An Admission Control Algorithm for Isochronous and Asynchronous Traffic in IEEE 802.11ad MAC

Anirudha Sahoo
Communications Technology Laboratory,
National Institute of Standards and Technology, Gaithersburg, Maryland, USA
Email: anirud@nist.gov

Abstract—**Due to availability of large amount of bandwidth in the 60 GHz band and support of contention-free channel access called *Service Period* (SP), the IEEE 802.11ad/ay Wi-Fi standard is well suited for low latency and high data rate applications. IEEE 802.11ad supports two types of SP user traffic: *isochronous* and *asynchronous*. These user traffic need guaranteed SP duration before their respective deadlines. Hence, admission control plays an important role in an IEEE 802.11ad system. In an earlier work, we studied admission control and scheduling of isochronous and asynchronous traffic in an IEEE 802.11ad system, but we assumed the asynchronous requests to be periodic to keep the algorithm simple. That assumption resulted in overallocation of resource and potential degradation of performance. In this paper, we present an admission control algorithm which does not make such assumption and yet still maintains a linear run time complexity and allocates resources to the requests in a proportional fair manner. We provide arguments to establish correctness of the algorithm in terms of guaranteeing SP allocation to the requests before their respective deadlines.**

Index Terms—**admission control, scheduling, IEEE 802.11ad, MAC, isochronous, asynchronous.**

I. Introduction

A large amount of unlicensed bandwidth is available in the millimeter wave (mmWave) 60 GHz band. In recent times, the use of high data rate and low latency applications such as 8K TV, Virtual Reality (VR) and Augmented Reality (AR) are on the rise. These factors have led to development of standards for the next generation Wi-Fi such as IEEE 802.11ad and its successor IEEE 802.11ay. In terms of providing high data rate and low latency, the IEEE 802.11ad Medium Access Control (MAC) plays an important role. The IEEE 802.11ad MAC supports contention-free channel access between a pair of stations, called *service period* (SP). It handles two types of SP traffic: *isochronous* and *asynchronous*. Isochronous traffic needs guaranteed allocation of certain minimum *service period* (SP) duration in every period. Asynchronous traffic, on the other hand, requires one time guaranteed allocation of SP before its deadline. So, every admitted isochronous and asynchronous request must be guaranteed its requested SP duration before its deadline. These two types of traffic are requested using *Add Traffic Stream* (ADDTS) request[1], which carries the traffic parameters [1]. Due to the stringent allocation and deadline requirements, admission control and scheduling of requests is an important component of an IEEE 802.11ad system. We have studied admission control and scheduling algorithms for only isochronous requests in an IEEE 802.11ad system [2], [3]. Further, in [4], we presented admission control and scheduling of both isochronous and asynchronous traffic. However, in that work, we treat asynchronous traffic as periodic traffic, which allows us to use the simple admissibility criteria used in *Earliest Deadline First* (EDF) scheduling for periodic Central Processing Unit (CPU) tasks in a realtime system [5]. However, this assumption results in overallocation of resources to asynchronous requests, leading to potential performance loss in terms of admitting fewer requests. In this paper, we present an admission control algorithm called, Efficient Admission Control for Isochronous and Asynchronous Requests (EACIAR), which does not make such assumption and yet still maintains a linear run time complexity as well as allocates resources in a proportional fair manner. Periodic and aperiodic CPU tasks are quite similar to isochronous and asynchronous traffic respectively. Admission control and scheduling of periodic and aperiodic CPU tasks have been studied in the literature [5]–[8]. But some of them have severe limitations, whereas some others cannot be directly applied to IEEE 802.11ad MAC due to difference in traffic description (please refer to Section IV for more detailed discussion). We present the detailed admission control algorithm along with the associated scheduling. We provide arguments to establish correctness of the algorithm in the sense that allocation requirement of every admitted request is met before its deadline. To the best of our knowledge, this is the first comprehensive admission control and scheduling algorithm for IEEE 802.11ad MAC without any restrictions on the request type or on the traffic parameter values of the requests.

