is obtained from the monitoring process defined at the auxiliary interface. This process constitutes a viable-testbed-based alternative that replaces the desirable direct feedback from the driver (illustrated in Fig. 2).

The monitoring process collects per-frame information from all frames received by the auxiliary interface, including Radiotap parameters (e.g., MCS), IEEE 802.11 header fields (source and destination addresses, sequence number, packet length, retry flags), and IP-layer information such as the Differentiated Services Code Point (DSCP) and packet length. Regarding the frames captured at the main or auxiliary wireless interfaces, the software discards those that are not addressed by or to the AP, and properly manages the other. Captured information enables functionalities such as:

- Rate control, and thus knowledge of the MCS used by the main interface (used on airtime estimation).
- Control over the number of frames pending transmission in the physical driver, which is required because rate control is defined at the software agent. In addition, it allows control over the queue overflow at that level.
- Estimation of the actual transmission time for each data packet. This allows for corrections to airtime consumption based on the number of retries.
- Collection of statistics to gather metrics (e.g., throughput, delay, MCS, airtime usage per STA, queue or slice, etc.) for evaluation purposes and to define dynamic quantum adjustments linked to ADWRR-based slicing strategies.

3.3.4. Wireless interface operational requirements

The main and auxiliary 802.11 interfaces of each AP must operate on the same physical channel (e.g., on 5 GHz) and with identical bandwidth settings (e.g., 20 MHz, 40 MHz, etc.). While uniformity in hardware manufacturers is not required (in fact, multi-vendor deployment is considered), main and auxiliary interfaces must support similar capabilities (e.g., HT or VHT), regardless of the specific features of the connected STAs. This ensures there are no limitations on monitoring capabilities of the auxiliary interface (e.g. it captures all possible rates).

To ensure optimal monitoring capabilities, the antennas of the main and auxiliary interfaces need to be positioned close to each other but sufficiently separated to avoid the blocking of the passive auxiliary interface. Additionally, the most critical requirement for spatial diversity and MIMO spatial multiplexing MCS options is that the channels affecting the multiple antennas should be uncorrelated. The radio wave propagation conditions should be sufficiently different for each Tx-Rx antenna pair between the main and auxiliary interfaces, to ensure that different data streams can be transmitted simultaneously through multiple antennas and received effectively at the auxiliary interface. That is, we need to ensure that the auxiliary interface is able to monitor frame transmission using MCSs/rates that apply MIMO spatial multiplexing. For instance, placing a reflector between the antennas is an effective strategy for reducing channel correlation in MIMO systems or configurations with closely positioned antennas.

4. Slicing design and practical implementation issues

As defined previously, the IEEE 802.11 RAN consists of a set of APs responsible for delivering downlink traffic from network services to their associated STAs. This section focuses on the design of the local slicing mechanisms implemented at the APs, addressing their logical structure as well as the key implementation considerations required for correct operation, and affecting complexity and feasibility in real-world deployments

4.1. Queue architecture and traffic classification

Regarding the local architecture, incoming IP packets at each AP are temporarily stored in the socket buffer and classified into queues according to predefined traffic rules.

Our design focuses on the ISS slicing concept defined by Richart et al. [1], while introducing a second level of slicing within each slice to manage multiple Service Classes (SCs), resulting in a hybrid ISS–QoSS approach.

This approach, while maintaining some similarities, differs from the pure ISS or QoSS schemes described in Section 2, where each slice typically maintains a separate queue per STA and applies round-robin or equal airtime sharing among STAs associated with it [1]. In our architecture, packets are classified by service class rather than by individual STA. Consequently, traffic from different STAs belonging to the same service class within a slice may share a common FIFO queue, while a single STA may generate traffic mapped to multiple queues within the same slice, depending on the type of service flow.

As shown in Fig. 2, incoming IP data packets are first associated with an AP based on the destination STA, whereas slice and queue identification are based on the Differentiated Services Code Point (DSCP) field

in the IP header. The 6-bit DSCP allows hierarchical traffic mapping by using higher-order bits for slice identification (e.g., 3 bits allow 8 slices) and lower-order bits for intra-slice service differentiation (e.g., 3 bits allow up to 8 Service Classes).

4.2. Hierarchical slice scheduling principles

Our hierarchical slicing approach aims to ensure that resources are managed at the slice level (through an inter-slice policy based on the ISS concept), ensuring isolation and resource segmentation among tenants or client categories. In addition, within each slice, resources are further managed through a second slicing level (e.g. intra-slice policy connected to the QoSS concept), enabling fine-grained traffic differentiation, prioritization, or resource isolation for various service classes, even when multiple STAs are sharing the same queue. We emphasize that at the agent level, the primary goal of this hierarchical scheduling implementation is to ensure resource isolation at both inter- and intra-slice levels. Once resources are guaranteed, any remaining capacity can be allocated according to various policies. However, to ensure that the hierarchical scheduler meets specific metrics (e.g., QoS requirements), the centralized slicing component (controller) must adaptively adjust the resource partitions, as noted above.

Inter-slice scheduling is based on the principles of the ADWRR scheme. Regarding intra-slice slicing, various alternatives are possible; however, we also opt to develop schemes derived from ADWRR to ensure isolation among service classes (SCs). For clarity, we briefly introduce the basic principles and notation of the well-known ADWRR scheduling discipline before describing the particularities of our implementation.

According to the ADWRR scheduling, each service flow contending for a link has a corresponding queue with an associated count called Deficit Counter (DC), which in our case indicates the amount of airtime (in μs) the flow can use in the round. Queues are visited in a round robin fashion. Upon each visit, the DC is firstly increased by a fixed quantity: the called quantum (Q). Then, packets in the queue are selected for transmission as long as the airtime required by each packet is smaller than the DC's current value, knowing that DC is decreased by the airtime of each selected packet after each one is sent. Otherwise, if DC is smaller than the required airtime, the queue is skipped, and the process continues with the next queue. When queues become empty, DC is set to zero.

4.3. Inter and intra-slice scheduling mechanisms

The pseudocode of the implemented inter-slice and the embedded intra-slice scheduling is presented in Algorithms 1 and 2 respectively, with the notation used defined in Table 1. Improvements, including local quantum adaptations to meet specific QoS requirements and modifications to the natural ADWRR reallocation of unused slice or queue resources, are introduced in Section 6. These enhancements aim to refine resource allocation, ensuring a more precise alignment with the performance needs of individual service classes and slices.

With N_{slices} denoting the number of all slices instantiated in a given AP and N_{queues} *[s]* representing the number of queues instantiated in the slice *s*, the quantum of a slice *s, Q* [*s*], is set to the nominal value (Q_{nom} [*s*]). This nominal value is selected by the centralized slicing component (global controller) to ensure that the proportion of the AP's airtime allocated to slice *s* meets its performance and resource requirements. The actual portion allocated to slice *s* depends on its own quantum (*Q [s]*) and the sum of the quantum of all the active slices. Specifically, when a slice *s* has pending traffic in its queues, it is guaranteed to receive at least a portion of AP's airtime, as shown in Eq. (1):

$$Q[s] \Big/ \sum_{j=0}^{N_{slices}-1} Q[j] \tag{1}$$

Additional airtime can be received if not all competing slices are active. As shown in Algorithm 1, upon each visit, the deficit counter of the slice *s* (*Deficit [s]*) is increased by *Q [s]*, but only if slice *s* has pending traffic.

At the second scheduling level, the portion of the AP's airtime corresponding to slice *s* is shared among its N_{queues} *[s]* according to their respective quantum values *Q [s,i] for* $i = 1,..,$ N_{queues} *[s]*. These quantum values are derived from a set of weights configured by the global controller, and they are dynamically computed at the beginning of each visit to the associated slice. Instead of setting the quantum directly, the global controller provides a dimensionless weight (*W [s,i]*) to ensure that the portion of slice *s*'s airtime allocated to each queue *i* satisfies the performance and resource requirements of its service class. The allocation depends on *W [s,i]* relative to the sum of the weights of all active queues within slice *s*. Any queue *i* with pending traffic is guaranteed to receive at least a portion of slice *s* airtime as expressed in Eq. (2):

$$W[s,i] \Big/ \sum_{j=0}^{N_{queues}[s]-1} W[s,j] \tag{2}$$

The per-round quantum *Q [s,i]* (in μs) for each non-empty queue within slice *s* is then computed from *Q [s],* as shown in Eq. (3) (lines 1–3 of Algorithm 2). In addition, the corresponding deficit counter (*Deficit [s,i]*) is increased by *Q [s,i]* (line 4 in Algorithm 2). It is important to note that the ratio of the derived quantum values match those of the weights, but while *Q [s,i]* is dynamically computed, *W [s,i]*=W_{nom} *[s,i]* remains unchanged.

$$Q[s,i] = \left(W[s,i] \big/ w_{QueueNotEmpty}[s]\right) \cdot Q[s] \tag{3}$$

Table 1
Variables ADWRR.

Variable	Description
airtime [*s,i*]	Airtime estimated for a packet of queue *i* of slice *s* (without retries).
Q [*s*]	Quantum of slice *s* (configured by the controller).
W [*s,i*]	Weight of queue *i* of slice s (W_{nom} configured by the controller).
Deficit [*s*]	Deficit of slice *s*.
Deficit [*s,i*]	Deficit of queue *i* of slice *s*.
$Deficit_{retries}$ [*s*]	Accumulated deficit of slice s due to retries.
$Deficit_{retries}$ [*s,i*]	Accumulated deficit of queue *i* of slice *s* due to retries.
N_{slices}	Number of slices.
N_{queues} [*s*]	Number of queues in slice *s*.

Algorithm 1
Inter-slice scheduling ADWRR.

Algorithm 1 Inter-Slice Scheduling ADWRR

```
1:  s = 0
2:  loop every
3:      As long as the physical driver's buffer occupation==threshold, it waits
        #Update the deficit of the slice s. Only if it has data to send
4:      if Slice[s].empty == false then   ▷ Queues of slice s have packets
            ▷ Update deficit with quantum and deficit of the slice s because
              of retries of previous transmissions. Deficit_retries is always ≥ 0
5:          Deficit[s] += Q[s] - Deficit_retries[s]
6:          Deficit_retries[s] = 0
            #Send slice's packets while Deficit[s]>0 and there are packets
7:          with airtime smaller than Deficit
8:          INTRA-SLICE ALGORITHM for serving queues (Algorithm 2)
9:      end if
        #It continues with the next slice
10:     if s < (N_slices - 1) then s++
11:     else s = 0
12: end loop
```

Algorithm 2
Intra-slice algorithm for serving queues of slice Option ADWRR.

Algorithm 2 Intra-Slice algorithm for serving queues of slice Option ADWRR

#Compute deficit of all the queues of slice s
$w_{QueueNoEmpty}[s] = \sum_{\forall i_{empty} \in s} W[s,i]$

for each queue i not empty of slice s **do**
  $Q[s,i] = (W[s,i]/w_{QueueNoEmpty}) * Q[s]$
  $Deficit[s,i]$ += $Q[s,i] - Deficit_{retries}[s,i]$
  $Deficit_{retries}[s,i] = 0$
end for

#Dequeueing process
$q_{ini} = Slice[s].Queue\_index$ ▷ Store the initial queue index for round
$i = q_{ini}$
while $Deficit[s] > 0$ **do**
  while $Deficit[s,i] > 0$ and $Queue[s,i]$.empty == false **do**
    As long as the physical driver's buffer occupation==threshold, it waits

    #Non Packet Aggregation Option
    ▷ Calculate airtime of first packet of *Queue[s,i]*
    $airtime[s,i] = Queue[s,i]$.airtime(head packet)
    $npAggr = 1$

    #Packet Aggregation Option
    ▷ Calculate the number of packets that can be aggregated (*npAggr*) based on the *Deficit[s,i]*, the maximum size of the aggregation, and the number of queue packets (A-MSDU)
    $npAggr = Queue[s,i]$.MaxAggregationAllowed($Deficit[s,i]$)
    $airtime[s,i] = Queue[s,i]$.airtime($npAggr$)

    if $airtime[s,i] \leq Deficit[s,i]$ **then**
      ▷ Dequeue the head packet (A-MSDU) from queue i (one or more if they are to be aggregated) of slice s
      $Queue[s,i]$.dequeue($npAggr$)
      Construct aggregated frame with packets dequeued
      Send(frame)
      $Deficit[s]$ -= $airtime[s,i]$
      $Deficit[s,i]$ -= $airtime[s,i]$

      if $Queue[s,i]$.empty == true **then**
        ▷ The last packet in queue i has been sent. Its deficit is redistributed proportionally among the active queues based on their quantum
        $w_{QueueNoEmpty}[s] = \sum_{\forall i \in s} W[s,i]$
        for each queue j not empty of slice s **do**
          $Deficit[s,j]$+=$(W[s,j]/w_{QueueNoEmpty}[s])$*$Deficit[s,i]$
        end for
        $Deficit_{retries}[s,i] = 0$
        break ▷ Next queue
      end if
    else break ▷ Next queue
    end if
  end while

  #It continues with the next queue of the slice s
  if $i < (N_{queues}[s]$ - 1) **then** i++
  else $i = 0$
  end if
  if $i == q_{ini}$ **then**
    $Slice[s].Queue\_index = i$
    break ▷ Next slice
  end if
  if $Slice[s]$.empty == true **then** $Deficit[s] = 0$ ▷ Next slice
end while

where:

$$w_{QueueNotEmpty}[s] = \sum_{\substack{j \text{ not empty} \\ \in \text{ slices}}} W[s,j] \tag{4}$$

As a result of this process, the hierarchical inter- and intra-slice scheduling works as follows:

- Slices are visited in a round robin fashion (Algorithm 1). Upon visiting slice *s*, if it is not empty, both the slice deficit (*Deficit [s]*) and all its non-empty queue deficits (*Deficit [s,i]*) are incremented by their respective quantum values *(Q [s] in* line 5 in Algorithm 1 and *Q [s,i]* in line 4 in Algorithm 2). If the slice is empty, it is skipped, and the scheduler proceeds to the next slice (lines 10–11 in Algorithm 1).
- Then, the de-queuing process begins with the first queue of slice *s* that was not served in the previous round (tracked by variable q_{ini}, line 7 in Algorithm 2). Packets are de-queued and sent to the driver queue until either the slice or queue deficit counters (*Deficit [s]> 0* and *Deficit [s,i]> 0*, lines 9–10 in Algorithm 2) become insufficient to cover the estimated airtime of the next packet (line 16 in Algorithm 2), or the queue becomes empty (line 22 in Algorithm 2). If *Deficit [s]>0*, the scheduler continues with the next queue in slice *s* (line 33 in Algorithm 2) until all queues have been visited or the slice is empty (lines 36–40 in Algorithm 2). In all cases, after each packet de-queueing, *Deficit [s]* and *Deficit [s,i]* are decremented by the packet's airtime (*airtime [s,i],* lines 20–21 in Algorithm 2).

4.4. Practical refinements and corrections in the scheduling operation

Certain corrections and additions must be emphasized in the scheduling operation, as they are critical to achieving the desired performance and ensuring correct system behavior under real-world conditions.

4.4.1. Deficit redistribution when queues becomes empty

When a queue *i* of a slice *s* becomes empty, its remaining deficit, *Deficit [s,i]*, needs to be redistributed among all the remaining active queues of the slice *s* before being reset (*Deficit [s,i]=0*). Redistribution to each queue *j* is proportional to the ratio between the *W [s,j]* and the sum of weights of all active queues (lines 22–28 in Algorithm 2). If this redistribution was omitted, slice s would not fully consume its allocated airtime share, potentially violating inter-slice resource guarantees. By contrast, this mechanism ensures that any unused airtime from an inactive queue within a slice is reallocated to the remaining queues in proportion to their weights, while preserving intra-slice fairness. Finally, the adopted queue-level quantum adaptation mechanism simplifies implementation, while retaining the flexibility required for effective resource sharing.

