

Exploiting Scheduled Access Features of mmWave WLANs for Periodic Traffic Sources

Mattia Lecci, Matteo Drago, Andrea Zanella, Michele Zorzi
Department of Information Engineering, University of Padova, Italy
 E-mails: {name.surname}@dei.unipd.it

Abstract—Many current and future multimedia and industrial applications, like video streaming, eXtended Reality or remote robot control, are characterized by periodic data transmissions with strict latency and reliability constraints. In an effort to meet the stringent demand of such traffic sources, the WiGig standards support a contention-free channel access mechanism, named Service Period, that makes it possible to allocate dedicated time intervals to certain wireless stations. However, the standard only covers the fundamental aspects that ensure interoperability, while the actual schedule logic is left to vendors.

In this paper, we propose two algorithms for joint admission control and scheduling of periodic traffic streams with contrasting performance objectives, specifically a *simple scheduler* and a *max-min fair scheduler*. The schemes are compared in two different scenarios, in order to characterize and highlight some fundamental trade-offs. As expected from their design principles, the simple scheduler tends to trade acceptance rate for resource availability, contrary to the max-min fair scheduler, giving to implementers a clear performance trade-off, although performance cannot be balanced by means of a tunable parameter.

Index Terms—WiGig, 802.11ad, 802.11ay, periodic, scheduling, QoS

I. INTRODUCTION

The always-increasing capacity of wireless systems is promoting the design of applications and services with increasingly challenging demands, such as video streaming, teleconference, telepresence, eXtended Reality (XR), among others [1]. In order to meet the demand in terms of data rate of such applications, the latest versions of the Wi-Fi standard, i.e., IEEE 802.11ad and 802.11ay, also known as Wireless Gigabit (WiGig), offer the possibility to communicate over the mmWave band at 60 GHz, where multiple 2.16 GHz channels are available. By taking advantage of techniques such as channel bonding and Multiple Input, Multiple Output (MIMO), and by introducing novel features to the protocol stack, these standards can provide data rates over 100 Gbps [2].

However, many applications also have very stringent Quality of Service (QoS) requirements, in particular in terms of delay and jitter, which may be incompatible with the stochastic nature of contention-based channel access mechanisms generally supported by legacy Wireless Local Area Networks (WLANs). To address this problem, the WiGig standards introduced a

contention-free access mechanism that allows a Station (STA) to reserve radio resources at regular time intervals. These resources are organized in blocks, called Service Periods (SPs), and the standards only specify the basic procedures to ensure inter-vendor compatibility, leaving the design and the implementation of the scheduler to the manufacturers.

Regarding the practical design, handling multiple periodic traffic streams can be problematic, especially when traffic flows with different periodicities coexist. In this case, it is necessary to anticipate collisions among different periodic allocations and either adjust them or, in the worst case, reject new incompatible requests. Furthermore, even if a collection of requests with identical traffic requirements is considered, upon receiving a new request, the scheduler needs to decide whether to rearrange the previously allocated resources to improve fairness and efficiency, or to maintain the original schedule and then best accommodate the new request, in order not to perturb the pre-existing streams but potentially reaching suboptimal resource allocation. Moreover, SPs are subject to a number of constraints, described in Sec. II, which need to be accounted for when designing and optimizing scheduling algorithms.

With these challenges in mind, in this work we address both admission control and resource allocation for multiple periodic traffic sources, following the constraints given by the WiGig standards. Specifically, we cast the periodic scheduling problem within the WiGig allocation framework and design a simple and efficient algorithm to check for the feasibility of a new request. We then propose a simple admission control algorithm with limited scheduling capabilities, as well as a more elaborate and optimized strategy to increase the admission rate and, possibly, the fairness among independent flows. Finally, we compare these two policies and shed some light on basic trade-offs.

The rest of the paper is organized as follows. The resource allocation framework is described in Sec. II, while Sec. III provides an overview of the State of the Art on related problems, and motivates our need to fill the gap of the current literature in this topic. Then, we present the proposed algorithms in Sec. IV. Performance analysis is presented in Sec. V. Finally, in Sec. VI we draw our conclusions and propose possible extensions of this work.

This work was partially supported by NIST under Award No. 60NANB19D122. Mattia Lecci's activities were supported by *Fondazione CaRiPaRo* under the grant "Dottorati di Ricerca 2018."

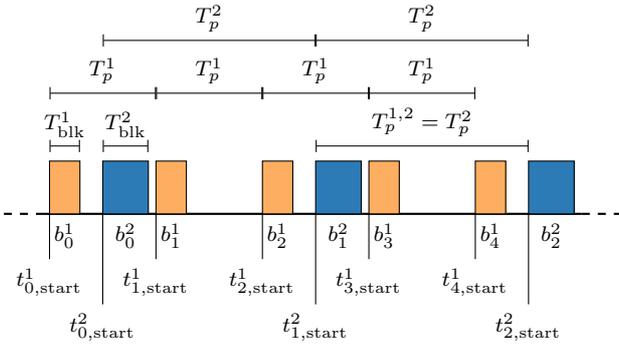


Fig. 1: Example of allocations A_1 (orange) and A_2 (blue), with $p_2 = 2p_1$.

II. FRAMEWORK DESCRIPTION

Based on [3], STAs can request the Access Point (AP) to reserve periodic transmission intervals by sending a control frame containing the required periodicity (p) and the minimum and maximum duration of each allocation ($[T_{\min}, T_{\max}]$). The AP advertises the allocated SPs to the STAs at each Beacon Interval (BI), specifying the starting time, duration, and periodicity of each block. The allocation needs to comply with a number of constraints:

- 1) Periodicity (p) can only be an integer multiple ($p \in \mathbb{N}$) or an integer fraction ($p^{-1} \in \mathbb{N}$) of a BI (T_{BI}), thus the block periodicity interval will be $T_p = p T_{\text{BI}}$;
- 2) Allocation blocks cannot be scheduled across the BI boundaries;
- 3) The allocated block duration T_{blk} should fall in the range $[T_{\min}, T_{\max}]$ specified in the resource request.

A more detailed description of the constraints imposed by the standard can be found in [3] and [4].