II. A Primer on IEEE 802.11ad MAC

A. IEEE 802.11ad Medium Access

The medium access duration in IEEE 802.11ad is divided into an infinite sequence of time intervals, called *Beacon Interval* (BI). The length of a BI duration is specified in Time Units (TU), where $1\,\mathrm{TU} = 1024\,\mu$s. In this paper, we represent a BI duration as a sequence of $1\,\mu$s time slots. A BI has two parts: a Beacon Header Interval (BHI) followed by a Data Transmission Interval (DTI). The DTI is primarily used for data transmission among IEEE 802.11ad Stations (STAs) and Personal Basic Service Set (PBSS) Control Point/Access Point (PCP/AP). An IEEE 802.11ad station can access the channel in a DTI using Contention Based Access Period (CBAP) or Service Period (SP) mechanism. When using CBAP, a station uses a contention based scheme called Enhanced Distributed Channel Access (EDCA) [1]. An SP, on the other hand, is used

[1]Although asynchronous traffic can be requested by other means, in this paper, we only consider ADDTS based method.

between two stations or between a station and its PCP/AP to have a contention free channel access. Before a DTI period starts, the schedules of CBAP and SP in a DTI are broadcast by the PCP/AP to its stations in a Directional Multi Gigabit (DMG) Beacon frame in the Beacon Transmission Interval (BTI) or in the Announce frame in the Announcement Transmission Interval (ATI) of the BHI [1].

B. IEEE802.11ad Traffic Parameters

The Traffic Specification (TSpec) element in the ADDTS request carries the traffic parameters for which resources need to be allocated. For isochronous requests, channel time allocation duration is repeated in every period, whereas for asynchronous traffic, allocation is one time. The main traffic parameters (used in the DMG TSpec frame) of isochronous traffic are [1], [2]:

- Allocation Period (P): Period over which allocation repeats. It can be an integer multiple or integer fraction of a BI.
- Minimum Allocation (C_{min}): Minimum acceptable allocation in microseconds in each allocation period. If the request is accepted, the PCP/AP must guarantee at least this duration to the STA in every allocation period.
- Maximum Allocation (C_{max}): Requested allocation in microseconds in each allocation period. This is the maximum duration that can be allocated to the user in each allocation period.

Asynchronous requests also use the same DMG TSpec parameters. However, the minimum allocation is one time allocation that must be done before the allocation period (or *deadline*). Maximum allocation field is reserved.

III. Admission Control

A. Isochronous and Asynchronous Traffic Model

A request T_i is an isochronous (resp. asynchronous) request when $T_i.reqType$ is *ISO* (resp. *ASYNC*). For the ease of notation, we will use those two terms throughout this paper to refer to the respective requests. In addition, we will use *ISO$_M$* and *ISO$_F$* for ISO requests whose periods are integer multiple of a BI and integer fraction of a BI, respectively. The period, minimum allocation, and maximum allocation of an isochronous request T_i are denoted by $T_i.P$, $T_i.C_{min}$ and $T_i.C_{max}$, respectively. The actual (or operational) channel time allocation to isochronous request T_i is denoted as $T_i.C_{op}$ and should satisfy $T_i.C_{min} \leq T_i.C_{op} \leq T_i.C_{max}$. When there are no ASYNC requests in the system, an ISO request T_i would be allocated $T_i.C_{op}$. However, when there is at least one ASYNC request in the system, the operational channel time of an ISO request is decided based on the schedule carried in *long_schedule[]* (see Algorithm 1 for its computation). The period (or deadline) and minimum allocation of asynchronous request T_i are denoted by $T_i.P$ and $T_i.C_{min}$, respectively. The operational allocation $T_i.C_{op}$, in this case, is equal to $T_i.C_{min}$. An isochronous request has allocation request in every period. Such allocation requests are referred to as *jobs* of the request. Since an *ISO$_F$* request has multiple periods in a BI, it will have multiple jobs in a BI. An ISO_M request, on the other hand, will have one job in multiple BIs. Since an asynchronous request does not repeat beyond its deadline, it has exactly one job in its lifetime. A job of a request becomes ready to be allocated at the beginning of its period, which we refer to as the *release time* of the job. A job of a request may be allocated one contiguous block or it may be broken into multiple *fragments* to fit smaller idle durations in a BI.