4.4.2. Airtime estimation and retransmission handling

A main consideration concerns airtime estimation. In most existing works, the airtime consumed by ADWRR-based scheduling policies is estimated according to Eq. (5). It models the time required to deliver a unicast frame over the air interface and to receive the corresponding acknowledgment, accounting all the required retransmissions:

$$Airtime = \overline{n_{Tx}} \cdot \left(\overline{T_{Backoff}} + T_{DIFS} + T_{DATA} + T_{SIFS} + T_{ACK} \right) \tag{5}$$

Let $p_{ACK}(R)$ denote the probability of successful ACK reception when packets are transmitted at physical rate of *R* (associated to a specific MCS). Under this typical formulation, the mean number of transmission ($\overline{n_{Tx}}$), including the original and the retransmission ($\overline{n_{retries}}$) —assuming as a simplified approach, an unlimited number of retransmissions —is approximated as $\overline{n_{Tx}} = 1/p_{ACK}(R)$.

The per-frame transmission overhead due to the CSMA/CA protocol is estimated as the sum of $T_{Backoff} + T_{DIFS} + T_{SIFS} + T_{ACK}$. This includes the average back-off time before transmission ($T_{Backoff}$, which is not always considered in all works), the Distributed Inter-Frame Space (T_{DIFS}), the Short Inter-Frame Space (T_{SIFS}) and the transmission time of the ACK control frame (T_{ACK}). In addition, when the RTS/CTS (Request to Send / Clear to Send) mechanism is active, airtime estimation must also account for the RTS/CTS control frame durations and the required additional inter-frame spacing.

Concerning the data frame transmission time (T_{DATA}), it is computed according to the used standard and antenna configuration. Eq. (6) shows the expression for OFDM based standards 802.11n and 802.11ac:

$$T_{DATA} = T_{PHY_{OH}} + T_{MPDU} = T_{PHY_{OH}} + T_{symbol}\left\lceil \frac{8\cdot(MAC_{OH}+L)}{T_{symbol}\cdot R(MCS)} \right\rceil \quad (6)$$

where T_{PHYOH} (μs) involves the physical overheads and T_{MPDU}. The time associated to MPDU transmission, T_{MPDU}, depends on how many OFDM symbols are needed to transmit the total bits of MPDU at the given physical rate. It depends on the MAC overheads, MAC_{OH}, and on the length of the data received from the LLC level ($L = l_{LLCOH} + l_{data}$, where l_{data} is the IP packet).

The data frame transmission time (T_{DATA}) is inversely proportional to the transmission rate R. However, in most prior works, the R and $p_{ACK}(R)$ are estimated based on the rate that provides the highest effective throughput (R_{best}), as inferred from statistics collected by the rate adaptation module. Critical implementation details are often omitted. These include the number of required transmission to be averaged, the size of the observation window, and the accuracy and sensitivity of R estimates to network dynamics (e.g., bursty or fluctuating traffic patterns, varying channel conditions, collisions, and interference). These factors are fundamental, and neglecting them can significantly compromise the accuracy of airtime estimation and, consequently, the depth of performance analysis and the outcomes obtained when applying the proposed approach.

In this work, rather than relying on statistical airtime estimations, airtime for each scheduled packet is computed based on the actual transmission time, which depends on the MCS/rate index. This approach requires precise tracking of packet transmissions to ensure accurate airtime accounting. Airtime is divided into two components and subtracted from the deficit counters in two phases:

1) When a packet is scheduled and selected for transmission, retransmissions are not yet known. Therefore, the airtime estimated at this stage—computed in lines 12 and 15 of Algorithm 2— accounts only for the initial transmission and is given by (7):

$$Airtime = \overline{T_{Backoff}} + T_{DIFS} + T_{DATA} + T_{SIFS} + T_{ACK} \quad (7)$$

2) Once the packet is transmitted, the airtime consumed by retransmissions can be estimated using the retry count. In scenarios where direct feedback from network driver is not accessible or supported, the auxiliary interface allows counting the number of retransmissions for each packet/frame, including details about the rates used in the respective retransmissions (for instance, if the same physical rate is used or a rate back-off algorithm is applied). If retransmissions occur at the same rate as the original packet, the additional airtime can be approximated as $n_{retries}$ times the airtime from Eq. (7).

In practice, airtime accounting is performed in real time. Upon detection of a retransmission, the corresponding extra airtime is accumulated in the variables $Deficit_{retries}$ *[s]* and $Deficit_{retries}$ *[s,i]*, according to the slice and queue the packet belongs to. The monitoring delay corresponds to the time interval between the scheduling decision—when the packet is enqueued at the driver buffer—and the retransmission event, including driver buffering and transmission delays. However, deficit correction is applied during the next scheduling visit to the corresponding slice and queue. That is, upon visiting the slice, $Deficit_{retries}$ *[s] and* $Deficit_{retries}$ *[s,i]* are subtracted from *Deficit [s]* and *Deficit [s,i]*, respectively (line 5 in Algorithm 1 and line 4 in Algorithm 2) and subsequently reset to continue the accumulation process. As a result, a single correction may account for the airtime accumulation of multiple retransmissions.

The duration of a scheduling round depends on the cumulative airtime of the packets scheduled from the other active queues. As long as the driver's de-buffering delay exceeds the duration of a round, a greater misalignment may occur between scheduler decisions and the application of corrections. Nevertheless, such misalignment can be bounded to a few milliseconds by a regulation process imposed in the scheduling operation, which limits the driver buffer occupancy through a predefined threshold. Under this condition, the impact of the monitoring delay on the deficit correction process is very limited. This ensures that the system quickly adapts to the required corrections and that both slices and queues receive their corresponding shares of resources, even over very short monitoring windows.

4.4.3. Driver buffer regulation

The scheduling process is explicitly regulated to control packet injection into the driver. Specifically, packets are scheduled only while the occupancy of one shared driver queue remains below a predefined threshold (lines 2 in Algorithm 1 and 11 in Algorithm 2); otherwise, the scheduler temporarily halts. There are two main reasons for limiting the driver buffer occupancy.

The first is related to slicing enforcement and aims to maintain slice isolation and performance guarantees. Limiting the buffer prevents packet drops caused by driver queue overflow—a well-known issue in tandem queue systems. In addition, it ensures that the delay between a packet transmission (including possible retransmissions) and the corresponding correction of slice or queue deficits remains short. Such timely corrections are essential to preserve fairness and isolation at short time scales.

The second is transitional and arises from the fact that rate control is implemented in user space at the AP agent. By minimizing the number of packets buffered in the driver, we reduce the likelihood that packets are transmitted at an outdated rate following a rate adaptation decision. This limitation would naturally disappear if rate control is implemented directly within the driver.

The primary objective of this regulation is therefore to keep the number of packets queued at the driver as low as possible—ideally limited to the packet currently under transmission and one additional packet. However, the exact threshold value may vary depending on how feedback about the driver queue occupancy is obtained. As explained in Section 3, when direct driver feedback is unavailable or unsupported, the auxiliary interface provides an alternative monitoring mechanism. By comparing the number of data frames transmitted over the main interface with those received at the auxiliary interface and by tracking their sequence numbers, it is possible to infer driver buffer occupancy and detect losses or retransmissions.

In our implementation, the threshold is conservatively set to 10 packets. This value accounts for the possibility that very short packet bursts may not be detected by the auxiliary monitoring interface, ensuring that such events do not compromise the regulation process. In addition, it is short enough to keep the induced delay bounded to a few milliseconds. Overall, without of direct driver feedback, this approach provides a practical and effective solution for enforcing reliable slicing under real-world constraints.

4.5. Frame aggregation considerations

Inter- and intra-slice scheduling explicitly considers the impact of frame aggregation. The 802.11n and 802.11ac standards allow two types of aggregation: Aggregated MAC Protocol Data Unit (A-MPDU) and Aggregated MAC Service Data Unit (A-MSDU). In A-MPDU aggregation, each MSDU is encapsulated into an individual MPDU by adding a MAC header and a Frame Check Sequence (FCS), and multiple MPDUs are then aggregated into a single A-MPDU with an additional PHY header. In contrast, A-MSDU aggregation combines multiple MSDUs into a single MPDU, sharing a single MAC header and FCS, followed by the corresponding PHY header.

Although both options could be supported, to simplify the model we focus exclusively on A-MSDU aggregation. Assuming a frame that aggregates N IP packets, each one of size l_n bytes ($n = 1..N$), the size of the aggregated A-MSDU (L), which can be used in Eq. (8), becomes:

$$L = \sum_{n=0}^{N} \left(l_{MSDUheader} + l_{LLC_{OH}} + l_n + l_{Padding} \right) \quad (8)$$

where $l_{MSDUheader}$ (14 bytes) includes the destination address (DA), sender address (SA) and length of the MSDU received from LLC. Each MSDU is composed by the 8 bytes of LLC (l_{LLCOH}) and the received IP packet ($l_{n.}$). $l_{Padding}$ represents the required 0–3 bytes of padding.

Frame aggregation is an optional feature that can be applied selectively and independently, depending on the class of service or even the capabilities of the STA. Accordingly, in Algorithm 2, two different options are distinguished. When aggregation is enabled, the maximum length of the aggregated A-MSDU that can be transmitted is determined based on the available *deficit (D [s,i])* (line 14 in Algorithm 2).

This value, further constrained by the maximum allowable aggregation size, is used to aggregate as many packets as possible, provided they all belong to the same STA as the packet at the head of the queue. The airtime of the resulting aggregated frame is then computed (line 15 in Algorithm 2). Different packet selection policies for aggregation may be implemented. In this work, a simple policy is adopted. Since a queue may store packets from data flows belonging to different STAs, it is allowed to search within the buffer for packets from the same STA to complete the aggregation up to the allowed size.

4.6. Quantum values and design guidelines

Regarding the quantum values and their order of magnitude, in this case, they may range from tens of microseconds (μs) to tens of milliseconds (ms). When frame aggregation is not enabled, since the quantum is accumulated in the deficit counter, the differences between values such as 50 μs or 5 ms are not particularly significant. Nevertheless, smaller quantum values may provide higher granularity and fairness in resource allocation, as well as lower jitter. Conversely, when aggregation is enabled, the quantum must be large enough to accommodate the airtime of aggregated frames. Therefore, the specific quantum range ultimately depends on the system's scheduling granularity and, eventually, on traffic characteristics, and quality of service requirements.

5. Experimental testbed & results

This section evaluates and demonstrate the effectiveness of the proposed hybrid ISS-QoSS slicing at the AP level.

5.1. Testbed setup

The evaluation was conducted on the experimental testbed corresponding to the data plane illustrated in Fig. 2, within the overall architecture shown in Fig. 1. While the proposed system is designed to support multiple APs, the experimental analysis focuses on a single AP, as depicted in Fig. 3. This is sufficient to characterize the local slicing behavior, since slicing enforcement and scheduling decisions are performed independently at each AP under the considered deployment assumptions. As discussed in Section 3, the obtained results are representative and generalizable to multi-AP scenarios.

Experiments were carried out with real hardware in the 5 GHz band using IEEE 802.11n and 802.11ac. The equipment used for the tests comprises the AP, Ethernet switches supporting both data and management planes, a traffic generator and IP flow classifier, and a set of STAs.

AP runs on a mini PC (*Asus NUC RNUC13ANKI5, i5–1340P*) with *Ubuntu 22* (*kernel 5.19.9*) and uses two USB Wi-Fi cards (*Alfa AWUS036ACH* and/or *AWUS036ACM*, based on *RTL8812au* and *MT7612u* chipsets). These cards (main and auxiliary interfaces) support frame injection in monitor mode. Their antennas are physically separated by a reflector to reduce channel correlation and prevent receiver blocking, which is relevant for MIMO operation and with closely spaced radios.

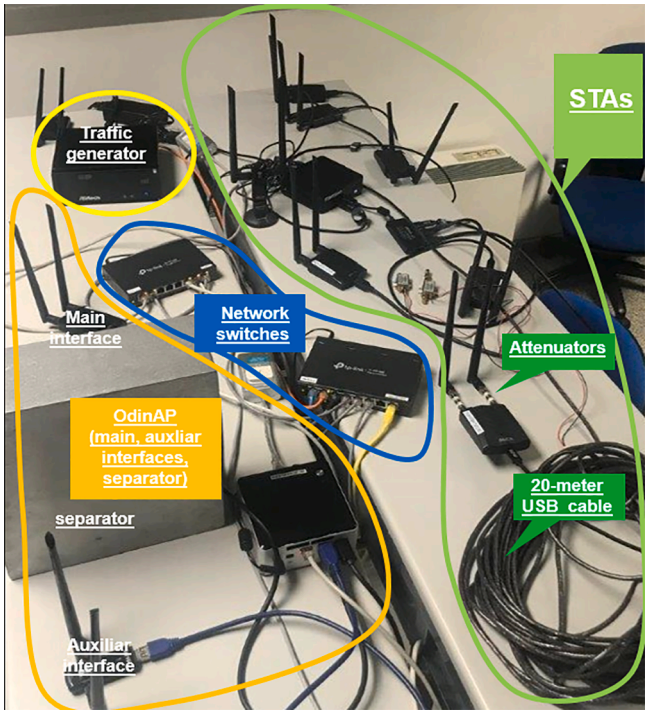


Fig. 3. Equipment used for the tests.

STAs are implemented using the same USB Wi-Fi cards connected to a mini PC (*Asus NUC RNUC13ANKI5, i5–1340P, Ubuntu 22, kernel 5.19.9*). These cards operate in active monitor mode and are capable of generating ACK frames. To emulate heterogeneous channel conditions and mobility, some STAs are connected via 20-m USB cables and may include fixed or programmable attenuators.

Downlink traffic is generated by a dedicated *mini PC* (*ASRock Core 100HT*) with *Debian* (*kernel 4.19.28*) using the *iperf* tool and manipulating the ARP cache to include the MAC addresses of the STAs. Additionally, the *iptables* tool is used as a traffic classifier by mapping the different slices and queues used in the DSCP field of the outgoing datagrams.

5.2. Evaluation conditions

Experiments are conducted with controlled traffic and mobility conditions while seeking minimal (though non-zero) external interference, to ensure reproducibility and enable analysis of the slicing mechanism under well-defined operating conditions. While more dynamic and fully realistic scenarios could be considered, enabling mobility and rate adaptation across all devices would significantly hinder repeatability and limit the ability to create specific conditions for validating the expected behaviour of the scheme. Reducing external interference does not compromise the validity of the results, as only the airtime accessible to the AP is considered allocable. Airtime consumed by uplink or sensed external transmissions merely reduces the available downlink airtime, affecting throughput, but not the relative airtime distribution among slices. Moreover, interference originating from hidden nodes manifests itself through retransmissions. Since the environment is not fully isolated and external WiFi activity may still occur, its impact is naturally reflected in the results.

The system operates in 802.11n/HT supporting MCS 0 to 15 with bandwidth of 20 MHz. The proposed slicing mechanism is independent of the selected PHY configuration, as variations in bandwidth or MCS do not affect its main objective. Moreover, higher PHY rates do not necessarily result in proportional gains in effective throughput due to the impact of MAC and PHY overheads, particularly at high rates.

Table 2
List of input parameters used during the experimentation.

Parameters	Values							
Slices	**Slice 0**			**Slice 1**		**Slice 2**		
Quantum (Q_{nom})	3500µs			2500µs		4000µs		
Queues	**Queue 0**	**Queue 1**		**Queue 0**	**Queue 1**	**Queue 0**	**Queue 1**	**Queue 2**
Weight (W_{nom})	50	50		30	70	50	30	20
STA id	STA 0	STA 1	STA 2	STA 3	STA 4	STA 5	STA 6	STA 7
MCS index	3	6	1	Variable.	1	2	4	6
Iperf Rate (Time interval)	1.4 Mbps (0-50 s)	1.4Mbps (0-50 s)	1.4Mbps (0-50 s)	1.4Mbps (0-50 s)	1.4Mbps (0-50 s)	1.4 Mbps (0-10s), 0.5 Mbps (10-20 s), 1.4 Mbps (20-30 s), 0.5 Mbps (30-50 s)	1.4 Mbps (0-30s) 0.5 Mbps (30-50 s)	1.4 Mbps (0-30s) 0.5 Mbps (30-50 s)
UDP size (bytes)	250B	1250 B	650 B	500 B	250 B	250 B	250 B	400 B
Airtime (µs)	281.5µs	345.5µs	625.5µs	-	377.5µs	313.5µs	249.5µs	241.5µs
Airtime required parameters	Band = 5GHz, BW = 20MHz, $T_{DIFS} = 34\mu s$, $T_{SIFS} = 16\mu s$., $T_{Backoff} = T_{slot} \cdot CW/2$ with CW = CWmin = 15 , T_{phyOH} (11a)=20µs, T_{ACK} (24Mbps) =28µs, $T_{slot} = 9\mu s$, T_{phyOH} (mixed) = 36µs + 4µs ($n_{antenna} - 1$) , T_{phyOH} (greenfield) = 28µs + 4µs ($n_{antenna} - 1$) $MAC_{OH} = L_{service}$ (2B) + L_{HMAC} (24B) + L_{QoS} (2B) + L_{FCS} (4B), L_{LLCOH} (8B), L_{data} (xB) = IP (20B) + UDP (8B) + DATA							

Downlink traffic consist of eight UDP flows destined to eight STAs, following the workload parameters in Table 2. Flows are distributed across three slices (0–2) and up to three queues/SCs per slice, with nominal quantum (Q_{nom}) and weight (W_{nom}) values detailed in Table 2.