Since this work is focused on allocation algorithms for periodic traffic sources, we neglect the Contention-Based Access Periods (CBAPs), which is present in each BI for asynchronous traffic. In addition, to compare the scheduling algorithms in challenging conditions, we assume that the allocated resources will be maintained indefinitely, so that the channel load increases progressively as new resource reservations are accepted. For the sake of simplicity and clarity, we also assume that the allocation blocks of a given accepted request are not fractioned into multiple disjoint intervals (i.e., each SP will consist of a single time interval of duration T_{blk}). Furthermore, we consider a *strict periodicity* constraint, which prevents the scheduler from changing the starting time of already allocated blocks, while the block duration T_{blk} can be freely changed within the interval $[T_{\min}, T_{\max}]$.

III. STATE OF THE ART

Many works analyzing the Medium Access Control (MAC) layer of the WiGig standards focus mostly on CBAPs, which extend the traditional WiFi access to cope with directional communications, either neglecting SP allocations or considering extremely simple allocation schemes. In [5], the authors

proposed a mathematical framework for the analysis of End-to-End (E2E) metrics in 802.11-based systems, comparing throughput and average packet delay in scenarios where the nodes are equipped with advanced antenna systems. The characteristics of the Distributed Coordination Function (DCF) were taken into account, for which a theoretical performance analysis was carried out in [6]. Instead, the authors in [7] present a model to assess the performance of CBAPs for the IEEE 802.11ad standard, taking into account a directional channel model and the presence of scheduled SPs, but they do not focus on how to assign such SPs.

To the best of our knowledge, little work has been done on contention-free scheduling for WiGig networks. In [8], [9], the authors analyze the case where all contention-free allocations occupy the beginning of each BI, while the rest of the interval is left for a single CBAP. This allocation strategy, however, cannot support requests for periodic resource allocations with time period shorter than T_{BI} . The authors of [10], instead, propose an accurate mathematical analysis of the performance of a realistic Variable Bit Rate (VBR) traffic source in the presence of channel errors, when using a periodic resource allocation scheme, but do not tackle the problem of scheduling multiple periodic allocations at once.

On the other hand, the problem of periodic scheduling has been widely studied in other areas, such as real-time computation and task scheduling, where the objective is to complete tasks within a given time, while minimizing the resource utilization. For example, the authors of [11] develop and compare heuristic algorithms for scheduling tasks with hard periodic deadlines and constant resource utilization, showing that a deadline-first approach ensures maximum resource utilization. In [12], the authors try to schedule safety-critical periodic tasks with precedence constraints, distributed over multi-processor systems using an adapted deadline-first approach, while the authors of [13] use simulated annealing to optimize a similar problem. Finally, [14] finds a low-overhead optimal solution (from a resource utilization point of view) assuming that tasks have a fixed resource requirement.

All these approaches, however, cannot be directly used in WiGig systems, either because they are not compliant with the constraints imposed by the resource allocation procedures (i.e., granularity of the allocation periods, BI boundaries), or because they cannot exploit the WiGig standards' flexibility (e.g., the dynamic allocation of T_{blk}). This work contributes to fill the gap by proposing admission control and scheduling algorithms that account for the specific features of Millimeter Wave (mmW) WLANs.

IV. SCHEDULING ALGORITHMS

We denote by $A_n = (t_{0,\text{start}}^n, T_p^n, T_{\text{blk}}^n)$ the allocation for the n -th traffic stream, where $t_{0,\text{start}}^n$ is the starting epoch, T_p^n is the period, and T_{blk}^n is the allocated duration of each individual block. Therefore, the allocation consists of a sequence of

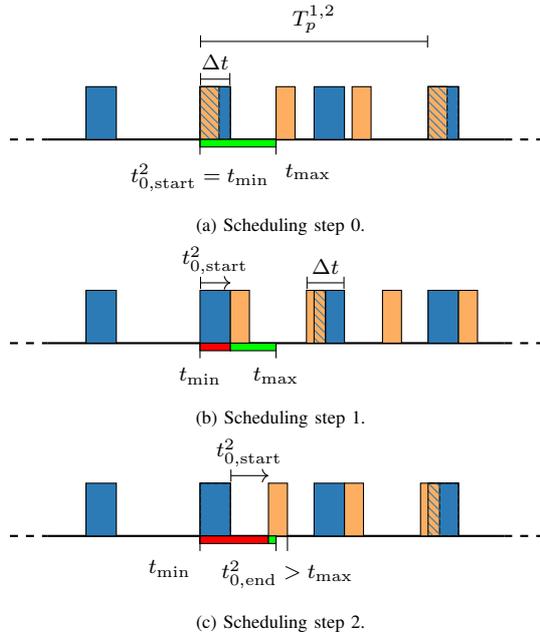


Fig. 2: Feasibility check for an *infeasible* pair of allocations, where A_1 (blue) was a pre-existing allocation with $p_1 = \frac{1}{2}$, and the algorithm is checking whether a new allocation A_2 (orange) with $p_2 = \frac{1}{3}$ is compatible.

Algorithm 1 Feasibility check under strong periodicity conditions (see Figs. 2 and 3).

Require: $\{A_1, \dots, A_{N-1}\}$ (fixed), A_N (new allocation), $[t_{\min}, t_{\max}]$

- 1: Compute $T_p^{1, \dots, N}$
- 2: $t_{0, \text{start}}^N = t_{0, \text{start}}^N \leftarrow t_{\min}$ {Fig. 2a}
- 3: **while** $t_{0, \text{end}}^N < t_{\max}$ **do**
- 4: Check for collisions in $[t_{\min}, t_{\min} + T_p^{1, \dots, N})$
- 5: **if** no collisions **then**
- 6: $t_{\text{feas}} \leftarrow t_{0, \text{start}}^N$
- 7: **return** A_N is feasible with starting time t_{feas} {Fig. 3}
- 8: **else**
- 9: Allocation block $h \in A_N$ collides with allocation block $k \in A_i$, for some $i \in 1, \dots, N-1$
- 10: $\Delta t \leftarrow t_{k, \text{end}}^i - t_{h, \text{start}}^N$
- 11: $t_{0, \text{start}}^N \leftarrow t_{0, \text{start}}^N + \Delta t$
- 12: **end if**
- 13: **end while**
- 14: **return** A_N is not a feasible allocation {Fig. 2}

blocks, where the k -th block of the n -th traffic stream takes the interval $b_k^n = (t_{k, \text{start}}^n, t_{k, \text{end}}^n)$, where

$$\begin{aligned} t_{k, \text{start}}^n &= t_{0, \text{start}}^n + kT_p^n; \\ t_{k, \text{end}}^n &= t_{0, \text{start}}^n + kT_p^n + T_{\text{blk}}^n; \end{aligned} \quad (1)$$

for $k = 0, 1, 2, \dots$. A graphical example is shown in Fig. 1.