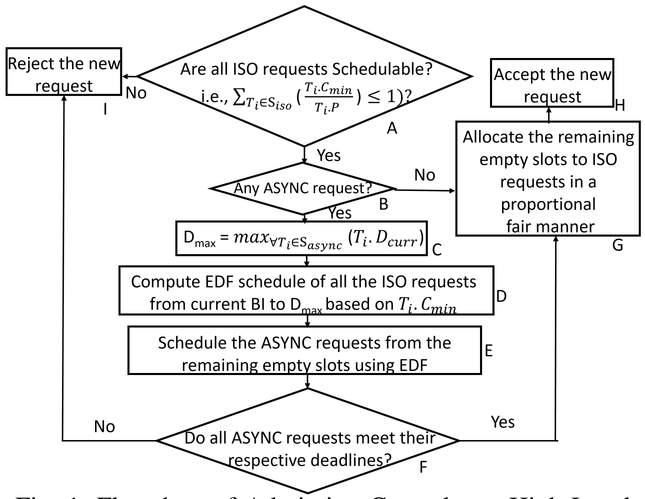
Are all ISO requests Schedulable?
i.e., $\sum_{T_i \in S_{iso}} (\frac{T_i.C_{min}}{T_i.P}) \leq 1)$?
A
Reject the new request
I
No
Yes
Any ASYNC request?
B
No
Accept the new request
H
Allocate the remaining empty slots to ISO requests in a proportional fair manner
G
Yes
D$_{max}$ = $max_{\forall T_i \in S_{async}} (T_i.D_{curr})$
C
Compute EDF schedule of all the ISO requests from current BI to D$_{max}$ based on $T_i.C_{min}$
D
Schedule the ASYNC requests from the remaining empty slots using EDF
E
No
Do all ASYNC requests meet their respective deadlines?
F
Yes

Fig. 1: Flowchart of Admission Control at a High Level

B. CPU Scheduling of Periodic Tasks

ISO requests are periodic and hence, have similarity with periodic tasks in CPU scheduling which has been studied in the literature [5]–[8]. So, we design scheduling of ISO request based on theories developed for CPU scheduling of periodic tasks. A periodic task in CPU scheduling has two parameters *(C_i, P_i)*, where C_i is the duration of the task and P_i is the period as well as the deadline of the task [4]. The feasibility or admissibility of a set of n preemptive periodic tasks for an EDF scheduler is given by [5]

$$\sum_{i=1}^{n} \frac{C_i}{P_i} \leq 1. \tag{1}$$

C. Admission Control at a High Level

The overall admission control flowchart is shown in Fig. 1. The basic idea is to check whether each request (both existing and the new request) is schedulable before its deadline, when a new request arrives. It first checks if all the ISO requests are schedulable or not based on their respective C_{min} demand (Decision Box A). Notice that the Decision Box A utilizes admissibility criteria of periodic CPU tasks presented in Eq. (1) based on $T_i.C_{min}$. If all the ISO requests are not schedulable, then the new request is rejected (Box I). Otherwise, it checks if there is any ASYNC request in the system (Decision Box B). If not, then it allocates the remaining empty slots to the ISO requests in a proportional fair manner (Box G) and accepts the new request (Box H). Otherwise, it computes D_{max}, which is the maximum over all the current deadlines of ASYNC requests (Box C). Then it computes EDF schedule of all the ISO requests from current BI to D_{max} (Box D). ASYNC requests are then scheduled from the remaining empty slots using EDF (Box E). If all the ASYNC requests meet their respective deadlines (decision Box F), then any remaining empty slots are allocated to ISO requests in a proportional fair manner (Box G) and the new request is accepted (Box H). Otherwise the new request is rejected (Box I).