Traffic parameters are designed to cover diverse operational conditions while keeping data visualization and analysis manageable. Each STA typically uses a single service class and slice, except STA 1 and STA 2, which share SC (queue 1) in slice 0 to evaluate performance under shared queue usage. Packets are generated with a uniform distribution but with varying parameters during the experiment. Table 2 details the *iperf* rates set over 50 s for each flow. Note that, even with identical *iperf* rates, differences in packet sizes lead to varying packet rate demands, resulting in distinct traffic patterns. Traffic parameters are specifically selected to include periods with load-saturated conditions, which are necessary to verify strict airtime isolation when slices or service classes exceed their guaranteed demand. Once isolation is achieved, the objective is to observe how any surplus resources are redistributed according to the expected AWDRR policy, and to evaluate the system's real-time adaptation to dynamic changes in traffic and channel conditions.

As illustrated in Fig. 3, most STAs (0, 1, 4 to 7) are located within a 2-meter radius of the AP, ensuring stable and good RSSI conditions. In order to emulate different propagation conditions, which results in different selected transmission rates by rate control mechanism, different but fixed MCSs are selected for all the STAs except STA3. That is, really, rate control (ONOE) is only active in STA 3. This assumption ensures that airtime requirements differ among flows. Estimated airtimes (excluding retries) and parameters used in the airtime estimation are summarized in Table 2. Note that, in this work, mean $T_{Backoff}$ is estimated using the minimum Contention Window (*CW*), assuming a no-collision case as the most common operating scenario. Improvements in airtime estimation considering the on-time *CW* are possible. Finally, to clearly observe the impact of retransmissions and mobility, STA 1 and STA 3 are connected via a 20-meter USB cable and positioned outside the laboratory. While STA 1 remains stationary and uses a fixed MCS 6, STA 3 is in motion with the rate control activated, introducing variability to the experimental conditions. This setup allows, without loss of generality, the evaluation of the performance of the hierarchical local slice system and its ability to efficiently handle variations in airtime demands linked to retransmissions and rate control.

5.3. Results

The following discussion focuses on the first 50 s of the experiment. To assess the ability of the proposed hierarchical scheduler to enforce isolation while redistributing unused resources in a controlled manner, we analyze three complementary, interrelated metrics: airtime share, effective frame transmission rate (successfully transmitted packets), and throughput. All metrics are averaged over 200 ms intervals.

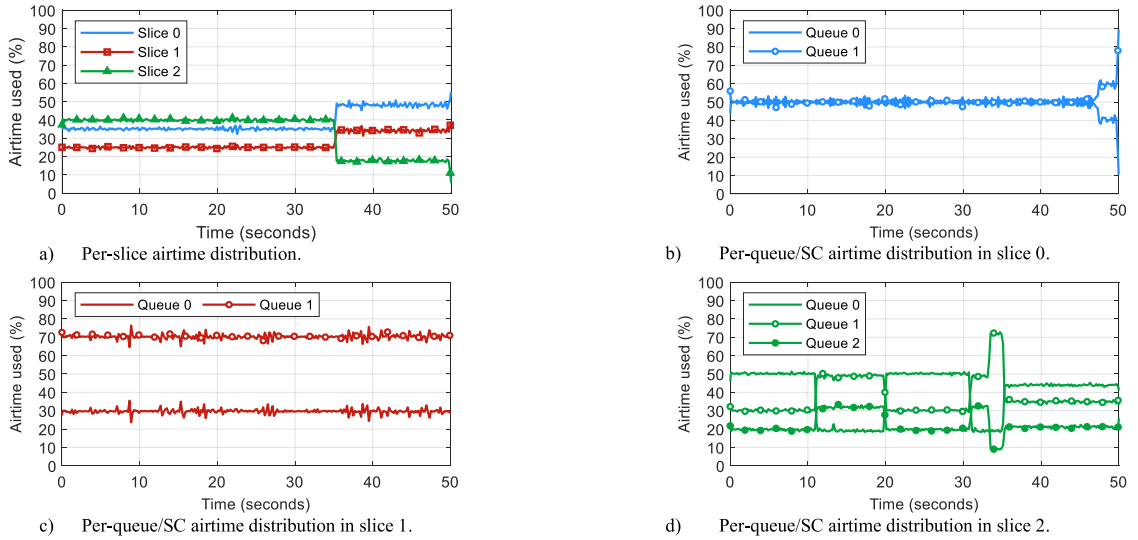
Fig. 4 presents the airtime distribution across slices (Fig. 4a) and across service classes within each slice (Fig. 4b–d). Figs. 5 and 6 show the effective frame transmission rate and UDP throughput, respectively. Frame aggregation is disabled in this section, so each data frame corresponds to a single UDP packet. Retransmissions affecting the flows are shown in Fig. 7, while Fig. 8 illustrates the rate variations experienced by STA 3.

During the initial congestion period ($t = 0$–10 s), Fig. 4a shows that airtime is distributed among slices exactly in accordance with the guaranteed quantum percentages (35 %, 25 %, and 40 %, respectively). Similarly, intra-slice airtime allocation follows the configured nominal weights: slice 0 exhibits a 50 %/50 % split (Fig. 4b), slice 1 a 30 %/70 % split (Fig. 4c), and slice 2 a 50 %/30 %/20 % split (Fig. 4d). This confirms that the hierarchical scheduler enforces strict isolation when traffic demand exceeds available airtime, resulting in full partitioning of resources.

Under these congested conditions, packet transmission rates (Fig. 5) and throughput (Fig. 6) remain below the offered ones, reflecting the airtime limitations. For example, in slice 0, queue 0 achieves 650 p/s against a demand of 700 p/s, while queue 1 reaches only about 145 p/s, significantly below its demand of 410 p/s (140 p/s + 270 p/s for STA1 and STA2) due to a high number of retransmissions affecting STA 1 (Fig. 7). Similar effects are observed in slices 1 and 2. A noticeable event occurs at $t \approx 6.1$ s, where transmission rates drop simultaneously across all slices despite stable airtime allocation. This behavior is attributed to external interference, which temporarily reduces the effective usable airtime at the AP.

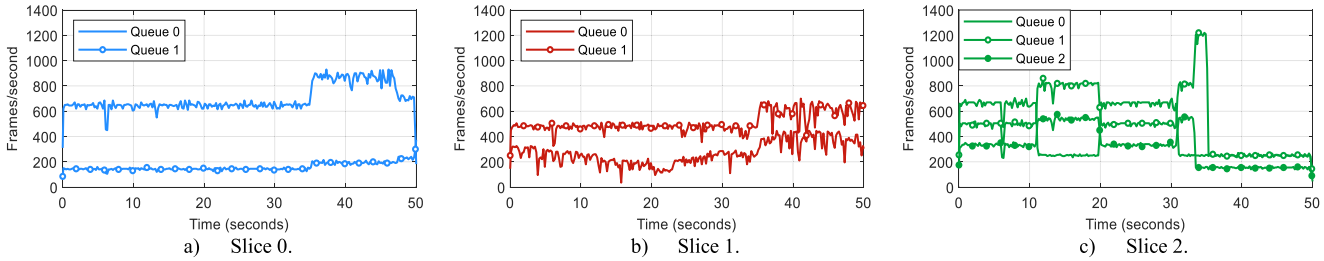
It is worth noting that under congestion conditions, throughput and frame-rate degradation depend on the mismatch between airtime demand and the allocation assigned to each slice and service class. This behavior differs fundamentally from a non-sliced setup, where throughput degradation is proportional to traffic demand, yielding identical satisfaction ratios—defined as the ratio between achieved throughput and demand—across all STAs, regardless of retransmissions. In that case, airtime consumption simply follows transmitted traffic, with no enforced constraints.

Slice 2 provides a clear illustration of controlled airtime redistribution. Between $t = 10$ s and $t = 20$ s, the demand of queue 0 drops to 250 p/s (0.5 Mb/s). As shown in Fig. 4d, queue 0 becomes satisfied and consumes only 19 % of the slice airtime, leaving 31 % unused. This excess airtime is redistributed to queues 1 and 2 proportionally to their



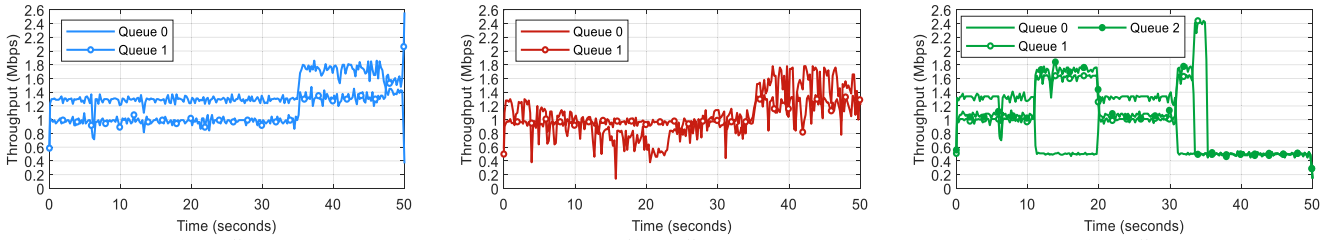
a) Per-slice airtime distribution.
b) Per-queue/SC airtime distribution in slice 0.
c) Per-queue/SC airtime distribution in slice 1.
d) Per-queue/SC airtime distribution in slice 2.

Fig. 4. Airtime distribution among slices and service class (time average: 200 ms).



a) Slice 0. b) Slice 1. c) Slice 2.

Fig. 5. Correct frame/packet transmission rates for queues/SCs in slices 0, 1, and 2 (time average: 200 ms).



a) Slice 0. b) Slice 1. c) Slice 2.

Fig. 6. Effective throughput for queues/SCs in slices 0, 1, and 2 (time average: 200 ms).

weights (60 % and 40 %, respectively), resulting in approximately 48–49 % airtime for queue 1 (30 % + 0.6 × 31 %) and 32–33 % for queue 2 (20 % + 0.4 × 31 %), while preserving the slice's total airtime share. Redistribution takes effect at $t = 10.9$ s, once buffered packets are drained, and the guaranteed airtime of queue 0 is immediately restored when its demand increases again between $t = 20$ s and $t = 30$ s.

From $t = 30$ s onward, the demand of all queues in slice 2 drops to 0.5 Mb/s. After a short transient needed to empty buffered packets, the queues become satisfied sequentially: queue 0 at $t = 30.9$ s, queue 2 at $t = 33.1$ s, and queue 1 at $t = 35.1$ s. During this process, the unused airtime is progressively redistributed among the unsatisfied queues according to the configured weights. By $t = 35.1$ s, slice 2 no longer requires its full allocation, and its unused airtime (22 %) is redistributed between slices 0 and 1 in proportion to their nominal quantum values

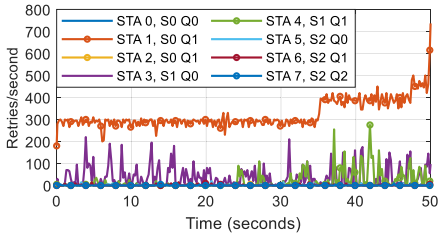


Fig. 7. Frame retries/s associated with data flows (STAs) for queues/ SC (Q0, Q1, Q2) of slices 0 (S0), 1 (S1), and 2 (S2) (time average: 200 ms).

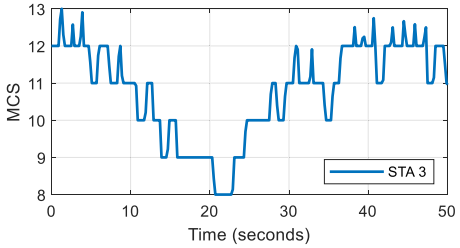


Fig. 8. MCS variations of STA 3.

(58.3 % =35/(35+25) and 41.6 % = 25/(35+25), respectively). This behavior is reflected in Fig. 4a, where slice 0′s airtime increases to approximately 48 % and slice 1′s to 34 %. Within slice 0 (Fig. 4b), airtime continues to be shared among queues according to their weights until $t = 46.7$ s. At this point, the unused airtime of queue 0—once its required rate is satisfied (Fig. 5a)—is redistributed to queue 1.

Effective throughput results over the 50 s interval (Fig. 6) are consistent with the previously referred metrics and input parameters. However, while throughput provides a relevant metric from the application perspective, it does not reflect whether the hierarchical slicing objectives—such as isolation and fair resource sharing—are met. Therefore, airtime-based metrics remain the primary and most representative measure for evaluating slice performance and fairness. A key observation from the results is that the proposed scheme accurately estimates airtime consumption for each service class, even under rate variability and significant number of retries. For example, queue 0 of slice 1 undergoes MCS changes (Fig. 8) and experiences a significant percentage of transmission errors, as seen in STA 3 in Fig. 7. That results in variable transmission times, leading to a variable number of transmitted frames/s (Fig. 5b) and variable effective throughput (Fig. 6b). Despite these variations, airtime isolation between service classes is consistently maintained.

The accuracy of airtime consumption estimation is also verified in queue 1 of slice 0, which aggregates two flows destined to STA 1 and STA 2. These flows differ not only in airtime requirements—due to distinct packet sizes and MCS values—but also in channel conditions. In particular, the flow toward STA 1 experiences frequent retransmissions (Fig. 7). Despite this heterogeneity, airtime accounting for queue 1 remains accurate, retransmission corrections are effective, and isolation with respect to other queues and slices is always ensured. This confirms that heterogeneous flows sharing a queue do not compromise the scheduler's isolation guarantees.

To explicitly highlight the importance of retransmission-aware airtime accounting, Fig. 9 presents results obtained from a run of the experiment under similar mobility and channel conditions (i.e. retries), but without incorporating the airtime correction due to retransmissions in the algorithm. In this case, airtime consumptions for queue 1 of slice 0 and queue 0 of slice 1 are underestimated, leading to excess airtime allocations and visible imbalances in both inter- and intra-slice distributions. For instance, slice 0 receives 48 % of resources instead of its nominal 35 %. This misallocation degrades the resources available to slices 1 and 2 and, consequently, reduces the packet transmission rate and throughput of the queues associated with these slices, as observed in Fig. 9b and c when compared with Figs. 5 and 6. Within slice 0, queue 1 absorbs more resources than allocated (70 % instead of 50 % in Fig. 9d), resulting in a degradation of queue 0 performance. In slice 1, where retransmissions are present, but not severe, the lack of airtime estimation leads to a temporary reduction in fairness in the intra-slice airtime distribution (Fig. 9e). By contrast, in slice 2—where errors and retransmissions are minimal—intra-slice airtime distribution remains largely unaffected (Fig. 9f).

This highlights the critical the importance of accurately accounting for retransmissions to ensure fair resource management between slices and effective performance within the experimental framework.