Following this definition we can say that, given N distinct allocations A_1, \dots, A_N , they are jointly periodic over a period

$$T_p^{1, \dots, N} = \text{lcm}(T_p^1, \dots, T_p^N), \quad (2)$$

where lcm indicates the *least common multiple* of the periods. Note that, since all block periods are integer multiples or fractions of T_{BI} , the least common multiple (lcm) can always

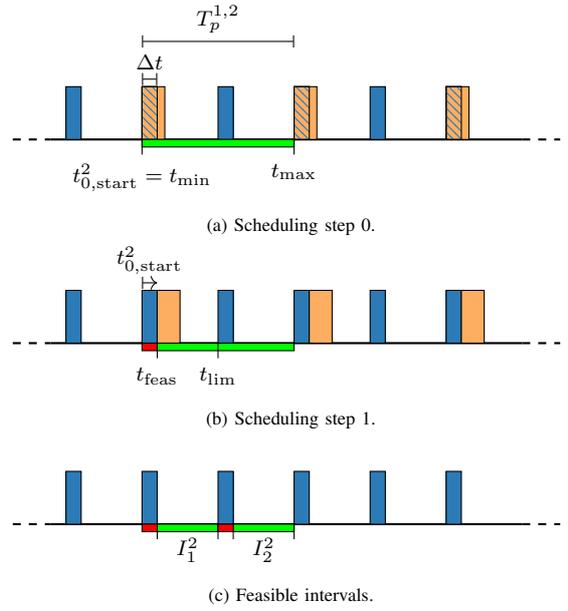


Fig. 3: Feasibility check for a *feasible* pair of allocations, where A_1 (blue) was a pre-existing allocation with $p_1 = \frac{1}{4}$, and the algorithm is checking whether a new allocation A_2 (orange) with $p_2 = \frac{1}{2}$ is compatible.

be properly defined [15] as

$$\text{lcm}\left(\frac{a}{b}, \frac{c}{d}\right) = \frac{\text{lcm}(a, c)}{\text{gcf}(b, d)}, \quad (3)$$

where gcf is the *greatest common factor*.

Given the periodicity of the allocation patterns, a new allocation A_N should start within a time interval T_p^N since the beginning of the BI. Moreover, a necessary requirement for admission is that in an interval of duration $T_p^{1, \dots, N}$, no block b_h^N overlaps with any block b_k^n , $n \in \{1, \dots, N-1\}$, $\forall h, k \geq 0$.

The remainder of this section is structured as follows: in Sec. IV-A we will illustrate an algorithm to efficiently check whether a new allocation is compatible with a pre-existing schedule, in Sec. IV-B we will present a simple scheduling algorithm, and finally in Sec. IV-C we will describe in detail a more complex algorithm that aims at minimizing the rejection of new allocations under the *strict periodicity* assumption.

A. Feasibility Check Algorithm

This feasibility check can be performed as described in Algorithm 1, whose arguments consist of the existing allocations A_1, \dots, A_{N-1} , the new request A_N and a search interval $[t_{\min}, t_{\max}]$. For reasons that will be clear later, we assume that the existing allocations cannot be changed, while for A_N only $t_{0, \text{start}}^N$ can be modified, keeping the period T_p^N and the block duration T_{blk}^N fixed. Based on these input values, the aim of the procedure is to find the earliest feasible starting time t_{feas}^N such that a block of duration T_{blk}^N fits in the search interval.

To do so, starting from t_{\min} , the algorithm progressively shifts the starting time by an interval Δt (described in Al-

gorithm 1) until either all feasibility conditions are met, or $t_{0,\text{end}}^N > t_{\text{max}}$, in which case the allocation A_N with block duration T_{blk}^N is rejected.

A trivial example involves A_1 , i.e., the first received allocation request from a STA. In this case, t_{min} will be set to the start time of the first BI following the reception of the request, while $t_{\text{max}} = T_p^N$ to guarantee the periodicity. Since no previous allocated SPs exist, A_1 is immediately accepted with $t_{\text{feas}} = t_{\text{min}}$. It is important to highlight that, however, by choosing specific combinations of input parameters, Algorithm 1 can be used also by more advanced scheduling schemes, as explained later.

Given any feasible starting time t_{feas} , it is useful to compute the rightmost *boundary* of the allocation, i.e., the largest interval $[t_{\text{feas}}, t_{\text{lim}}]$ that would still make $b_0^N \in [t_{\text{feas}}, t_{\text{lim}}]$ and, in turn, A_N feasible, even for larger values $t_{0,\text{start}}^N$ and T_{blk}^N . This boundary can be computed by finding the minimum distance between each $b_n \in A_N$ and each $b_k \in A_n$, $n \neq N$. The final results will be the minimum measured distance. A graphical illustration of how the algorithm behaves when the new request is infeasible is shown in Fig. 2, while a new feasible request is shown in Fig. 3.

Following this definition and the above numerical example, the first allocation request to be generated, i.e., A_1 , will find itself in the optimal condition where $t_{\text{feas}} = t_{\text{min}}$ and $t_{\text{lim}} = t_{\text{max}}$.

In general, multiple feasible intervals exist. To find an exhaustive list, we can iterate Algorithm 1 with $t_{0,\text{start}}^N$ initialized to the start time of the BI, and progressively updated at each iteration with the value of t_{lim} found in the previous execution. This procedure continues until the shift of $t_{0,\text{start}}^N$ leads to an infeasible allocation. We define the list of feasible intervals (which depend on T_{blk}^N) as $\mathcal{I}_N = \{I_1^N, \dots, I_M^N\}$, where $I_m^N = [t_{m,\text{feas}}^N, t_{m,\text{lim}}^N]$, $m = 1, \dots, M$ (see Fig. 3c). Hence, each of these intervals delimits the finite number of intervals in which the new allocation A_N can be fitted, considering all previous allocations. A good scheduling algorithm should then assess which interval yields the best overall performance, possibly trying to optimize a target Key Performance Indicator (KPI).