TABLE I: Request Table Entry

Parameter	Unit	Description
reqType	'ISO' or 'ASYNC'	set to 'ISO' for isochronous request or 'ASYNC for asynchronous requests
C_{min}, C_{max}	μs	C_{min} and C_{max} of the request. C_{max} is not relevant for ASYNC requests
P	BI	period of Request, if *reqType* is ISO. Original deadline of the request if *reqType* is ASYNC.
C_{op}	μs	operational channel time allocation. Only relevant for ISO requests.
C_{remain}	μs	remaining channel time allocation still needed before its current deadline to satisfy C_{min}. Only relevant for ASYNC requests and *ISO$_M$* requests.
t_{remain_life}	BI	time remaining for this request to leave the system
D_{curr}	BI	current deadline of the request. Only relevant for ASYNC requests and *ISO$_M$* requests.

D. Maintenance of Information and Operation in every BI

For admission control and scheduling purposes, a table for each admitted request is maintained. The content of the table is shown in Table I. In addition, two global variables, $\mathcal{S}_{iso}$ and $\mathcal{S}_{async}$ which carries the set of existing ISO and ASYNC requests respectively, are maintained. Once the schedule for the current BI is computed and executed, the parameters of the table are updated as follows:

- For each *ISO$_M$* request and each ASYNC request, C_{remain} is decremented by the duration (in μs) allocated to the request in the current BI. Note that, for an *ISO$_M$* request, its C_{remain} can become less than zero (which means the request has been allocated more than its C_{min}), since it may be allocated more than C_{min} in a given period. *ISO$_F$* requests are fully allocated within a BI and hence, C_{remain} is not relevant for them.
- For all requests, t_{remain_life} is decremented by one. If it becomes zero, it signifies that the request is finished. Hence, that request is removed from the set $\mathcal{S}_{iso}$ or $\mathcal{S}_{async}$ depending on its reqType and the corresponding table entry is removed. At this point, if there are no ASYNC requests, then an EDF schedule is followed. Otherwise, admission control algorithm is executed again to recalculate *long_schedule[]* which is followed from the next BI.
- For each *ISO$_M$* request and ASYNC request, its D_{curr} is decremented by one. For an *ISO$_M$* request, if its D_{curr} becomes zero and t_{remain_life} is greater than zero, then C_{remain} is reset to C_{min} and D_{curr} is reset to P. Note that for AYSNC requests when D_{curr} becomes zero, t_{remain_life} should also become zero.

E. Detailed Algorithm

When a new request arrives to the system, the admission control algorithm called *Efficient Admission Control for Isochronous and Asynchronous Requests* (EACIAR), presented in Algorithm 1, is executed. If the total utilization of all the ISO requests based on their $T_i.C_{min}$ is more than one (based on Eq. (1), then the new request is rejected (Line 5). If there is no ASYN request in the system, then it allocates any surplus duration to the ISO requests in a proportional fair manner (Line 6 to 11) and accepts the new request. Otherwise, it starts scheduling from current BI till D_{max}, which is the maximum of current deadline of all the ASYNC requests. First, it takes up scheduling of *ISO$_F$* requests in the order of non-decreasing deadlines. For each *ISO$_F$* request T_i, it allocates $T_i.C_{min}$ (see Line 21). Then, it takes up scheduling of each *ISO$_M$* request T_i in the order of non-decreasing deadlines and allocates $T_i.C_{remain}$ (see Line 34). Then finally, it schedules ASYNC requests, in the non-decreasing order of their current deadlines and allocates $T_i.C_{remain}$ for each ASYNC request T_i (see Line 59). Then it checks for any surplus duration (over C_{min}) available (Line 67 to Line 72). If so, it distributes the surplus duration among the jobs of ISO requests, whose release time falls within the *long_schedule[]* interval, in a proportional fair manner (Line 77). It then goes back in *long_schedule[]* and allocates as much of the surplus duration as possible, to each job of a given ISO request before their respective deadlines (Line 75 to Line 78). Note that for some ISO requests, it may not be possible to allocate all of its share of surplus duration to all of their jobs before their deadlines. This is because the ASYNC requests in the system may have taken up some slots where these surplus slots could have been allocated. Also note that the order of handling of the type of requests is important, i.e., first the *ISO$_F$* requests should be handled followed by *ISO$_M$* requests and finally, ASYNC requests. This is to honor priority of ISO requests over ASYNC requests[2] and to handle ISO requests in non-decreasing order of their respective deadlines.