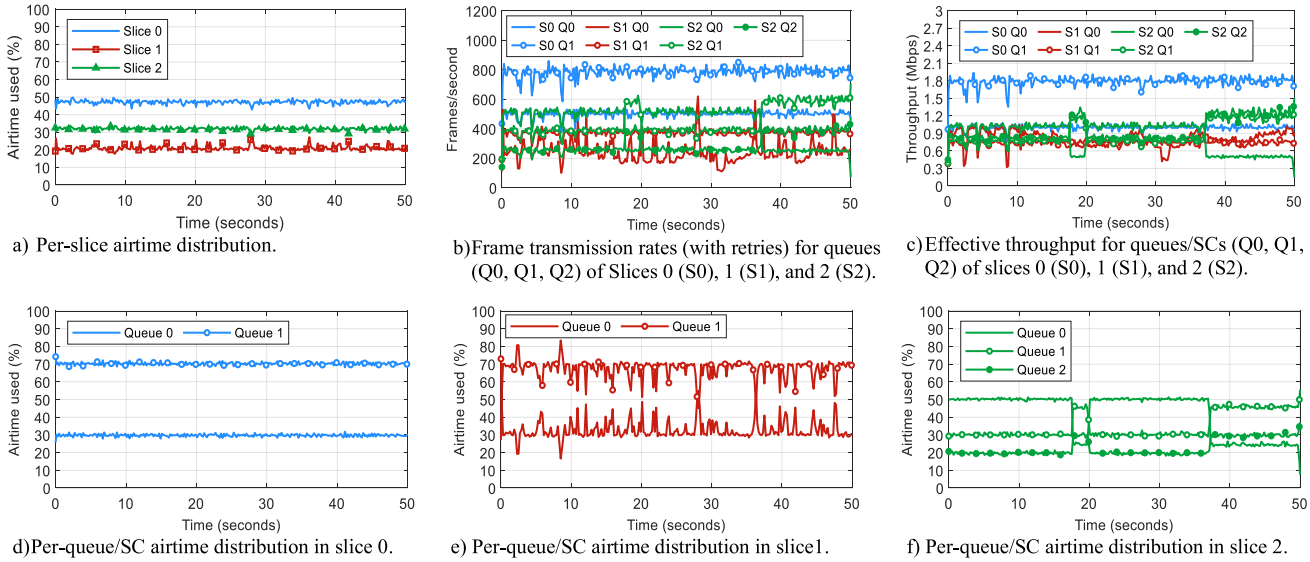
The impact of the quantum order of magnitude is analyzed. Fig. 10 shows per-slice airtime distribution for different quantum values. Small quantum values (below 1 ms) yield behavior comparable to that observed with quantum values up to 10 ms, with similar performance across all metrics. In contrast, larger quantum values (up to 100 ms) still ensure correct long-term airtime distribution, as shown by 1 s averages (Fig. 10c), but significantly reduce granularity and short-term fairness, as evidenced by 200 ms averages (Fig. 10b). Therefore, when frame aggregation is disabled, smaller quantum values are preferable due to their finer granularity and improved responsiveness.

Finally, Figs. 11 and 12 extend the analysis to a less controlled scenario with full mobility and consequent MCS variability across STAs, comparing the proposed hierarchical method with a baseline no-slicing implementation. In the latter, packets are marked with DSCP for comparison purposes only but served using a FIFO policy.

The setup is analogous to that in Table 2, but without controlled traffic-demand variations, as full mobility prevents isolating specific effects. Instead, a simplified saturated regime is imposed to clearly expose slicing effects. All flows have identical, slightly higher iPerf rates (1.7 Mbps), ensuring saturation even with higher MCSs, and different packet sizes, as detailed in Table 2.

Metrics include airtime share at the first slicing level (a), airtime share among SCs at the second level (d), MCS evolution per STA (c), effective throughput (excluding erroneous frames) at the first level (b), and effective throughput at the SC level (e). It is important to emphasize that throughput is not the primary optimization target; rather, it serves as an indirect indicator, while the fundamental objective is to guarantee controlled airtime allocation and strict isolation between slices.

As observed in the MCS evolution (Figs. 11 and 12c), even with similar STA movements along corridors and rooms, each experiment



a) Per-slice airtime distribution.

b)Frame transmission rates (with retries) for queues (Q0, Q1, Q2) of Slices 0 (S0), 1 (S1), and 2 (S2).

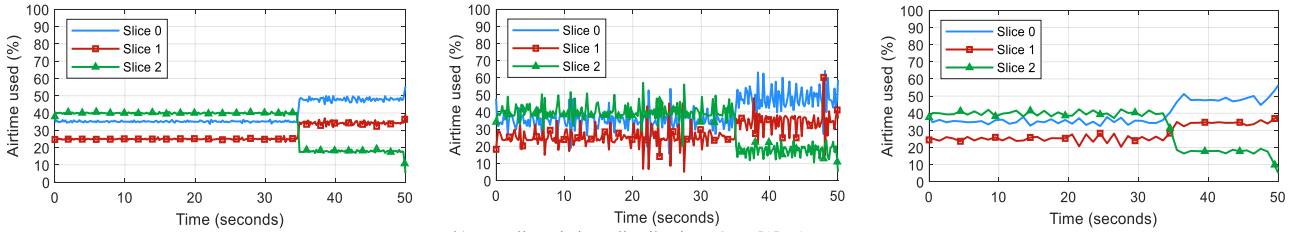
c)Effective throughput for queues/SCs (Q0, Q1, Q2) of slices 0 (S0), 1 (S1), and 2 (S2).

d)Per-queue/SC airtime distribution in slice 0.

e) Per-queue/SC airtime distribution in slice1.

f) Per-queue/SC airtime distribution in slice 2.

Fig. 9. Results obtained for scenario in Table 2 when airtime corrections related to retries are not applied.



a) Per-slice airtime distribution. $Q_{nom}[0]$ =350µs, Q_{nom} [1]=250µs, Q_{nom} [2]=400µs (time average: 200ms).

b) Per-slice airtime distribution. Q_{nom} [0] =35ms, Q_{nom} [1]=25ms, Q_{nom} [2]=40ms (time average: 200ms).

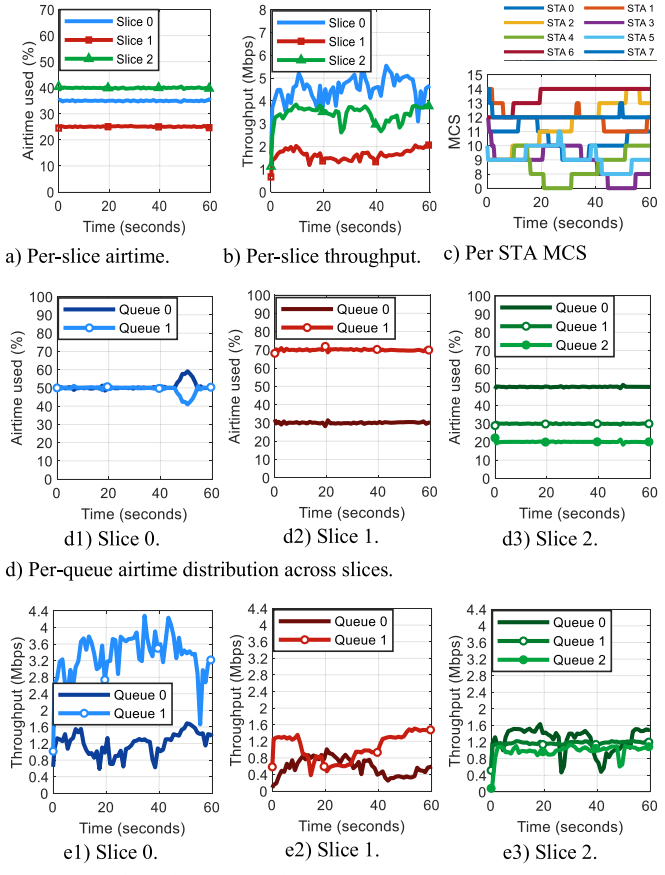
c) Per-slice airtime distribution. Q_{nom} [0] =35ms, Q_{nom} [1]=25ms, Q_{nom} [2]=40ms (time average: 1s).

Fig. 10. Per-slice airtime distribution for scenario in Table 2 depending of the order of magnitude of the quantum.

differs due to channel variability and transmission errors, naturally leading to MCS adjustments according to the rate control algorithm.

Consistent with previous observations, only the proposed hierarchical slicing method guarantees airtime allocation and isolation at both levels, whereas the no-slicing baseline provides no controlled resource sharing. Minor effects, such as the peak observed in slice 0 (e.g., graph Fig. 11d1), arise from the redistribution of unused resources from other queues.

Without slicing, as theoretically expected, effective throughput is proportional to traffic demand, yielding identical satisfaction ratios (throughput over demand) across all STAs and resulting in overlapping curves for flows with identical rates (Fig. 12b, e2, e3). In contrast, with slicing, throughput is primarily governed by allocated airtime. For example, slice 0 achieves higher throughput than slice 2 despite a



a) Per-slice airtime. b) Per-slice throughput. c) Per STA MCS

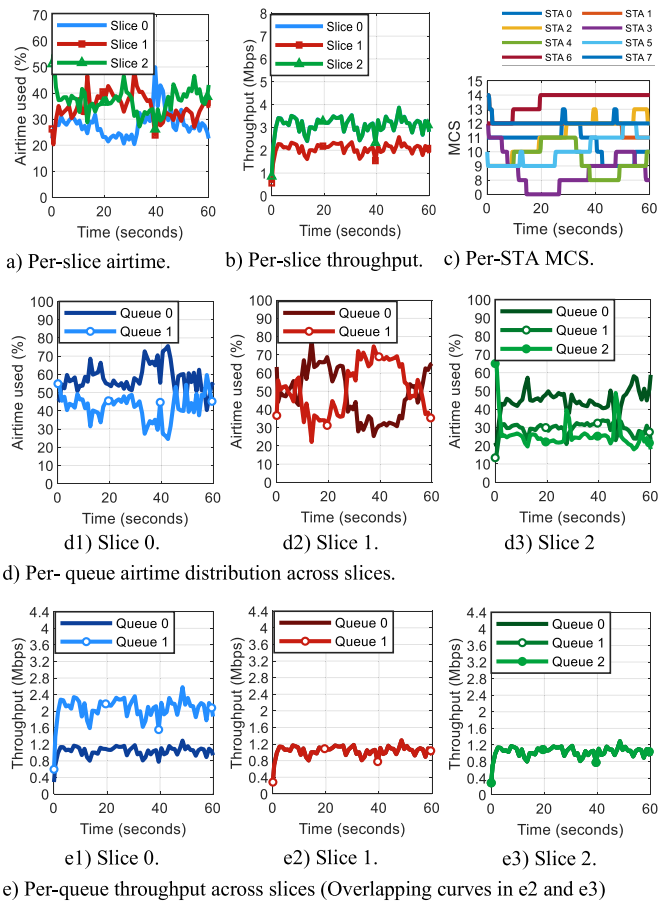
d1) Slice 0. d2) Slice 1. d3) Slice 2.

d) Per-queue airtime distribution across slices.

e1) Slice 0. e2) Slice 1. e3) Slice 2.

e) Per-queue throughput across slices.

Fig. 11. Results of applying hierarchical slicing under the conditions defined in Table 2, assuming full STA mobility and an iPerf rate of 1.7 Mbps per STA (time average: 1 s).



a) Per-slice airtime. b) Per-slice throughput. c) Per-STA MCS.

d1) Slice 0. d2) Slice 1. d3) Slice 2

d) Per- queue airtime distribution across slices.

e1) Slice 0. e2) Slice 1. e3) Slice 2.

e) Per-queue throughput across slices (Overlapping curves in e2 and e3)

Fig. 12. Results without applying hierarchical slicing under the conditions in Table 2, assuming full STA mobility and an iPerf rate of 1.7 Mbps per STA (time average: 1 s).

smaller quantum (3500 µs vs. 4000 µs), due to more efficient airtime utilization with larger packet sizes (Fig. 11b).

Overall, the experimental results confirm that the proposed hierarchical scheduling mechanism enforces strict airtime isolation while enabling efficient and proportional redistribution of unused resources. Depending on traffic intensity, the system naturally alternates between periods of full partitioning and periods of dynamic sharing, without violating isolation guarantees or fairness objectives.

6. Dynamic reconfiguration proposal

As discussed in Section 2, numerous algorithms have been proposed for quantum adaptation, particularly in QoS slicing, to satisfy flow-specific QoS requirements through resource reallocation among slices. These approaches include heuristic, stochastic optimization, and AI/ML-

based methods. In this work, the proposed approach distinguishes itself from prior work in several key aspects:

- *Preservation of nominal airtime shares:* The proposal does not modify the airtime portions guaranteed for each slice or for each service class (queue) within a slice. That is, it does not alter the nominal quantum values (Q_{nom} *[s]*) or the nominal weights (W_{nom} *[s,i]*) configured by the centralized slicing component at the controller. Note that the centralized slicing component may still perform adaptations to re-orchestrate resource allocations among slices across the set of APs, as well as within individual APs, in accordance with tenant and QoS requirements. However, such adaptations are confined to that level of operation. Local scheduling at the AP strictly adheres to the resulting adjusted parameters.
- *Local short-term adaptation to meet specific underutilized resource redistribution criteria:* Our contribution focuses on short-term, local adaptations of the ADWRR-based scheduler to handle transient traffic fluctuations while preserving the guaranteed airtime portions defined by nominal values. This mechanism is relevant only when the AP channel is saturated, but some slices or service classes (SCs) require less than their guaranteed share. As shown in Sections 4 and 5, ADWRR naturally redistributes unused airtime from underutilized entities to those with higher demand, proportionally to their quantum. By locally adjusting quantum or weight values, this redistribution can be fine-tuned to achieve alternative objectives, such as prioritizing high-importance traffic (e.g., premium users) or equalizing SLA-defined satisfaction metrics, including rate satisfaction (achieved-to-requested rate), latency requirements, and other performance targets. We emphasize that quantum/weight adaptation is not required or has no effect when AP resources exceed demand.
- *Constant-sum quantum redistribution approach:* Unlike prior work, the proposed approach neither applies independent step-based adjustments to each slice or service class (SC) [11,12,17] nor computes quantum values through explicit optimization [7]. Instead, it operates under a quantum (or weight) redistribution framework, in which the total quantum (or total weight in intra-slice scheduling) is kept constant. Any increase in the quantum of a given slice—which corresponds to a proportional increase in its allocated airtime—is balanced by a corresponding decrease in others, ensuring controlled guarantees. This framework can be extended to other levels of operation, such as the centralized slicing component. It relies on periodically checking whether resource requirements are met. Statistics are collected locally during time intervals of *T* seconds, which also serve as the update intervals. The duration of these intervals is configurable, with a value of 1 s considered in this study. This interval length provides sufficient data for accurate adjustments while maintaining adaptability to dynamic traffic and performance objectives.

Specifically, in this work, the quantum/weight distribution scheme can be applied intra-slice (among service/application classes), inter-slice (among tenants), or both, according to desired granularity and allocation objectives.

6.1. Intra-slice weight redistribution

As an illustrative case, the high-level steps of the dynamic reconfiguration framework for intra-slice weight redistribution are presented in Algorithm 3, while the detailed pseudocode of its main phases is given in Algorithm 4. As shown in Algorithm 3, the local adaptation scheme periodically (every *T* seconds) adjusts the weights *W [s,i]* used in Algorithms 1 and 2 around nominal values W_{nom} *[s,i]*, which are semi-statically provided by a controller slicing component over a timescale significantly longer than *T*.

Specifically, two alternative objectives of local redistribution are considered: a) ensuring equal levels of rate satisfaction (lines 26–46 in Algorithm 4) and b) prioritizing queues according to their priority level (lines 47–72 in Algorithm 4). The notation used is defined in Table 3. Since all variables refer to the same monitoring interval, only airtime-related variables explicitly include a time reference. Unlike in Algorithms 1 and 2, $airtime_T$ *[s,i]*, $airtime_T$ *[s]*, $airtime_T$, here denote, respectively, the total airtime consumed (including retries) during interval *T* by queue *i* of slice *s*, by slice *s*, and by all slices at the AP.

For both objectives, the SLA specifies a per-queue (SC) Maximum Bit Rate (MBR), defined as the maximum IP-layer data rate guaranteed by the service provider for the aggregate traffic associated with the queue. Accordingly, *MBR [s,i]* denotes the MBR of queue *i* (SC *i*) of slice *s*. The aggregated data rate demanded by queue *i*, $R_{demanded}$ *[s,i]*, may exceed the *MBR [s,i]*. In such cases, the local scheduler limits resource allocation to *MBR [s,i],* unless surplus resources are available and all other queues have already met their respective demands.

The overall objective is to maximize the channel utilization.

According to Algorithm 3, each execution of the redistribution process proceeds as follows.

In each interval *T*, statistics are collected by the auxiliary interface (step #1) for each queue *i* of slice *s*, including achieved rate $R_{achieved}$ *[s,i]* (at the IP level) and $R_{demanded}$ *[s,i]*.