B. Simple Scheduler

The first scheduler that we propose assumes that the block duration and periodicity of already accepted traffic streams cannot be varied. Then, a new request A_N with a block duration of $T_{\text{blk}}^N \in [T_{\text{min}}^N, T_{\text{max}}^N]$ can be accepted only if there exists a feasible interval in \mathcal{I}_N with a duration of at least T_{min}^N . Therefore, the maximum amount of available resources that can be allocated to A_N is determined by the longest feasible interval, or by T_{max}^N , whichever is smaller; $t_{0,\text{start}}^N$ and T_{blk}^N need to be set accordingly. We can already notice that, using this simple first-come-first-served approach, the latest requests are highly disadvantaged if the first ones require big slices of time resources. In the long term, as we will see in Sec. V, this could lead not only to poor performance in terms of fairness, but also to a very low admission rate.

C. Max-Min Fair Scheduler

A more flexible approach consists in dynamically adapting the duration of the allocated intervals within the admissible range, $T_{\text{blk}}^n \in [T_{\text{min}}^n, T_{\text{max}}^n]$, $\forall n = 1, \dots, N$, in order to distribute time resources among all traffic streams in a fairer manner.

Consider the following parameterized block duration:

$$T_{\text{blk}}^n(r) = T_{\text{min}}^n + r_n(T_{\text{max}}^n - T_{\text{min}}^n), \quad r_n \in [0, 1]. \quad (4)$$

We consider a scheduler to be fair if $r_{\text{min}} = \min_n \{r_n\}$ cannot be increased without breaking the limits imposed by some allocation under the *strict periodicity* constraint (see Sec. II). The scheduling algorithm, then, should assign the largest possible SP to each allocation, while respecting all the constraints.¹

To fit a new traffic stream, the pre-existing allocations will thus have to either maintain or reduce their block duration, depending on whether and how the new allocation collides with them. This will lead to a lower rejection rate with respect to the *Simple Scheduler* (Sec. IV-B), and more fairness among requests distributed in time.

The proposed algorithm is here presented in two parts: the first part describes how the allocation scheme works (Sec. IV-C1), while the second part describes the fairness paradigm (Sec. IV-C2).

1) *Allocation Algorithm*: Differently from the simple scheduler, this scheduler can change the block duration within the range imposed by the requesting STA, i.e., $T_{\text{blk}}^n \in [T_{\text{min}}^n, T_{\text{max}}^n]$. To reduce the rejection rate, we check the feasibility of a new allocation A_N (Sec. IV-A) by assuming all existing allocations are shrunk to their minimum, i.e., $T_{\text{blk}}^n = T_{\text{min}}^n$ for $n = 1, \dots, N$. If A_N is infeasible even under these conditions, then the allocation cannot be granted without disattending the requests of some previously accepted flow. Therefore, A_N is rejected. Conversely, if A_N is feasible, it gets accepted, and in a later step the algorithm will try to increase the resource utilization of all allocations fairly.

From now on, we use the symbol $*$ to represent the parameter values at the end of the execution of the algorithm. We recall that, based on the strict periodicity assumption, the starting times of the already allocated blocks cannot change

Note that, given a set of feasible allocations, reducing any r_n (and, in turn, the T_{blk}^n) still yields a valid configuration. Similarly, a valid configuration for A_N with $t_{0,\text{start}}^N$ and $r_N \geq 0$ will remain valid if $t_{0,\text{start}}^{N*} \geq t_{0,\text{start}}^N$ and $t_{\text{end}}^{N*} = t_{0,\text{start}}^{N*} + T_{\text{blk}}^N(r_N^*) \leq t_{\text{end}}^N$. We thus consider the following constraints:

$$r_n^* \leq r_n, \quad \forall n \leq N; \quad (5a)$$

$$t_{0,\text{start}}^{N*} \geq t_{0,\text{start}}^N; \quad (5b)$$

$$t_{\text{end}}^{N*} \leq t_{\text{lim}}^N. \quad (5c)$$

The algorithm starts by considering the first feasible interval I_1^N , which ensures a valid configuration when $r_n = 0$,

¹Note that if $T_{\text{min}}^n = T_{\text{max}}^n$, r_n has no meaning. For simplicity, this case has not been included in this study.

Require: $A_1, \dots, A_N, T_p^{1, \dots, N}$

- 1: Compute \mathcal{I}_N considering $T_{\text{blk}}^n = T_{\text{min}}^n \forall n = 1, \dots, N$
- 2: **for all** $I_m^N = [t_{\text{feas}}, t_{\text{lim}}]_m \in \mathcal{I}_N$ **do**
- 3: $t_{0, \text{start}}^N \leftarrow t_{\text{feas}}$
- 4: Set r_N such that $T_{\text{blk}}^N = \min \{T_{\text{max}}^N, t_{\text{lim}} - t_{\text{feas}}\}$ {Eq. (4)}
- 5: **for all** $A_n, n = 1, \dots, N - 1$ **do**
- 6: **for all** (block $k \in A_n$) $\in T_p^{1, \dots, N}$ **do**
- 7: **if** block k collides with A_N **then**
- 8: Update $r_n^*, r_N^*, t_{0, \text{start}}^{N*}$ {Sec. IV-C2}
- 9: **if** $r_n^* < r_n$ **then**
- 10: Add/update A_n to a list \mathcal{C} of colliding allocations
- 11: Memorize $r_{n, \text{prev}} \leftarrow r_n$
- 12: **end if**
- 13: Update $r_n, r_N, t_{0, \text{start}}^N$
- 14: **end if**
- 15: **end for**
- 16: **end for**
- 17: **for all** $A_n \in \mathcal{C}$ **do**
- 18: Compute $\Delta t = t_{\text{lim}} - t_{0, \text{start}}^n$ for A_n given A_N {see Sec. IV}
- 19: $T_{\text{blk}}^n \leftarrow \min \{T_{\text{blk}}^n(r_{n, \text{prev}}), \Delta t\}$ {Try to improve the allocation duration if A_N has been further reduced}
- 20: **end for**
- 21: Compute allocation score $s_m \leftarrow \min_{n=1, \dots, N} r_n$
- 22: **end for**
- 23: **return** The configuration which maximizes the allocation score $\{s_m\}$

$\forall n \leq N$. The new request is temporarily accepted with $t_{0, \text{start}}^N = t_{1, \text{feas}}^N$ and maximum possible r_N , such that $T_{\text{blk}}^N = \min \{T_{\text{max}}^N, t_{\text{lim}} - t_{\text{feas}}\}$.