Queues are then classified (step #2) into groups based on their degree of rate satisfaction, defined as *DS [s,i]* = $R_{achieved}$ *[s,i]*/$R_{demanded}$ *[s,i]*. If $R_{demanded}$ *[s,i]*>*MBR [s,i]*, it is bounded by *MBR [s,i]* yielding *DS [s,i]* =$R_{achieved}$ *[s,i]*/*MBR [s,i]*. Once all the queues are satisfied, *DS [s,i]* may be recomputed in step #3 using the original $R_{demanded}$ *[s,i]*.

A queue is classified as unsatisfied and included in the group $S_{unsatisfied}$ *[s]* if *DS [s,i]* < *1*. Conversely, a queue is classified as underutilized and included in the group $S_{underutilized}$ *[s]* if *DS [s,i]*=*1* and occupies less resources than its nominal airtime percentage, so that part of its resources can be safely ceded to other queues within the same slice. A guard margin α is defined to enable this resource release.

Once queues are classified, step #3 determines the intended weight redistributions which are then applied in step #4.

Algorithm 3
Dynamic reconfiguration framework.

Algorithm 3 Dynamic reconfiguration framework

#Initial configuration or adaptation from the controller
for each slice s and queue (s, i) **do**
  $Q[s] \leftarrow Q_{nom}[s]$ ▷ Nominal quantum for slice s
  $W[s, i] \leftarrow W_{nom}[s, i]$ ▷ Nominal weight for queue (s, i)
  $T \leftarrow update\_interval$ ▷ Monitoring interval (configurable)
end for

#Main loop
while local slicing is running (between controller adaptation periods) **do**
  #Run scheduling algorithm (inter-slice + intra-slice) (Computes $Q[s, i]$ based on current $Q[s]$ and $W[s, i]$)
  for each slice s every T seconds **do**

    #Step 1: Monitoring. Metric collection for each queue (s, i)
    $R_{archived}[s, i] \leftarrow measure\_transmitted\_rate(s, i)$
    $R_{demanded}[s, i] \leftarrow estimate\_desired\_rate(s, i)$

    #Step 2: Queue classification
    $DS[s, i] \leftarrow compute\_DS(R_{archived}, R_{demanded}, MBR[s, i])$
    $S_{unsatisfied}[s] \leftarrow \{i \mid DS[s, i] < 1\}$
    $S_{underutilized}[s] \leftarrow \{i \mid DS[s, i] = 1 \ \&\ can\_cede(W[s, i], \alpha)\}$

    #Step 3: Weight redistribution (intra-slice) from queues of $S_{underutilized}[s]$ to queues of $S_{unsatisfied}[s]$, based on similar similar rate satisfaction or priority satisfaction option

    #Step 4: Apply new weights and update per-queue quantums
    $W[s, i] \leftarrow W_{nom}[s, i] + W_{added}[s, i] - W_{ceded}[s, i]$
  end for
end while

Algorithm 4
Intra-slice redistribution of unused quantum.

Algorithm 4 Intra-Slice Redistribution of unused quantum

#Identify unsatisfied queues of slice *s*
▷ $DS[s,i]$: Degree of satisfaction of queue i of slice s
▷ $IDS^{exp}[s,i]$: The increase of *DS* expected for an increment of weight
for each queue $i \in$ slice s **do**
  $DS[s,i] = \max\left(\frac{R_{achieved}[s,i]}{R_{demanded}[s,i]}, \frac{R_{achieved}[s,i]}{MBR[s,i]}\right)$
  if $DS[s,i] < 1$ **then**
    Add i to $S_{unsatisfied}[s]$ group and keep pair $(W[s,i], DS[s,i])$
  end if
  $IDS^{exp}[s,i] = DS[s,i]/W[s,i]$
  $ds[s,i] = DS[s,i]$ ▷ Variable to keep the expected satisfied rate
end for

#Estimate *DS* if weight was the nominal
▷ $S_{ceded}[s,i]$: group of queues that have ceded resources to queue $i \in s$
▷ $W_{ceded}[s,i,j] = W_{added}[s,j,i]$: resources ceded by queue i to $j \in s$
▷ $W_{added}[s,i] = \sum_{\forall j \in S_{ceded}[s,i]} W_{ceded}[s,j,i]$
▷ $W[s,i] = W_{nom}[s,i]$ + $W_{added}[s,i]$ - $W_{ceded}[s,i]$
for each queue $i \in$ slice s whose $W[s,i] > W_{nom}[s,i]$ **do**
  for each queue $j \in S_{ceded}[s,i]$ **do**
    $W[s,j]$ += $W_{ceded}[s,j,i]$
    $ds[s,j] = \min(ds[s,j] + IDS^{exp}[s,j]*W_{ceded}[s,j,i], 1)$
    $W[s,i]$ -= $W_{added}[s,i,j]$
    $ds[s,i] = \max(ds[s,i] - IDS^{exp}[s,i]*W_{added}[s,i,j], 0)$
  end for
  if $ds[s,i] < 1$ **then** Add i to $S_{unsatisfied}[s]$ group
  else Remove i to $S_{unsatisfied}[s]$ group
end for
▷ Weight values have been reset, so $W[s,i] = W_{nom}[s,i]\ \forall i \in s$
$W_{ceded}[s,i,j]=W_{added}[s,i,j]=W_{ceded}[s,i]=W_{added}[s,i] = 0\ \forall i,j \in s$

#Identify underutilized queues of slice *s*
▷ $excess[s,i]$: percentage of the portion of resources that theoretically correspond to queue i and are not occupied
for each queue $i \notin S_{unsatisfied}[s]$ **do**
  $excess[s,i] = \left(\frac{W[s,i]}{W_{total}[s]} - \frac{airtime_T[s,i]}{airtime_T[s]}\right) / \left(\frac{W[s,i]}{W_{total}[s]}\right)$
  if $excess[s,i] > \alpha$ **then**
    Add i to $S_{underutilized}[s]$ group
    $W^{exp}_{ceded}[s,i] = (excess[s,i]\text{-}\alpha)*W[s,i]$ ▷ Extra weight to cede
  end if
end for

#Redistribute weight in order to get a similar satisfied rate Option
$w_{ceded} = \sum_{\forall i \in S_{underutilized}[s]} W^{exp}_{ceded}[s,i]$
▷ Distribute w_{ceded} among queues $\in S_{unsatisfied}[s]$
while $w_{ceded} > 0$ **do**
  Find queue $j \in S_{unsatisfied}$[s] with lower $ds[s,j]$
  Find $i \in S_{underutilized}[s]$ with the highest $W^{exp}_{ceded}[s,i]$
  if $\lceil W_{nom}[s,i] * \beta \rceil < w_{ceded}$ **then** $step = \lceil W_{nom}[s,i] * \beta \rceil$
  else $step = w_{ceded}$
  w_{ceded} -= $step$
  $W^{exp}_{ceded}[s,i]$ -= $step$
  $W_{ceded}[s,i]$ += $step$ and $W_{ceded}[s,i,j]$ += $step$
  $W_{added}[s,j]$+= $step$ and $W_{added}[s,j,i]$ += $step$
  $W[s,i]$ -= $step$
  $W[s,j]$ += $step$
  $ds[s,j]$ += $IDS^{exp}[s,j]$ * step
  if $ds[s,j] \geq 1$ **then** Remove j to $S_{unsatisfied}[s]$ group
  if $S_{unsatisfied}[s] = \emptyset$ **then**
    if $R_{demanded}[s,j] > MBR[s,j]$ for any j **then**
      Redefine $S_{unsatisfied}[s]$ with real $R_{demanded}$ (*lines 1-7*)
      if $S_{unsatisfied}[s]$ still empty **then** Exit *while*
    end if
  end if
end while

Algorithm 4 (*continued*)

#Redistribute weight in order to follow priority Option
Calculate the W required by each queue j to be satisfied $W^{need}_{added}[s,j]$
for each queue $i \in S_{underutilized}[s]$ sorted in descending order by $W^{exp}_{ceded}[s,i]$ **do**
  while $W^{exp}_{ceded}[s,i] > 0$ **do**
    Iterate through the queues of $S_{unsatisfied}[s]$ in priority order
    if $W^{need}_{added}[s,j] > W^{exp}_{ceded}[s,i]$ **then**
      $W^{need}_{added}[s,j]$ -= $W^{exp}_{ceded}[s,i]$
      $w = W^{exp}_{ceded}[s,i]$
    else
      $W^{need}_{added}[s,j] = 0$
      $w = W^{need}_{added}[s,j]$
    end if
    $W^{exp}_{ceded}[s,i]$ -= w
    $W_{ceded}[s,i]$ += w and $W_{ceded}[s,i,j] = w$
    $W_{added}[s,j]$ += w and $W_{added}[s,j,i] = w$
    $W[s,i]$ -= w
    $W[s,j]$ += w
    $ds[s,j]$ += $IDS^{exp}[s,j]$ * w
    if $ds[s,j] \geq 1$ **then** Remove j to $S_{unsatisfied}[s]$ group
    if $S_{unsatisfied}[s] = \emptyset$ **then**
      if $R_{demanded}[s,j] > MBR[s,j]$ for any j **then**
        Redefine $S_{unsatisfied}[s]$ with real $R_{demanded}$ (repeat *lines 1-7*)
        if $S_{unsatisfied}[s]$ still empty **then** Exit *while*
      end if
    end if
  end while
end for

Recall that, for intra-slice resource allocation, the airtime share of queue *i* within slice *s* is proportional to the ratio between *W [s,i]* and the sum of the weights of all active queues in the slice. By adjusting *W [s,i]*, we implicitly control the corresponding quantum *Q [s,i]*, which is derived from the slice quantum *Q [s]* as defined in Eq. (3). The objective of the adaptation algorithm is to adjust *W [s,i]* around its nominal value—W_{nom} *[s,i]*—increasing it for unsatisfied queues and ceding it for underutilized ones.

Specifically, an unsatisfied queue *i* may increase its weight according to Eq. (9), where W_{added} *[s,i]* denotes the total weight received from underutilized queues, and W_{added} *[s,i,j]* corresponds to the weight transferred from queue *j*.

$$W[s,i] = W_{nom}[s,i] + W_{added}[s,i] \tag{9}$$

Meanwhile, an underutilized queue *i* decreases its weight according to Eq. (10), where W_{ceded} *[s,i]* denotes the total weight relinquished to queues in the set $S_{unsatisfied}$ *[s]* and W_{ceded} *[s,i,j]* corresponds to the weight ceded to queue *j*.

$$W[s,i] = W_{nom}[s,i] - W_{ceded}[s,i] \tag{10}$$

Note that, for all *i* and *j*, W_{ceded} *[s,i,j]*=W_{added} *[s,j,i]*, ensuring that the total sum of weights for all queues in slice *s*, W_{total} *[s]*, remains unchanged.

Generally, *W [s,i]* is defined by Eq. (11), but W_{added} *[s,i]*>*0* and W_{ceded} *[s,i]*>*0* mutually exclusive.

$$W[s,i] = W_{nom}[s,i] + W_{added}[s,i] - W_{ceded}[s,i] \tag{11}$$

Another important implementation detail is the granularity at which the portions of W_{nom} *[s,i]* are ceded. Granularity can be adjusted according to the order of magnitude of W_{nom}. Although the weights are dimensionless, their values can range, for example, from tens to thousands. Larger values allow finer granularity in allocating resource portions within a slice to its queues. When W_{nom} values are in the order of

Table 3
Variables intra-slice redistribution.

Variable	Description
T	Actualization interval
$airtime_T$ [s,i]	Estimated airtime for all the packets served of queue i of slice s in the interval T, where $n_{retries}$ is the number of retries. $\Sigma_{\forall\ k\in\ T}\ (1 + n_{retries}\ [s,i,k] \cdot airtime\ [s,i,k])$
$airtime_T$ [s]	Estimated airtime for all queues of slice s. $\Sigma_{\forall\ i\in\ s}\ airtime_T\ [s,i]$
$airtime_T$	Total estimated airtime for all slices. $\Sigma_{\forall\ s}\ airtime_T$ [s]
MBR [s,i]	Maximum Bit Rate of queue i of slice s.
$R_{demanded}$ [s,i]	Data rate in bps to be transmitted of queue i.
$R_{achieved}$ [s,i]	Effective data rate in bps (IP level) of queue i.
$S_{unsatisfied}$ [s]	Group of unsatisfied queues of slice s.
$S_{underutilized}$ [s]	Group of queues of slice s that have weight to cede.
W_{nom} [s,i]	Nominal weight of queue i in slice s (controller - configured).
W [s,i]	Used weight by queue i of slice s in the interval T. W_{nom} [s,i] by default. W_{nom} [s,i] - W_{ceded} [s,i] if queue i gives weight to other unsatisfied queues of the same slice s. W_{nom} [s,i] + W_{added} [s,i] if queue i receives weight from satisfied queues of the same slice s.
W_{total} [s]	Sum of weights of the queues in slice s. It remains constant throughout all the time intervals. $\Sigma_{\forall\ i\in\ s}\ W_{nom}$ [s,i]
W_{ceded} [s,i,j]	Weight ceded by the queue i to the queue j of slice s. Same as W_{added} [s,j,i]
W_{ceded} [s,i]	Total weight ceded by the queue i of slice s. $\Sigma_{\forall\ j\in\ S\ unsatisfied\ [s]}\ W_{ceded}$ [s,i,j]
W_{added} [s,i,j]	Weight received by the queue i from the queue j of slice s. Same as W_{ceded} [s,j,i]
W_{added} [s,i]	Total weight received by the queue i of slice s. $\Sigma_{\forall j\in S\ underutilized\ [s]}\ W_{added}$ [s,i,j]
W^{exp}_{ceded} [s,i]	Expected extra weight from queue i to be ceded.
W^{need}_{added} [s,i]	Weight required by queue i of slice s to be satisfied.
W [s,i] /W_{total} [s]	Expected percentage of airtime (at the AP) used by queue i of slice s respect the slice assignment.
DS [s,i]	Degree of satisfaction of queue i of slice s (0=totally unsatisfied, 1= totally satisfied). $R_{achieved}$ [s,i]/$R_{demanded}$ [s,i] if $R_{demanded}$ [s,i] $<\langle$ MBR [s,i] $R_{achieved}$ [s,i]/MBR [s,i] if $R_{demanded}$ [s,i] $>\rangle$ MBR [s,i]
IDS^{exp} [s,i]	Increase of DS [s,i] to queue i of slice s expected for an increment of weight

tens, ceding weight in minimum steps of one is reasonable. For larger values, weight can be ceded in larger steps.

Finally, based on the performance analysis of the system and the proposed constant-sum-of-weights framework, an increase of one unit in the weight corresponds to an expected increase of IDS^{exp} *[s,i]* in the *DS [s,i]*. IDS^{exp} *[s,i]* is proportional to the ratio between the measured *DS [s, i]* and the weight *W [s,i]* used in the previous interval *T*.

Based on these principles, steps #2 and #3 of the intra-slice redistribution algorithm are effectively done in four phases, detailed in Algorithm 4, with the first three phases associated with queue classification.

Phase 1 (lines 1–8) computes *DS [s,i]* for all the queues using statistics collected (by the auxiliary interface) over the previous interval *T*, preliminarily identifying unsatisfied queues of slice *s* (lines 19–25). Unsatisfied queues are added to the $S_{unsatisfied}$ *[s]* group, while the expected IDS^{exp} *[s,i]* is computed and *DS [s,i]* is stored in a provisional variable *ds[s,i]*.

Phase 2 (lines 9–18) introduces a correction in the estimation of provisional DS values (*ds*) with the sole objective of reducing the computational complexity of the weight redistribution process among queues (phase 4) and avoiding the need to simultaneously track all the cessions and additions made during the previous interval *T* among queues.

The precision needed for effective resource allocation adjustments remains unaltered. Phase 2 assumes that all queues revert to their nominal weights W_{nom}. Any previously added weights are returned to the contributing queues, and all W_{ceded} and W_{added} variables reset to zero. Accordingly, the provisional DS values are updated (reduced or increased up to 1) using the estimated IDS^{exp} on phase 1 (lines 11–14). Then, queues are reclassified or removed from $S_{unsatisfied}$ *[s]* (lines 16–17). Unsatisfied queues have been identified, but it is still necessary to find which queues can cede resources to them in phase 3.

Phase 3 (lines 19–25) identifies queues capable of ceding weight. For each queue not in $S_{unsatisfied}$ *[s]* group, the algorithm computes the percentage of the portion of resources that theoretically corresponds to a queue *i* and that has not been occupied in the last period (*excess [s,i]*, line 20). Note that ratios *W [s,i]*/W_{total} *[s,i]* and $airtime_T$ *[s,i]*/$airtime_T$ *[s]* represent, respectively, the theoretically guaranteed portion of resources and the actual portion used in the previous monitoring period *T*. The *excess [s,i]* is then obtained by dividing the difference between these ratios by the theoretical value. Only if *excess [s,i]* exceeds a predefined margin α (e.g., 20 %), the queue is added to set $S_{underutilized}$ *[s]* group, and up to (*excess [s,i]*-α).*W [s,i]* weight units (line 23) may be ceded (denoted as $W^{exp}_{ceded}[s,\ i]$). Margin α reduces the probability that an underutilized queue becomes unsatisfied in the next period. Note that a queue i may be satisfied and *excess [s,i]*<0. This occurs if competing queues have lower activity than the guaranteed or are inactive.

Phase 4 (lines $\geq$26) performs the redistribution of weights.

When the equal rate satisfaction approach is applied (lines 26–46), weight is iteratively transferred from queues in $S_{underutilized}$ *[s]* to those in $S_{unsatisfied}$ *[s]* in discrete steps—one at a time or larger, controlled by parameter β (e.g., 0.01 or 0.001)— up to the total surplus available (stored in $w_{ceded,}$ line 26). As long as there are unsatisfied queues and $w_{ceded}>0$, the algorithm subtracts a weight equal to the step size from the queue with the highest ceding capacity (lines 29, 32–34, 36) and assigns it to the queue with the lowest *ds* (lines 28, 35, 37). The expected DS (*ds* in line 38) of the receiving queue is then updated using IDS^{exp} values obtained in Phase 1 multiplied by the chosen step. Once all queues reach $ds = 1$ with respect to their min($R_{demanded}$, MBR), they are considered satisfied. At this point, if $w_{ceded}>0$ and some queues still have $R_{demanded}>MBR$, several alternatives are possible. The algorithm may either stop—ADWRR will naturally redistribute any excess airtime according to the final weights—or redefine $S_{unsatisfied}$ *[s]* using the actual $R_{demanded}$ (lines 1–8) and repeat the process (lines 27–46).

When the priority-based approach is applied (lines 47–73), the implementation of phase 4 is quite different. First, the weight required by each unsatisfied queue to reach $DS = 1$ with respect to its min($R_{demanded}$, $-MBR$) is computed using its IDS^{exp} from Phase 1 and stored in W^{need}_{added}. Then, starting from the most underutilized queue, its extra weight (W^{ext}_{ceded}) is distributed among unsatisfied queues in descending order of priority. Updates of weights and intermediate variables follow the same principle as in the equal rate satisfaction criterion. Similarly, any surplus weight may again trigger a repetition of the redistribution using the actual $R_{demanded}$ to redefine $S_{unsatisfied}$ *[s]*.

6.2. Inter-slice quantum redistribution

The pseudocode of the inter-slice quantum redistribution algorithm mirrors that of the intra-slice scheduling algorithm and is obtained by straightforward variable substitution.

The notation is defined in Table 4. The main differences are as follows. Updates operate on slice quantum values *Q [s]* rather than on weights; consequently, weight-related variables (W_{ceded}, W_{added},…) are replaced by their quantum counterparts (Q_{ceded}, Q_{added},…). All remaining variables—such as *DS*, IDS^{exp}, *excess*,…—as well as the sets $S_{unsatisfied}$ and $S_{underutilized}$, are defined referred to the slices. Finally, the MBR of slice *s, MBR [s]*, is computed as the sum of the MBRs of its queues.

Inter-slice and intra-slice adaptation algorithms operate independently, updating *W [s,i]* and *Q [s]*, which remain constant during each slicing interval *T*. However, in each scheduling round, *Q [s,i]* is recomputed in Algorithms 1 and 2 according to the number of active queues.

Table 4
Variables inter-slice redistribution.

Variable	Description
MBR [s]	Maximum Bit Rate of slice *s*
$R_{demanded}$ [s]	Data rate in bps to be transmitted of slice *s*.
$R_{achieved}$ [s]	Effective data rate in bps (IP level) of slice *s*.
$S_{unsatisfied}$	Group of unsatisfied slices.
$S_{underutilized}$	Group of slices that have quantum to cede.
Q [s]	Used quantum by slice *s* in the interval *T*. Q_{nom} [s] by default. Q_{nom} [s] − Q_{ceded} [s] if slice *s* reduces the quantum in a quantity which is given to other unsatisfied slices. Q_{nom} [s] + Q_{added} [s] if slice s receives ceded quantum for one or more satisfied slices.
Q_{nom} [s]	Nominal quantum of slice s (configured by the controller).
Q_{total}	Sum of quantum values of the slices (ΣQ_{nom} [s]) remains constant throughout all the time intervals.
Q_{ceded} [s,j]	Quantum ceded by slice *s* to slice *j* (same as Q_{added} [s,j])
Q_{ceded} [s]	Total quantum ceded by slice *s*. $\Sigma_{\forall\ j \in\ S\ unsatisfied}\ Q_{ceded}$ [s,j]
Q^{exp}_{ceded} [s]	Expected extra quantum from slice *s* to be ceded.
Q_{added} [s,j]	Quantum received by slice *s* from slice *j* (same as Q_{ceded} [j,s])
Q_{added} [s]	Total quantum received by slice *s*. $\Sigma_{\forall\ j \in\ S\ underutilized}\ Q_{added}$ [s,j]
Q^{need}_{added} [s]	Quantum required by slice *s* to be satisfied.
Q [s] / Q_{total}	Expected percentage of airtime (at the AP) used by slice *s* respect the total usage.
DS [s]	Degree of satisfaction of slice *s*. (0=totally unsatisfied, 1= totally satisfied). $R_{achieved}$ [s]/$R_{demanded}$ [s] if $R_{demanded}$ [s] < *MBR* [s] $R_{achieved}$ [s]/*MBR* [s] if $R_{demanded}$ [s] > *MBR* [s]
IDS^{exp} [s]	Expected increase of *DS* [s] for an increment of quantum.

6.3. Experimental results

This section presents and discusses the experimental evaluation of the proposed quantum/weight adaptation methods. Two scenarios are considered: evaluates intra-slice adaptations only, with inter-slice adjustments disabled, while the second examines overall system performance with both inter- and intra-slice adaptations enabled.

Test conditions for both scenarios are summarized in Table 5. The experimental setup corresponds to the same laboratory environment described in Section 5, with minimal but non-zero external interference. Traffic injection is performed using IEEE 802.11n/HT in the 5 GHz band, with STAs located within a 2 m radius of the AP. Nine downlink UDP flows are generated, each targeting a distinct STA and using a fixed MCS to emulate heterogeneous propagation conditions. Each flow is mapped to a single SC and slice, and the flows are distributed across three slices, with slice 2 comprising up to four SCs.

Table 5 summarizes the nominal quantum (Q_{nom}) and weight (W_{nom}) configured for each slice and SC, as well as the iperf transmission rates over a 70-second interval, packet sizes per flow, and the corresponding MBR values. These settings yield distinct traffic patterns across flows.

Contrary to the standard definition, and to facilitate the visualization of the results analysis, the MBR values are expressed as net data rates (excluding IP/UDP overhead). In contrast to Section 5, frame aggregation is enabled (Table 5 includes the maximum aggregation size). While this increases achievable throughput, it also introduces variable A-MSDU sizes and airtime demands. Finally, flows directed to STA2 and STA3 share the same SC (queue 0) in slice 1, enabling analysis of scenarios where multiple flows from different STAs are assigned to the same queue.

6.3.1. Experiment 1 – intra-slice redistribution

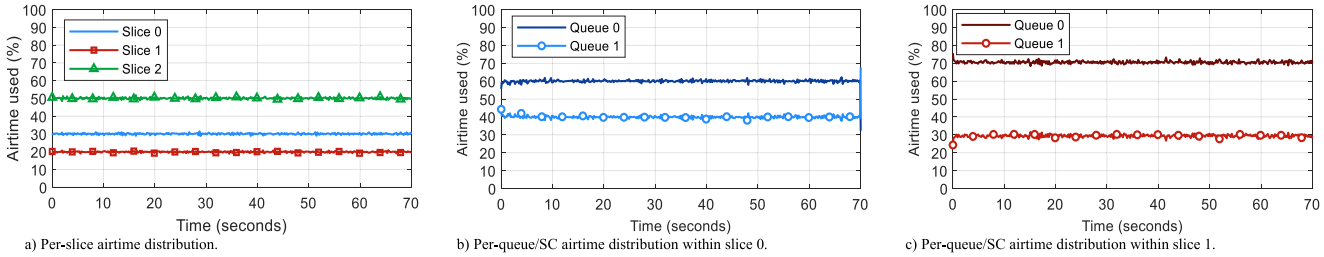
This section presents the results of Experiment 1, in which the intra-slice redistribution algorithms are applied with a 1-second update interval, while inter-slice scheduling relies on a static quantum (ADWRR). Two intra-slice strategies—equal rate satisfaction (Equal DS) and priority-based (Priority Index)—are evaluated against a static weight baseline (Static Weight). In the Priority Index strategy, a lower index indicates higher priority. The results demonstrate that these algorithms effectively redistribute resources according to the chosen criterion, maintain guaranteed airtime relative to nominal values when required, and prevent throughput degradation for ceding SCs.

The experiment was designed to focus on the intra-slice redistribution in slice 2, while traffic in slices 0 and 1 remained constant and high, preventing any airtime redistribution between slices. The same applies to the service classes of slices 0 and 1. Consequently, Fig. 13a confirms that downlink airtime is distributed across slices according to the guaranteed quantum values (30 %, 20 %, and 50 %), while Fig. 13b and c show that the same behavior is observed for SCs within slices 0 and 1, respectively. It should be noted that these results were obtained using the Equal DS strategy, although similar outcomes were observed for the other two scenarios.

Fig. 14 shows the results for the SCs/queues in slice 2, with columns (a), (b), and (c) corresponding to the results of Equal DS, Priority Index,

Table 5
List of input parameters used during the experimentation of dynamic proposals.

Parameters		Values								
Slices		**Slice 0**		**Slice 1**			**Slice 2**			
Quantum (*Qnom*)		3000µs		2000µs			5000µs			
Queues		**Queue 0**	**Queue 1**	**Queue 0**		**Queue 1**	**Queue 0**	**Queue 1**	**Queue 2**	**Queue 3**
Weight (*Wnom*)		120	80	140		60	70	60	40	30
MBR		2.5Mbps	2Mbps	2.5Mbps		2Mbps	3Mbps	2.5Mbp	2.5Mbps	2Mbps
STA id		STA 0	STA 1	STA 2	STA 3	STA 4	STA 5	STA 6	STA 7	STA8
Priority index		0	1	0		1	0	1	2	3
MCS index		2	4	3	2	6	2	3	3	4
Iperf Rate (Time interval)	Experiment 1	2.5Mbps (0-70 s)	2Mbps (0-70 s)	1Mbps (0-70 s)	0.8Mbps (0.70 s)	1.2Mbps (0-70 s)	1.2 Mbps (0 s-10 s), 2,8 Mbps (10 s-40 s), 3,3 Mbps (40 s-70 s)	2.5Mbps (0 s-10s), 2 Mbps (10 s-20 s), 1 Mbps (20 s-60 s) 1.3Mbps (60 s-70s)	1.3 Mbps (0 s-10s), 1.6 Mbps (10 s-30 s), 2.3Mbps (30 s-70 s)	1.25 Mbps (0 s-50s), 0.5 Mbps (50s-70s),
	Experiment 2	2.5 Mbps (0-10 s), 1Mbps (10s-60s) 2Mbps (60s-70s)	2Mbps (0s-20s), 3Mbps (20s-30s), 1Mbps (30s-70s)	1.6Mbps (0-70s)	1.2Mbps (0-70s)	1.2Mbps (0-20s), 2Mbps (20s-70s)				
UDP size (bytes)		1250B	250 B	200 B	350 B	700 B	350 B	350 B	200B	250B
Max Aggregation Size		1200B	1200B	1200B		1200B	1200B	1200B	1200B	1200B



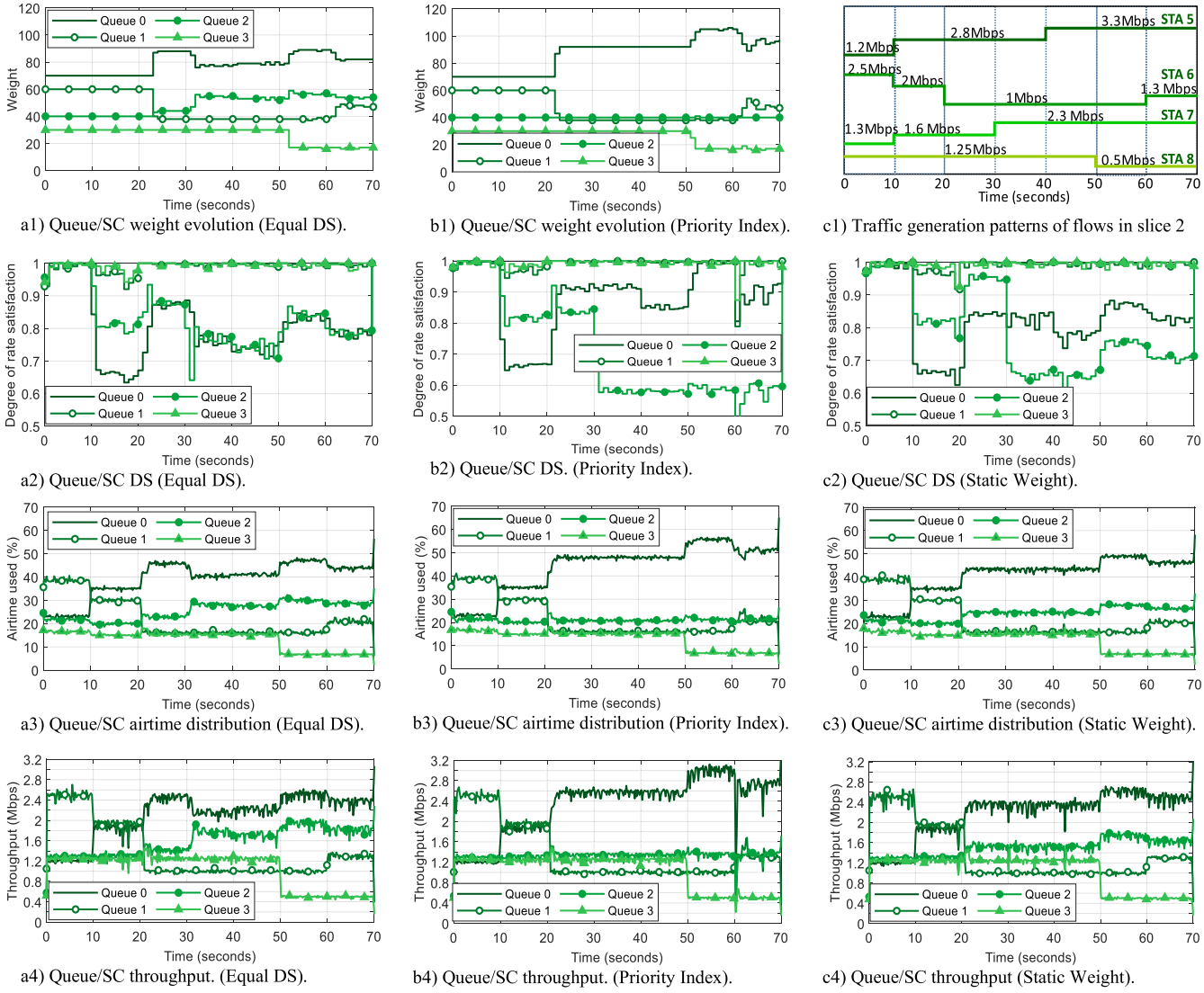
a) Per-slice airtime distribution. b) Per-queue/SC airtime distribution within slice 0. c) Per-queue/SC airtime distribution within slice 1.