Then, the algorithms try to re-balance the resource allocation by increasing all $\{r_n, \forall n \leq N\}$ to their previous values. Given that feasible intervals \mathcal{I}_N were computed considering all allocations with minimum duration, though, setting $t_{0, \text{start}}^N = t_{1, \text{feas}}^N$ may (or may not) create a collision with a generic A_i when setting $r_i \geq 0$ back to its previous value.

On the other hand, thanks to the information given by t_{lim} , we can always choose r_N such that the new allocation does not collide with a previous one, on the right.

Collisions can be found iteratively over each block of each previous allocation in a joint period.

If for a certain block $b_k^n \in A_n$ and a block $b_h^N \in A_N$ we have

$$t_{k, \text{start}}^n + T_{\text{blk}}^n(r_n) \geq t_{h, \text{start}}^N \quad (6)$$

then the two allocations are in conflict, as shown in Fig. 4. In this case, $t_{0, \text{start}}^{N*}$, r_N^* , and r_n^* have to be updated following the constraints in (5), as described in Sec. IV-C2.

The constraints from (5), the existence of a non-empty set of feasible intervals, and the iterative nature of the problem ensure that the algorithm will stop in a finite time with a valid configuration. Since each feasible interval $I_m^N \in \mathcal{I}_N$ has one locally fairest configuration, the exhaustive search described in Algorithm 2 is able to find the globally fairest configuration by exhaustive search.

2) *Optimally fair allocation:* In this section, we will discuss how fairness can be achieved given a pair of colliding allocations $A_n, n \in \{1, \dots, N - 1\}$, and A_N . In Sec. IV-C1 we explained how such a collision can be found, e.g. between blocks b_k^n and b_h^N . For the sake of clarity, in this section we

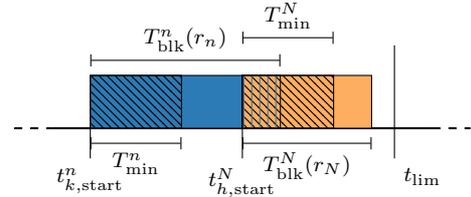


Fig. 4: Representation of a collision between A_n and A_N .

will drop the notation for the specific colliding blocks.

In order to fully exploit the available resources, looking at Fig. 4, we force $t_{0, \text{start}}^N = t_{\text{end}}^n$ and $t_{\text{end}}^N \leq t_{\text{lim}}$. In this way we make A_N start right after A_n , still respecting the limits imposed by t_{lim} .

While possibly not being optimal, this is still a sensible choice for a greedy approach that tries to maximize the fairness of the current configuration. By doing so and imposing $r_n = r_N = r^*$, what we call the *fairness equation*, and by noting that $r^* \leq 1$ should hold, we have that

$$r^* = \min \left\{ 1, \frac{t_{\text{lim}} - t_{0, \text{start}}^n - T_{\text{min}}^n - T_{\text{min}}^N}{(T_{\text{max}}^n - T_{\text{min}}^n) + (T_{\text{max}}^N - T_{\text{min}}^N)} \right\}. \quad (7)$$

We call r^* the *fair allocation ratio*, and note that if $r^* < 1$, it must be that $t_{\text{end}}^N = t_{\text{lim}}$, whereas if $r^* = 1$ in general $t_{\text{end}}^N \leq t_{\text{lim}}$ by construction.

Depending on the initial conditions of the problem, there is a number of different cases which have to be properly managed in order to obtain a fair distribution of resources.

First of all, if $r_n \leq r^*$, following Eq. (5a), it means that previous adjustments do not make it possible for A_n to obtain more resources while still ensuring a valid configuration, and thus $r_n^* = r_n$. Furthermore, since we assume that a collision happens between A_n and A_N with this configuration, A_N has to be delayed setting $t_{0, \text{start}}^{N*} = t_{0, \text{start}}^n + T_{\text{blk}}^n(r_n^*) > t_{0, \text{start}}^N$.

In case also $r_N \leq r^*$, allocation A_N cannot be extended either. Since both allocations have $r_n, r_N \leq r^*$, they will both surely fit in the feasible interval. If, instead, $r_N > r^*$, A_N can obtain $r_N^* \geq r_n^*$, i.e., $T_{\text{blk}}^{N*} = \min \{T_{\text{blk}}^N(r_N), t_{\text{lim}} - t_{0, \text{start}}^{N*}\}$.

On the other hand, if $1 \geq r_n > r^*$, the block duration must be reduced so that $r_n^* = r^*$. Then, if also $r_N > r^*$, both allocations must be trimmed and are fairly allocated, i.e., $r_n^* = r_N^* = r^* < 1$. It follows from the properties of Eq. (7) that $t_{\text{end}}^{N*} = t_{\text{lim}}$ and $t_{0, \text{start}}^{N*} > t_{0, \text{start}}^N$.

Finally, if $r_N \leq r^* < r_n \leq 1$, and therefore $r_N < 1$, the properties of Eq. (7) imply that $t_{\text{end}}^N = t_{\text{lim}}$. Since A_N cannot be extended without possibly reducing the allocation ratio of other allocations, $t_{0, \text{start}}^{N*} = t_{0, \text{start}}^N$ and $r_N^* = r_N$. Since, by assumption, $t_{\text{end}}^N > t_{0, \text{start}}^N$, the duration of A_n needs to be reduced so that $T_{\text{blk}}^{n*} = t_{0, \text{start}}^{N*} - t_{0, \text{start}}^n < T_{\text{blk}}^n$.