Fig. 13. Experiment 1: Airtime distribution across slices (percentage) and among service classes within slices 0 and 1 (time average: 200 ms).

and Static Weight schemes, respectively. To evaluate the proposed schemes, we analyze the weight adaptation (file 1) and the degree of satisfaction (DS, file 2), with DS capped at MBR and $T = 1$ s. Files 3 and 4 display the airtime share and the effective throughput (averaged over 200 ms). Exceptionally, since the weights for the static weight option remain unchanged, Fig. 14c1 includes traffic patterns to aid interpretation. Note that the DS monitoring and weight update windows are not aligned with the traffic flow start time.

Complementarily, to illustrate the impact of aggregation, Fig. 15a and b show the effective packet/MSDU transmission rate and the actual aggregated frame rate (A-MSDUs/s), respectively, under the equal rate satisfaction criterion. The comparison reveals how the number of MSDUs per A-MSDU varies over time, while airtime estimation remains accurate, as confirmed in Fig. 14.

The temporal evolution shown in Fig. 14(a–c) highlights three distinct operational phases:



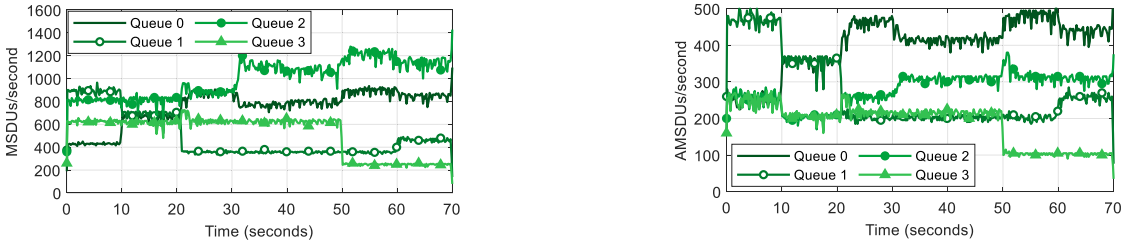
a1) Queue/SC weight evolution (Equal DS). b1) Queue/SC weight evolution (Priority Index). c1) Traffic generation patterns of flows in slice 2

a2) Queue/SC DS (Equal DS). b2) Queue/SC DS. (Priority Index). c2) Queue/SC DS (Static Weight).

a3) Queue/SC airtime distribution (Equal DS). b3) Queue/SC airtime distribution (Priority Index). c3) Queue/SC airtime distribution (Static Weight).

a4) Queue/SC throughput. (Equal DS). b4) Queue/SC throughput. (Priority Index). c4) Queue/SC throughput (Static Weight).

Fig. 14. Experiment 1. Comparison of weight redistribution strategies for the service class (SC)/queues of slice 2. Each column of figures corresponds to a different weight redistribution strategy: (a) redistribution aimed to achieve similar rate satisfaction (Equal DS), (b) redistribution based on the priority index (Priority Index), and (c) static weight configuration (Static Weight). Traffic generation patterns are shown in c1. The evaluated metrics include the evolution of weight values (a1, b1), the degree of rate satisfaction (DS) (a2, b2, c2), airtime distribution among SCs and queues (a3, b3, c3), and the achieved effective throughput (a4, b4, c4).



a) MSDUs per second

b) A-MSDUs per second.

Fig. 15. Experiment 1. Results for the service class (SC)/queues of slice 2: effective MSDUs per second and A-MSDUs per second. Option: Weight redistribution to achieve similar rate satisfaction (Equal DS).

1) From $t = 0$ s to 10 s, the AP channel is not saturated, allowing ADWRR to naturally redistribute unused resources from underutilized SCs (e.g., queue 0) to those in need, without triggering weight or quantum adaptation.
2) From $t = 10$ s to 20 s, the channel becomes saturated, and all SCs operate strictly according to their guaranteed airtime defined by the nominal weights (35 %, 30 %, 20 %, and 15 %, respectively).
3) From $t = 20$ s onward, although the channel saturation persists, queue 1′s demand falls below its guaranteed airtime, theoretically allowing the weight-redistribution process to begin around $t \approx 21$–22 s. However, this redistribution does not occur immediately because queue 1 still has pending packets and is only triggered once sufficient airtime margin becomes available, which in practice occurs around $t \approx 22$–23 s. Starting from $t = 20$ s, we observe that the buffer begins to drain, as reflected in the throughput results. In the Equal-DS case, external interference around $t \approx 16$–18 s (visible as a drop in throughput across all queues in Fig. 14.a4) prolongs the buffer draining period, further delaying redistribution. A similar interference ($t \approx 29$ s) and delay in airtime redistribution is observed in the Static Weight configuration.

Once redistribution begins at $t \approx 22$–23 s, the Equal-DS and Priority Index criteria exhibit distinct performance characteristics, showing notable differences in airtime allocation, throughput, and rate satisfaction compared to the Static Weight scheme.

In all cases, queue 3 retains its guaranteed airtime and stable throughput, achieving high satisfaction, while queues 0 and 2, being severely unsatisfied, become candidates to receive weight cessions from queue 1.

In the Equal DS case (column a), queues 0 and 2 achieve similar satisfaction levels (DS, Fig. 14a1) despite variations in traffic demands during this period, benefiting from further weight redistribution at $t = 50$ s when queue 3 also becomes a ceding queue. Note that queue 2′s degree of satisfaction briefly drops between $t = 30$ and 31 s, as its demand increases while no other queue's traffic decreases, causing weight adaptation to occur in two steps over the following 2 s.

On the contrary, under the Priority Index criterion (column b), queue 0 receives all the ceded weight until $t = 50$ s (Fig. 14b1) because it cannot fully satisfy its required rate (Fig. 14b2), while queue 2 benefits only afterward. Between $t = 50$ s and $t = 60$ s, queue 1 reaches its maximum degree of satisfaction (Fig. 14b2), and a small portion of its unused resources is redistributed to queue 2. This is because queue 0′s throughput (Fig. 14b4) is capped at the MBR (3 Mbps vs. 3.3 Mbps incoming traffic), which limits its ability to fully utilize the resources ceded by queues 1 and 3. Finally, from $t = 60$ s to $t = 70$ s, scheduling is impacted by external interferences in $t = 60$ s and $t = 66$ s (Fig. 14b4). As a result, ceding queues 1 and 3 reclaim part of the ceded weight and then cede it again (Fig. 14b1). In all the cases, all queues maintain their guaranteed airtime, even when they do not receive ceded weight.

Unlike these redistribution criteria, the Static Weight approach (column c) distributes leftover airtime proportionally to nominal weights, resulting in the rate satisfaction and throughput observed in the Fig. 14c2 and c4.

Overall, Fig. 14 demonstrates that intra-slice redistribution effectively ensures fair and controlled resource adaptation under both criteria, preserving the performance of ceding queues and thus validating the proposed framework.

6.3.2. *Experiment 2 – inter- and intra-Slice interaction*

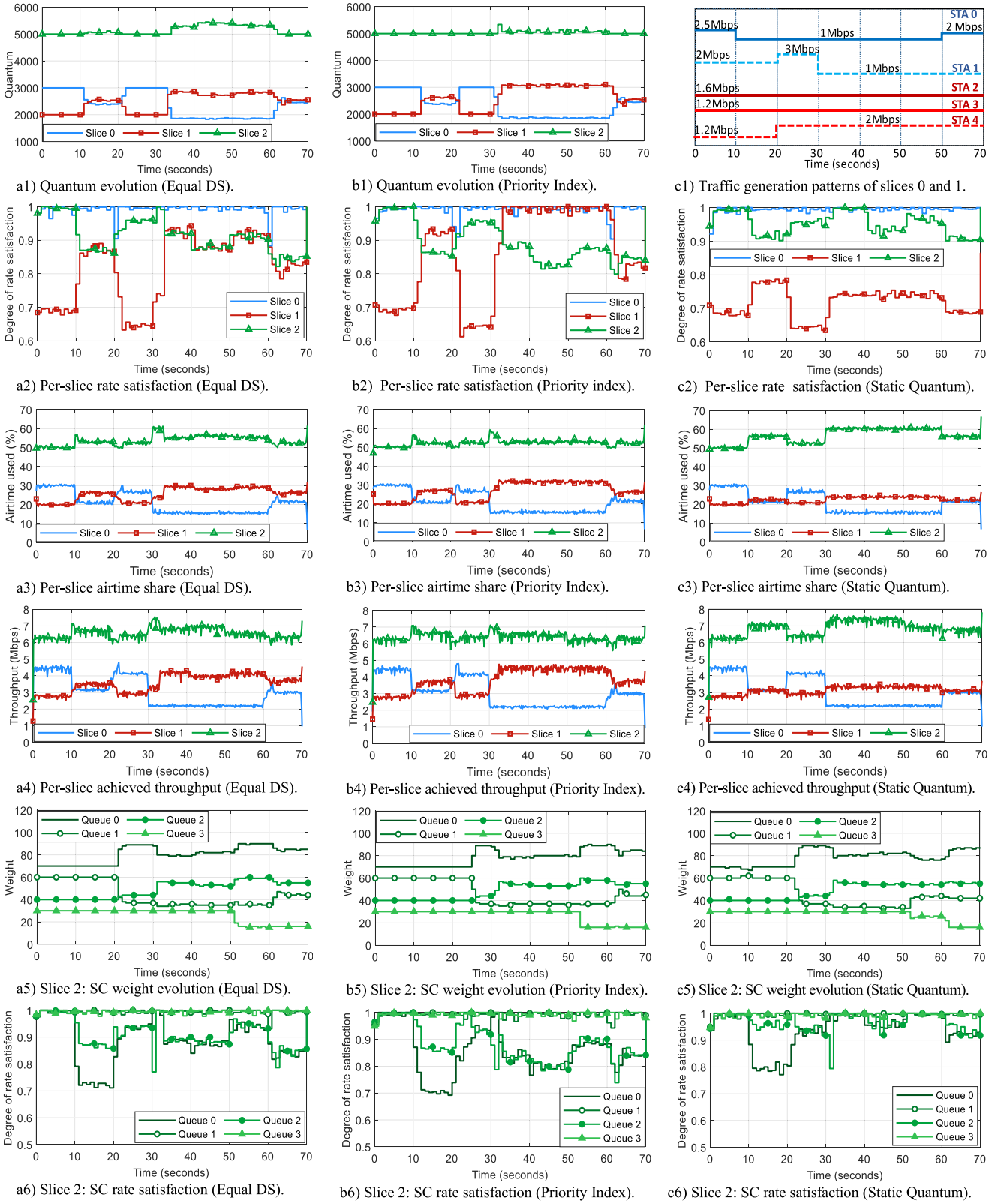
Experiment 2 analyzes the interaction between inter-slice and intra-slice quantum/weight redistribution algorithms. It compares inter-slice quantum redistribution using Equal DS and Priority Index criteria against Static Quantum inter-slice ADWRR scheduling. The results correspond to the configuration with intra-slice Equal DS redistribution, as this scenario yields the most evident and representative outcomes. The traffic demands of slice 2 are the same as in experiment 1, while slices 0 and 1 traffic varies as defined in Table 5.

Fig. 16 shows results for slices 0 to 2 and the SCs of slice 2, with columns (a), (b), and (c) corresponding to the results of Equal DS, Priority Index, and Static Quantum schemes, respectively. The metrics include quantum adaptation (File 1), the degree of rate satisfaction (DS) constrained by each slice's MBR and measured over $T = 1$ s (File 2), as well as airtime share (File 3) and effective throughput (File 4), both averaged over 200 ms intervals. Exceptionally, since the slice's quantum values remain fixed in the Static Quantum option, Fig. 16c1 shows the traffic flow patterns of slices 0 and 1 to aid in the interpretation of the results. Finally, files 5 and 6 display weight adaptation and DS evolution for the SCs/queues in slice 2.

Throughout the 70-second experiment, the AP channel remains saturated. Although this is not the standard operating condition, saturation highlights the differences among schemes; otherwise, ADWRR would naturally redistribute resources, rendering quantum adaptation unnecessary.

The temporal evolution shown in Fig. 16(a–c) shows four distinct operational phases.

1) From $t = 0$ s to $t \approx 11$–12 s, no slice has unused resources. Therefore, airtime distribution among slices (Fig. 16a3-c3) matches their guaranteed percentages (30 %, 20 %, and 50 %), aligned with the nominal quantum values (Fig. 16a1-b1).
2) At $t = 10$ s, SC0's traffic demand in slice 0 decreases, triggering quantum redistribution since slice 0′s demand falls below its guaranteed airtime.
3) Between $t = 20$ s and $t = 30$ s, however, the traffic demand of SC1 in slice 0 rises, rendering slice 0′s excess resources insufficient to sustain the redistribution. In this case, ADWRR naturally manages the redistribution process.



a1) Quantum evolution (Equal DS). b1) Quantum evolution (Priority Index). c1) Traffic generation patterns of slices 0 and 1.

a2) Per-slice rate satisfaction (Equal DS). b2) Per-slice rate satisfaction (Priority index). c2) Per-slice rate satisfaction (Static Quantum).

a3) Per-slice airtime share (Equal DS). b3) Per-slice airtime share (Priority Index). c3) Per-slice airtime share (Static Quantum).

a4) Per-slice achieved throughput (Equal DS). b4) Per-slice achieved throughput (Priority Index). c4) Per-slice achieved throughput (Static Quantum).

a5) Slice 2: SC weight evolution (Equal DS). b5) Slice 2: SC weight evolution (Priority Index). c5) Slice 2: SC weight evolution (Static Quantum).

a6) Slice 2: SC rate satisfaction (Equal DS). b6) Slice 2: SC rate satisfaction (Priority Index). c6) Slice 2: SC rate satisfaction (Static Quantum).

Fig. 16. Experiment 2. Comparison of quantum redistribution strategies across slices. Columns correspond to a different quantum redistribution strategy: (a) to achieve similar rate satisfaction (Equal-DS), (b) based on the priority index (Priority Index), and (c) static quantum configuration (Static Quantum). Intra-slice weight redistribution is used to achieve similar rate satisfaction (Equal-DS). Traffic generation patterns are shown in c1. Evaluated per-slice metrics include: quantum evolution (a1, b1), rate satisfaction (DS) (a2, b2, c2), airtime distributions (a3, b3, c3) and achieved throughput (a4, b4, c4), In addition, results for the service class (SC)/queues of slice 2 are shown: weight evolution (a5,b5,c5) and rate satisfaction (a6,b6,c6).

4) From $t = 30$ s onward, SC1's traffic demand in slice 0 decreases, reactivating the redistribution of slice 0's excess resources. This redistribution is delayed beyond $t = 31$ s, as the system requires extra time to reallocate resources and stabilize the rate satisfaction (DS) measurements. Once redistribution is reactivated ($t \approx 31$–33 s), the inter-slice quantum adaptation criteria, as in intra-slice scheduling, result in substantially different airtime distributions, effective throughputs, and rate satisfaction levels compared to each other and to the static quantum configuration.

When the Equal DS option is applied, slices 1 and 2 converge to similar rate satisfaction levels (DS, Fig. 16a2) as a result of the quantum increases applied from $t = 33$ s to $t = 61$ s (and previously during $t =$ 11–21 s). After $t = 61$ s, however, the quantum ceded by slice 0 decreases; consequently, only slice 1 receives additional quantum in an attempt to match slice 2's higher satisfaction, although this proves insufficient. In any case, during this interval, slice 0's reserved quantum margin (α % of excess resources) is naturally redistributed to slices 1 and 2 based on the updated quantum. This explains why slice 2's airtime exceeds its guaranteed value.

In contrast, under the Priority Index option (column b), slice 1 receives most—but not all—of the ceded quantum from $t = 33$ s to $t = 60$ s, allowing it to reach its maximum achievable rate satisfaction (Fig. 16b1). In Fig. 16b4, we observe that this maximum DS corresponds to 4500 Mbps (tied to MBR), not the 4800 Mbps required rate. Some ceded quantum is given to slice 2, improving its rate satisfaction compared to Experiment 1. After $t = 62$ s, slice 1 receives all ceded quantum, similar to the equal rate criterion.

Using the Static Quantum option (column c), ADWRR naturally redistributes airtime, increasing slices 1 and 2 proportionally to their nominal quantum. As a result, slice 2 attains higher airtime than with quantum adaptation. Between $t = 32$–40 s and $t = 51$–60 s, its SCs reach a DS of 1 (Fig. 16c6), meeting rates limited by MBR (≈7.45 and 6.8 Mbps vs. 7.1 Mbps in the second interval). These SC-level constraints explain why slice 2's overall DS (Fig. 16c2) remains below 1 from $t = 51$ s to 60 s.