V. RESULTS

In this section, we evaluate the algorithms described in Sec. IV. The proposed schedulers have been implemented in Python, only focusing on their capabilities of allocating communication resources to the different traffic streams. The

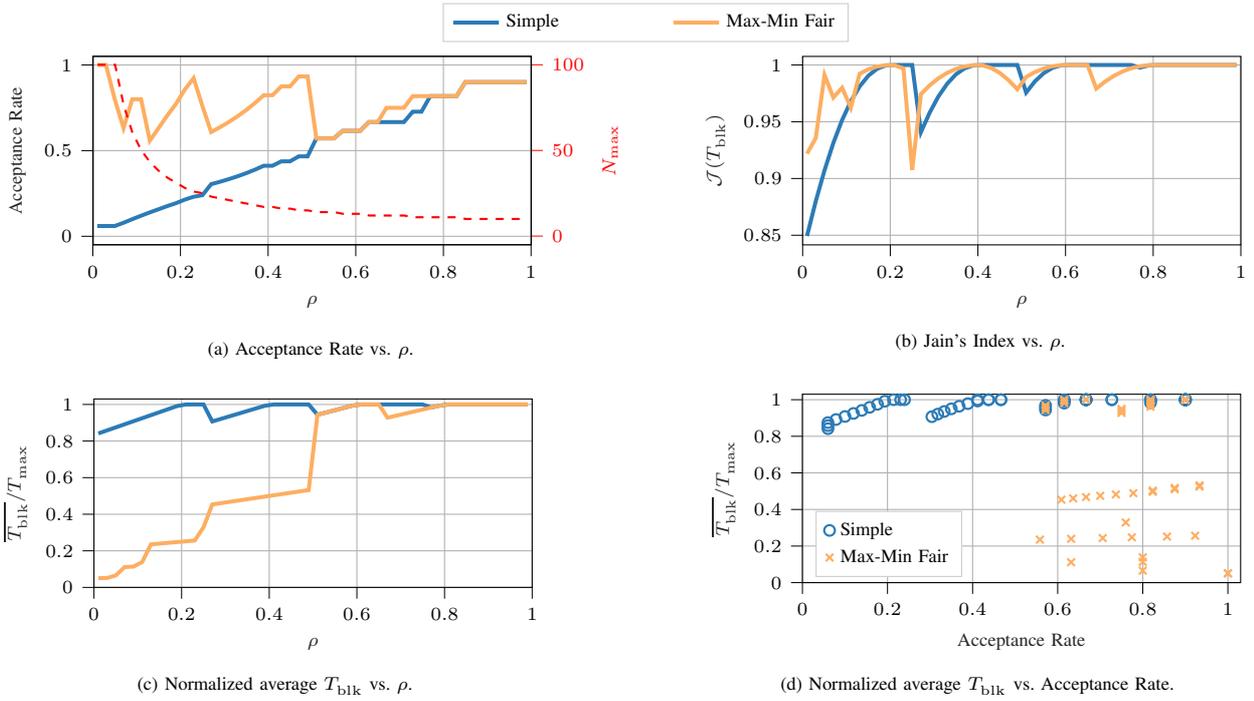


Fig. 5: Results for Scenario 1.

simulations proposed here do not include full stack behaviors, which will be investigated in future works, and their aim is thus to highlight the fundamental characteristics of each algorithm.

Based on their design criteria, we expected the two algorithms to differ mainly with respect to three KPIs, namely the acceptance rate of new requests, the fairness among accepted allocations, and the average scheduled block duration. To highlight these characteristics we shape the offered traffic based on three parameters, namely: the *average allocation request*

$$T_{\text{avg}} = \frac{T_{\text{min}} + T_{\text{max}}}{2}; \quad (8)$$

the *interval ratio*

$$\rho = \frac{T_{\text{min}}}{T_{\text{max}}} \in [0, 1]; \quad (9)$$

and the *load factor*

$$\lambda = \frac{T_{\text{avg}}}{T_p}. \quad (10)$$

Note that, for a given average allocation request T_{avg} , a low interval ratio ρ corresponds to very flexible allocations, while $\rho = 1$ corresponds to rigid allocations where $T_{\text{min}} = T_{\text{max}}$.

The proposed algorithms are compared in two different simulation scenarios:

- *Scenario 1*: all traffic streams are homogeneous, i.e., all requests have the same parameters. Specifically, we consider the case with periodicity $T_p = \frac{T_{\text{BI}}}{3}$, load factor $\lambda = 0.1$, and $\rho \in (0, 1)$. The impact of different periodicities and load factors is also discussed.
- *Scenario 2*: multiple non-homogeneous applications co-

exist on the same network, thus generating traffic streams with different characteristics. We analyze a scenario where traffic streams can be of class C_1 or C_2 , with periodicity $T_p^{C_1} = \frac{T_{\text{BI}}}{3}$ and $T_p^{C_2} = \frac{T_{\text{BI}}}{5}$, respectively. Both classes have load factor $\lambda = 0.1$ and interval ratio $\rho = 0.1$.

To evaluate the performance of the algorithms, we propose three KPIs for *Scenario 1*, shown in Fig. 5. Note that, given the discrete behavior of the problem and the lack of randomness in the proposed algorithms, the plots cannot be smoothed by running multiple repetitions.

The first metric is the acceptance rate (Fig. 5a), defined as the ratio between the number of accepted allocation requests and the maximum number of acceptable requests. To compute this achievable upper bound, since all allocations share the same parameters we ignore the strict periodicity assumption and calculate how many allocations with minimum duration $T_{\text{blk}} = T_{\text{min}}$ can fit in a period T_p , which is equal to $N_{\text{max}}(\rho) = \lfloor \frac{T_p}{T_{\text{min}}(\rho)} \rfloor$ and shown as a red, dashed line. The acceptance rates can thus be normalized in the interval $[0, 1]$, where 1 means that the scheduler reaches the peak acceptance rate. Since as $\rho \rightarrow 0$, $T_{\text{min}} \rightarrow 0$ and thus $N_{\text{max}}(\rho) \rightarrow \infty$, we consider at most 100 allocations.

As expected, the *simple scheduler* suffers from a lower acceptance rate than the *max-min fair* one, even though, starting from $\rho = 0.5$, the two algorithms tend to behave similarly. In fact, more rigid allocations do not give enough flexibility to the *max-min fair scheduler* to perform its optimization, thus yielding similar performance to the much *simple scheduler*.

The second metric is *Jain's Fairness Index* (Fig. 5b), defined

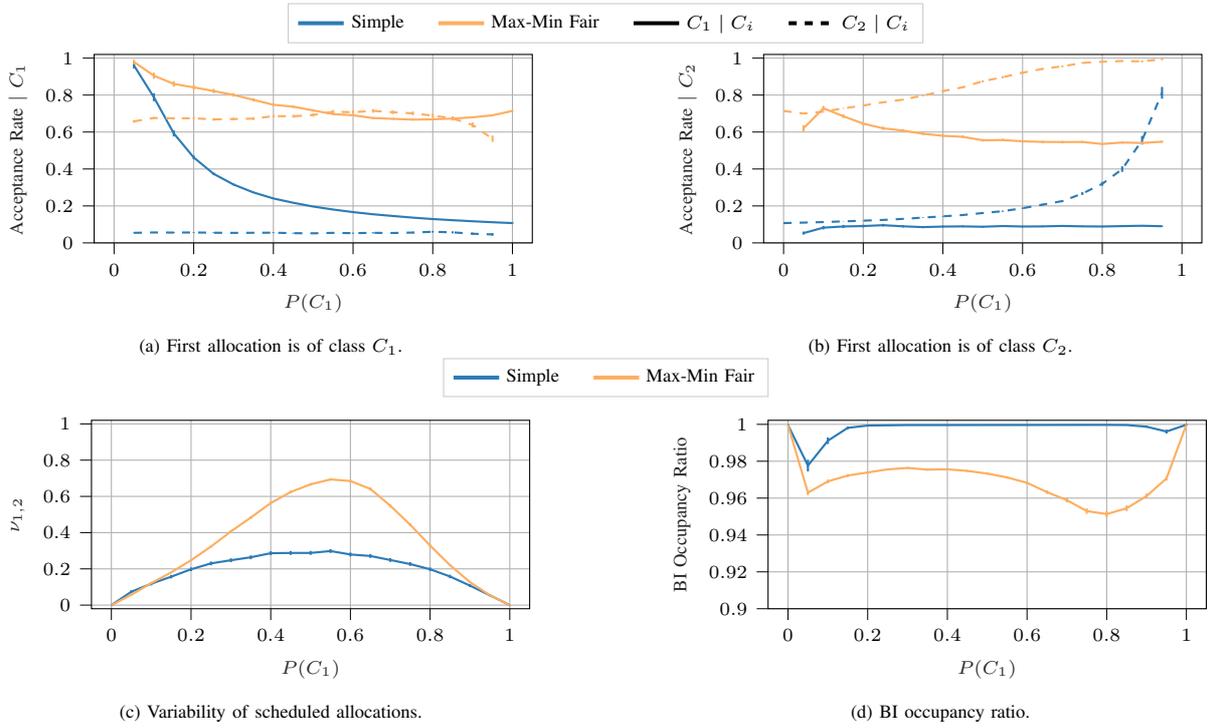


Fig. 6: Results for Scenario 2.

as:

$$\mathcal{J} = \frac{(\sum_n x_n)^2}{n \cdot \sum_n (x_n)^2}, \quad (11)$$

where only accepted allocations are counted and x_n can either be the block duration T_{blk}^n or the block duration ratio r_n . If the $\{x_n\}$ are all equal, then $\mathcal{J}(x) = 1$. On the other hand, the more unequal the values of $\{x_n\}$, the closer the metric to its minimum $\mathcal{J}(x) = \frac{1}{N}$.

Based on the results plotted in Fig. 5b, both algorithms behave fairly with respect to the accepted allocations.

The third metric, shown in Fig. 5c, offers a different perspective considering the average normalized block duration $\overline{T_{\text{blk}}}/T_{\text{max}}(\rho)$. As expected, the *simple scheduler* shows an oscillating trend, due to the discrete behavior of the allocations. In fact, the last scheduled allocation will only reduce T_{blk} down to $T_{\text{min}}(\rho)$, thus, if the portion of DTI left by the previous allocations is less than $T_{\text{min}}(\rho)$, no additional allocations can be fitted. On the other hand, the *max-min fair scheduler* will try to reduce all allocations up to their minimum duration in order to avoid rejecting new ones, granting more accepted allocations at the cost of an overall lower block duration.

Finally, the two most discriminating metrics, namely the average normalized block duration and the acceptance rate, are plotted against each other in Fig. 5d. In general, the *simple scheduler* tends to favor higher average block duration for a lower acceptance rate, while the *max-min fair scheduler* tends to favor acceptance rate at the cost of a lower average block duration, as expected. Both algorithms are able to ensure high fairness to the accepted allocations, generally well above 0.85.

Similar behaviors were also observed for load factors $\lambda \in \{0.025, 0.4\}$, not shown here. As expected, higher loads tend to have more pronounced variability in both the average block duration and the fairness granted to the accepted allocations. Curiously, regardless of the load factor, for values of the interval ratio larger than $\rho \approx 0.5$, the two algorithms tend to have very similar performance due to the more rigid allocation requests that do not allow the *max-min fair scheduler* to exploit its agility.

Scenario 2 allows us to analyze the impact of allocations with different periodicities on the overall network performance, as a function of the probability $P(C_1)$ that a request C_1 is offered to the system.

Since allocations with different periods coexist, it is mandatory to decide how many allocations should be offered to the schedulers, as this will affect how the acceptance rate is normalized. We define the *minimum occupancy* as the minimum BI occupancy ratio of the allocation of category C_i , namely $O_{\text{min}}^i = T_{\text{min}}^i/T_p^i$. Allocations are offered to the schedulers as long as the cumulative minimum offered occupancy does not exceed the value of 1.

In Figs. 6a and 6b we show the biasing effect of the first accepted allocation on the proposed schedulers. Clearly, the *simple scheduler* suffers a strong and symmetric effect, meaning that once the first allocation is scheduled with maximum duration, it will be harder for subsequent allocations with a different period to fit the constrained BI, making the scheduler favor allocations with the same period. On the other hand, it is significantly harder to interpret the behavior of the more complex *max-min fair scheduler*. From further results, not

shown here for lack of space, it is possible to notice that allocations with a lower average BI occupancy are favored, with a slight preference towards those with lower values of T_p and T_{\min} . As also shown here, in fact, allocations with lower values of T_p , such as C_2 with respect to C_1 , tend to fragment the BI more, making it harder to then fit allocations with different periodicity and higher T_{\min} .