The most noteworthy aspect is the interaction between inter-slice and intra-slice quantum/weight redistribution algorithms. As shown in Fig. 16(a6-c6), regardless of the inter-slice scheme, intra-slice dynamic weight adaptation—which affects the SCs of slice 2 (activated after $t =$ 21 s, Fig. 16x5 with x=a–c)—consistently achieves equal DS objectives.

Finally, it is important to note that the throughput of the ceding slice 0 is consistently maintained and fully satisfied in all cases. As shown in Fig. 16a4, throughput is lower than expected at $t = 20$ s and $t = 60$ s due to delays in quantum re-adaptation when the required rate increases. However, the buffer is emptied promptly during $t =$ 21–22 s and $t =$ 61–62 s. Notably, the effective throughput, observed at $t = 23$ s, exceeds the MBR limit of the slice (4500 Mbps). This is possible because the guaranteed airtime based on nominal quantum (30 %) is not surpassed.

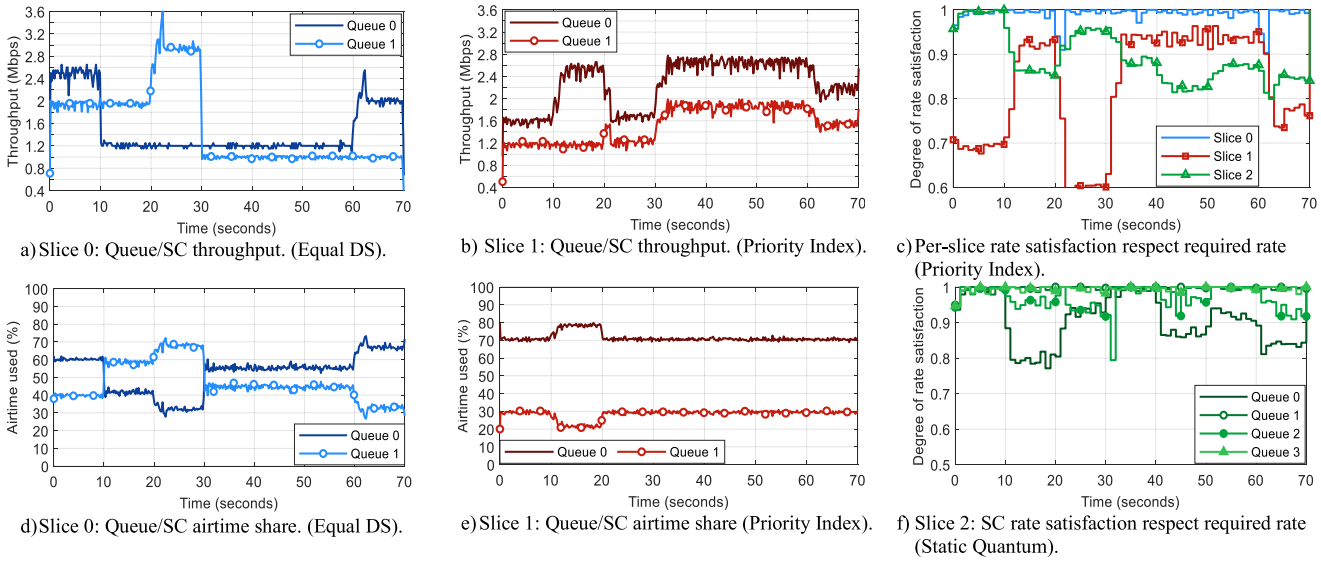
Fig. 17 complements the metrics presented in Fig. 16.

As an illustrative example of slice 0's behavior across all options, Fig. 17a and d show the throughput and airtime distribution, respectively, for the Equal DS quantum adaptation option. As observed, since the guaranteed resources of slice 0 are not saturated, ADWRR naturally redistributes airtime between its SCs based on their needs.

Fig. 17b and e show throughput and airtime distribution for slice 1, respectively, under Priority Index quantum adaptation. In this case, the goal is to highlight that, between $t = 32$ s and $t = 60$ s, although the slice's overall throughput stays below its MBR (4500 Mbps), SC0's throughput can exceed its MBR (2500 Mbps) due to guaranteed airtime allocation. By comparing Fig. 16b2 with Fig. 17c. where the degree of rate satisfaction is computed with respect to the required rate, not limited by to the required rate, not limited by the MBR, we can clearly observe how the algorithm enforces the slice's MBR thresholds. Similarly, for slice 1 (composed of only two SCs), natural ADWRR redistribution occurs (e.g., $t = 10$–20 s), ensuring throughput and guaranteed airtimes according to nominal values.

Finally, for the SCs of slice 2 under the Static Quantum option, Fig. 17f shows DS computed relative to the required rate, not constrained by MBR. Comparison with Fig. 16c6 demonstrates how the algorithms enforce SC-level MBR limits.

From Experiments 1 and 2, it is demonstrated that the proposed two-level local quantum/weight adaptation mechanism (ISS-QoSS) enables effective and deterministic modification of the natural ADWRR reallocation to satisfy specific QoS requirements. The framework provides predictable and strictly controlled resource adjustments while simultaneously preserving service guarantees for those slices and service classes (SCs) that relinquish quantum. Furthermore, the demonstrated adaptability confirms that the r framework is not only applicable at the local agent (AP) but can also be extrapolated to other operational levels.



a) Slice 0: Queue/SC throughput. (Equal DS).
b) Slice 1: Queue/SC throughput. (Priority Index).
c) Per-slice rate satisfaction respect required rate (Priority Index).
d) Slice 0: Queue/SC airtime share. (Equal DS).
e) Slice 1: Queue/SC airtime share (Priority Index).
f) Slice 2: SC rate satisfaction respect required rate (Static Quantum).

Fig. 17. Experiment 2. Comparison of effect of quantum redistribution strategies across slices. Complementary results.

7. Conclusions

In this paper, we propose a hybrid ISS-QoSS slicing mechanism, developed on an SDN-based experimental prototype, which incorporates both global and distributed slicing components located at a central controller and distributed APs. Our approach focuses on managing downlink traffic and includes an accurate and effective design of the local slicing component, where a hierarchical inter-intra slice scheduling framework, based on ADWRR principles and incorporating a quantum adaptation, is implemented. When AP channel resources are sufficient, the specific implementation effortlessly ensures a fair and natural airtime redistribution among slices or SCs, meeting the QoS requirements of all tenants and SCs, while always maintaining the guaranteed airtime share when necessary. However, the real contribution occurs when AP channel resources are limited. In this scenario, the specific practical implementation has proven to ensure isolation and resource segmentation among tenants or client categories, while also enabling fine-grained traffic differentiation, prioritization, and resource isolation for various service classes within slices. The most significant contribution lies in the proposed quantum/weight adaptation framework, which is based on ceding and addition principles among slices and SCs. This framework has proven to enable efficient and flexible resource reallocation, facilitating the fulfillment of various QoS criteria through predictable and controlled adjustments, while preserving unaltered the service guarantees for slices and SCs that relinquish quantum/weight. This framework could be extrapolated to define longer-term quantum adaptation methods, implemented in the global scheduler, to orchestrate resources across the entire deployment, composed of several APs, according to the demands of tenants and SCs within a tenant. Enabling channel reuse among overlapping APs within a controlled deployment would require partial adaptation of the slicing framework, including the addition of a global scheduler and treating APs sharing a channel as a single logical entity.

CRediT authorship contribution statement

Ángela Hernández-Solana: Writing – review & editing, Writing – original draft, Validation, Supervision, Software, Methodology, Investigation, Formal analysis, Conceptualization. **José Ruiz-Mas:** Writing – review & editing, Writing – original draft, Validation, Software, Project administration, Methodology, Investigation, Formal analysis, Conceptualization. **María Canales:** Writing – review & editing, Validation, Investigation, Formal analysis. **Julian Fernández-Navajas:** Writing – review & editing, Validation, Investigation, Formal analysis. **José Ramón Gállego:** Writing – review & editing, Validation, Investigation, Formal analysis. **Sara Ibáñez-Alloza:** Software, Formal analysis, Data curation.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This work was supported by Ministerio de Ciencia, Innovación y Universidades/Agencia Estatal de Investigación (MICIU/AEI)/10.13039/501100011033/Fondo Europeo de Desarrollo Regional (FEDER) EU, European Regional Development Fund/European Union (ERDF/EU), Gobierno de Aragón, under Grant T31_23R and Grant PID2022-136476OB-I00.

Data availability

Data will be made available on request.

References

[1] M. Richart, J. Baliosian, J. Serrat, et al., Slicing in WiFi networks through airtime-based resource allocation, J. Netw. Syst. Manag. 27 (2019) 784–814, https://doi.org/10.1007/s10922-018-9484-x.
[2] P. Libório, C. Lam, B. Guidoni, et al., Airtime Aware dynamic network slicing for heterogeneous IoT services in IEEE 802.11ah, in: 2021 IEEE Wireless Communications and Networking Conference (WCNC), Nanjing, China, 2021, pp. 1–6, https://doi.org/10.1109/WCNC49053.2021.9417414.
[3] C. An, D. Zhai, C. Zhong, L. Zhang, S. Shao, Power grid local communication service isolation technology based on WLAN network slice, in: 2023 IEEE Int'l Symposium on Broadband Multimedia Systems and Broadcasting (BMSB), Beijing, China, 2023, pp. 1–5, https://doi.org/10.1109/BMSB58369.2023.10211149.
[4] C. Xu, et al., Research on collaborative networking method of 5G slicing and WiFi6 in substation scenarios, in: 2023 7th Int. Conf. on Electrical, Mechanical and Computer Engineering (ICEMCE), Xi'an, China, 2023, pp. 198–203, https://doi.org/10.1109/ICEMCE60359.2023.10490809.
[5] E. Coronado, R. Riggio, J. Villa1ón, A. Garrido, Lasagna: programming abstractions for end-to-end slicing in software-defined WLANs, in: 2018 IEEE 19th Int'l Symposium on "A World of Wireless, Mobile and Multimedia Networks" (WoWMoM), Chania, Greece, 2018, pp. 14–15, https://doi.org/10.1109/WoWMoM.2018.8449797.
[6] R. Riggio, M.K. Marina, J. Schulz-Zander, S. Kuklinski, T. Rasheed, Programming abstractions for software-defined Wireless networks, IEEE Trans. Netw. Serv. Manag. 12 (2) (2015) 146–162, https://doi.org/10.1109/TNSM.2015.2417772.
[7] P.H. Isolani, et al., Airtime-based resource allocation modeling for network slicing in IEEE 802.11 RANs, IEEE Commun. Lett. 24 (5) (2020) 1077–1080, https://doi.org/10.1109/LCOMM.2020.2977906.
[8] R. Riggio, D. Miorandi, I. Chlamtac, Airtime Deficit Round Robin (ADRR) packet scheduling algorithm, in: 2008 5th IEEE International Conference on Mobile Ad Hoc and Sensor Systems, Atlanta, GA, USA, 2008, pp. 647–652, https://doi.org/10.1109/MAHSS.2008.4660101.
[9] K. Gomez, R. Riggio, T. Rasheed, I. Chlamtac, On efficient airtime-based fair link scheduling in IEEE 802.11-based wireless networks, in: in 2011 IEEE 22nd Int'l Symposium on Personal, Indoor and Mobile Radio Communications, Toronto, ON, Canada, 2011, pp. 930–934, https://doi.org/10.1109/PIMRC.2011.6140105.
[10] T. Høiland-Jørgensen, M. Kazior, D. Täht, P. Hurtig, A. Brunstrom, Ending the anomaly: achieving low latency and airtime fairness in WiFi, in: in 2017 USENIX Annual Technical Conference, July 12–14, 2017. Santa Clara, CA, USA.
[11] P.H. Isolani, et al., SDN-based slice orchestration and MAC management for QoS delivery in IEEE 802.11 networks, in: 2019 Sixth International Conference on Software Defined Systems (SDS), Rome, Italy, 2019, pp. 260–265, https://doi.org/10.1109/SDS.2019.8768642.
[12] K. Zia, A. Chiumento, P. Havinga, R. Riggio, Y. Huang, QoS Aware slice resource management using deep reinforcement learning in IoT networks, in: 2023 19th International Conference on Distributed Computing in Smart Systems and the Internet of Things (DCOSS-IoT), Pafos, Cyprus, 2023, pp. 150–154, https://doi.org/10.1109/DCOSS-IoT58021.2023.00035.
[13] M. Richart, et al., Guaranteed bit rate slicing in WiFi networks, in: 2019 IEEE Wireless Communications and Networking Conference (WCNC), Marrakesh, Morocco, 2019, pp. 1–8, https://doi.org/10.1109/WCNC.2019.8885636.
[14] M. Richart, et al., Slicing with guaranteed quality of service in WiFi networks, IEEE Trans. Netw. Serv. Manag. 17 (3) (2020) 1822–1837, https://doi.org/10.1109/TNSM.2020.3005594.
[15] A. Betzler, D. Camps-Mur, M. Catalan, G-ADRR: network-wide slicing of Wi-Fi networks with variable loads in space and time, IEEE Trans. Mob. Comput. 21 (11) (2022) 3986–4000, https://doi.org/10.1109/TMC.2021.3066875.
[16] J. Saldana, J. Ruiz-Mas, J. Almodóvar, Frame aggregation in Central controlled 802.11 WLANs: the latency versus throughput tradeoff, IEEE Commun. Lett. 21 (11) (2017) 2500–2503, https://doi.org/10.1109/LCOMM.2017.2741940.
[17] P.H. Isolani, et al., Delay-aware slicing and MAC management using MCDA in IEEE 802.11 SD-RANs, in: 2021 IFIP/IEEE Int'l Symposium on Integrated Network Management (IM), Bordeaux, France, 2021, pp. 331–339.
[18] P.H. Isolani, et al., Support for 5G mission-critical applications in software-defined IEEE 802.11 networks, Sensors 21 (2021) 693, https://doi.org/10.3390/s21030693.
[19] K. Koutlia, A. Umbert, R. Riggio, I. Vilà, F. Casadevall, A new RAN slicing strategy for multi-tenancy support in a WLAN scenario, in: 2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft), Montreal, QC, Canada, 2018, pp. 64–70, https://doi.org/10.1109/NETSOFT.2018.8460138.
[20] J.J. Aleixendri, A. Betzler, D. Camps-Mur, A practical approach to slicing Wi-Fi RANs in future 5G networks, in: 2019 IEEE Wireless Communications and Networking Conference (WCNC), Marrakesh, Morocco, 2019, pp. 1–6, https://doi.org/10.1109/WCNC.2019.8885777.
[21] T. Høiland-Jørgensen, P. Hurtig, A. Brunstrom, PoliFi: airtime policy enforcement for WiFi, in: 2019 IEEE Wireless Communications and Networking Conference

(WCNC), Marrakesh, Morocco, 2019, pp. 1–6, https://doi.org/10.1109/WCNC.2019.8885440.
[22] F. Fami, N. Hammami, C. Pham, K.-K. Nguyen, Slicing Wi-Fi networks for differentiated IoT service provisioning, in: 2022 IEEE Wireless Communications and Networking Conference (WCNC), Austin, TX, USA, 2022, pp. 866–871, https://doi.org/10.1109/WCNC51071.2022.9771816.
[23] S.H. Amur, K. Zia, A. Chiumento, P. Havinga, Autonomous network slicing and resource management for diverse QoS in IoT networks, in: 2023 IEEE Int. Conf. on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops), Atlanta, GA, USA, 2023.
[24] J. Saldana, et al., Attention to Wi-fi diversity: resource management in WLANs with heterogeneous APs, IEEE Access 9 (2021) 6961–6980, https://doi.org/10.1109/ACCESS.2021.3049180.
[25] J. Saldana, et al., Unsticking the Wi-Fi client: smarter decisions using a software defined wireless solution, IEEE Access 6 (2018) 30917–30931, https://doi.org/10.1109/ACCESS.2018.2844088.
[26] Julius Schulz-Zander, et al., Programmatic orchestration of WiFi networks, in: USENIX Annual Technical Conference, 2014.
[27] https://www.radiotap.org/.