To further confirm this biasing behavior, we show the variability ν among the scheduled allocations, defined as

$$\nu_{1,2} = \frac{\min\{|C_1|, |C_2|\}}{\max\{|C_1|, |C_2|\}}, \quad (12)$$

where $|C_i|$ represents the number of accepted allocations of category C_i . This metric takes values in $[0, 1]$, where a value of 0 means that only allocations of a single type have been accepted, while a value of 1 means that the same number of allocations of both categories have been accepted. Results are shown in Fig. 6c, which confirms that the *simple scheduler* favors a more homogeneous BI allocation, while the *max-min fair scheduler* shows once again more flexibility, being able to accommodate fairly requests from different classes, as shown by the higher variability of the accepted allocations.

Finally, we studied how efficiently the two algorithms are able to use the radio resources by measuring the BI occupancy ratio, i.e., the ratio between scheduled and unscheduled air time. It can be noticed that, while the *simple scheduler* accepts fewer requests, it is able to use almost all available resources. This is due to the fact that the scheduler tends to accept homogeneous allocations, allowing them to be packed more efficiently in the BI. On the other hand, the *max-min fair scheduler* successfully fits multiple allocations of both types, but the constraints on the periodicity and the minimum block duration T_{\min} prevent it from fully utilizing the whole BI when a mixture of the two types of sources is presented. Nonetheless, it ensures very high occupancy ratios, always above 95% for the shown example.

VI. CONCLUSIONS

In this paper, we presented a framework for periodic scheduling in WiGig-compatible devices. We proposed two heuristic algorithms, *simple* and *max-min fair schedulers*, and accurately described their inner workings. Finally, we assessed their performance in two different scenarios, showing that the *max-min fair scheduler* tends to trade resource availability for a much higher acceptance rate, contrary to the *simple scheduler's* behavior, while both schedulers obtained a high Jain's fairness index for the accepted allocations.

Even working in a simplified settings without considering further sources of complexity from other parts of the communication stack, it was possible to notice that both the design and the evaluation of WiGig-specific scheduling algorithms for periodic sources is highly non-trivial and can show surprising results. In fact, while the formalization of the problem is straightforward, scheduling algorithms often have to deal with many hard-to-predict edge cases, which greatly increases the difficulty of designing a general algorithm.

Future works will focus on multiple objectives. A first objective is to implement these algorithms in a full-stack simulator, allowing the study of APP-layer performance metrics for a range of possible applications. A second objective is to extend the framework by relaxing some of the assumptions made in Sec. II and test the impact of each one of them on the overall performance of a WiGig system. A third objective is to extend the study for scheduling multiple allocations at once, a possibility given by MIMO techniques. To further achieve a realistic evaluation, the impact of accurate wireless channel simulations, physical layer design, as well as user mobility will also be studied.

REFERENCES

- [1] Task Group ay, "TGay usage model," Nov. 2017, doc.: 802.11-15/0625r7. [Online]. Available: <https://mentor.ieee.org/802.11/dcn/15/11-15-0625-07-00ay-ieee-802-11-tgay-usage-scenarios.pptx>
- [2] Y. Ghasempour, C. R. C. M. da Silva, C. Cordeiro, and E. W. Knightly, "IEEE 802.11ay: Next-Generation 60 GHz Communication for 100 Gb/s Wi-Fi," *IEEE Communications Magazine*, vol. 55, no. 12, pp. 186–192, Dec. 2017.
- [3] I. P802.11, *IEEE Standard for Information technology – Telecommunications and information exchange between systems Local and metropolitan area networks – Specific requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, Std., Rev. IEEE Std 802.11-2012, Dec. 2016.
- [4] S. Mohebi, M. Lecci, A. Zanella, and M. Zorzi, "The challenges of Scheduling and Resource Allocation in IEEE 802.11ad/ay," in *18th Mediterranean Communication and Computer Networking Conference (MedComNet)*, Jun. 2020.
- [5] F. Babich and M. Comisso, "Throughput and delay analysis of 802.11-based wireless networks using smart and directional antennas," *IEEE Transactions on Communications*, vol. 57, no. 5, pp. 1413–1423, May 2009.
- [6] G. Bianchi, "Performance Analysis of the IEEE 802.11 Distributed Coordination Function," *IEEE Journal on Selected Areas in Communications*, vol. 18, no. 3, pp. 535–547, Mar. 2000.
- [7] C. Pielli, T. Ropitault, N. Golmie, and M. Zorzi, "An Analytical Model for CBAP Allocations in IEEE 802.11ad," *IEEE Transactions of Communications*, vol. 69, no. 1, pp. 649–663, Jan. 2021.
- [8] C. Hemanth and T. G. Venkatesh, "Performance Analysis of Contention-Based Access Periods and Service Periods of 802.11ad Hybrid Medium Access Control," *IET Networks*, vol. 3, no. 3, pp. 193–203, Sep. 2014.
- [9] M. U. Rajan and A. Babu, "Saturation Throughput Analysis of IEEE 802.11ad Wireless LAN in the Contention Based Access Period (CBAP)," in *IEEE Distributed Computing, VLSI, Electrical Circuits and Robotics (DISCOVER)*, Aug. 2016, pp. 41–46.
- [10] E. Khorov, A. Ivanov, A. Lyakhov, and V. Zankin, "Mathematical Model for Scheduling in IEEE 802.11ad Networks," in *IFIP Wireless and Mobile Networking Conference (WMNC)*, Jul. 2016, pp. 153–160.
- [11] C. L. Liu and J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment," *Journal of the ACM*, vol. 20, no. 1, pp. 46–61, Jan. 1973.
- [12] K. Ramamritham, "Allocation and Scheduling of Precedence-Related Periodic Tasks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 6, no. 4, pp. 412–420, Apr. 1995.
- [13] Sheng-Tzong Cheng and A. K. Agrawala, "Allocation and Scheduling of Real-Time Periodic Tasks with Relative Timing Constraints," in *International Workshop on Real-Time Computing Systems and Applications (RTCSA)*, Oct. 1995, pp. 210–217.
- [14] Dakai Zhu, D. Mosse, and R. Melhem, "Multiple-Resource Periodic Scheduling Problem: How Much Fairness is Necessary?" in *IEEE Real-Time Systems Symposium (RTSS)*, Dec. 2003, pp. 142–151.
- [15] R. Zazkis and J. Truman, "From Trigonometry to Number Theory... and Back: Extending LCM to Rational Numbers," *Digital Experiences in Mathematics Education*, vol. 1, no. 1, pp. 79–86, Apr. 2015.