When an existing request leaves the system, the same Algorithm 1 is used, but instead of executing it from the beginning, it starts executing from Line 6 and continues till the end[3].

When there is at least one ASYNC request in the system, the schedule in *long_schedule[]* is followed. But when there is no ASYNC request in the system, EDF schedule is followed. In this case, the allocation amount is decided as follows. When the system transitions from having at least one ASYNC request to having no ASYNC request, for each *ISO$_M$* request whose C_{remain} is greater than zero, it would be allocated C_{remain} by following EDF schedule. Once, its C_{remain} becomes zero, then it becomes eligible to get allocation of C_{op} in its next period. If C_{remain} of an *ISO$_M$* is less than or equal to zero, then the request becomes eligible for allocation of C_{op} from its next period. Each *ISO$_F$* request also becomes eligible for allocation of its C_{op} immediately. Note that the schedule changes only when a new request arrives or an existing request leaves the system.

Time complexity of our EACIAR algorithm can be determined as follows. Every request goes through BI number of time slots to compute its schedule (e.g., see the *while* loop at Line 21). Due to the *for* loop at Line 14, there are D_{max} number of BI's over which each request computes its

[2]Typically ISO requests represent applications which are more critical than those represented by ASYNC requests.

[3]This variation in execution between *arrival of a new request* and *departure of an existing request* could have been handled by distinguishing between them at the beginning of the algorithm and bypassing relevant code for the departure case. But we chose not to clutter the algorithm with such minor details.

Algorithm 1: Efficient Admission Control for Isochronous and Asynchronous Requests (EACIAR)

```
Input: the new request $T_n$, the set of existing isochronous requests $\mathcal{S}_{iso}$ and the set of existing asynchronous requests $\mathcal{S}_{async}$
Output: ACCEPT or REJECT; if no ASYNC request present and the new request is accepted, $C_{op}$ of every ISO request; if ASYNC
        request present and the new request is accepted, $long\_schedule[]$
Begin
    if ($T_n.reqType$ == ISO) then $\mathcal{S}_{iso} = \mathcal{S}_{iso} \cup \{T_n\}$ ;
    else $\mathcal{S}_{async} = \mathcal{S}_{async} \cup \{T_n\}$ ;
    $U^n_{min} = \Sigma_{T_i \in \mathcal{S}_{iso}} (\frac{T_i \cdot C_{min}}{T_i \cdot P})$ ;
    if ($U^n_{min} > 1$) then go to RJ ;
    if ($\mathcal{S}_{async} == \emptyset$) then // no ASYNC requests and all ISO requests are schedulable
        $U_{surplus} = 1 - U^n_{min}$ ;
        $\Delta u_{tot} = \Sigma_{T_i \in \mathcal{S}_{iso}} (\frac{T_i \cdot C_{max} - T_i \cdot C_{min}}{T_i \cdot P})$ ;
        for each $T_i \in \mathcal{S}_{iso}$ do
            $T_i.C_{op} = T_i.C_{min} + \min(1, \frac{U_{surplus}}{\Delta u_{tot}}) \cdot (T_i.C_{max} - T_i.C_{min})$; // proportional fair allocation of
              surplus
        return ACCEPT, (for each $T_i \in \mathcal{S}_{iso}$ $(T_i.C_{op})$) ;
    $\mathcal{S}^{temp}_{iso} = \mathcal{S}_{iso}$ ; $D_{max} = \max_{T_i \in \mathcal{S}_{async}} (T_i.D_{curr})$ ;
    long_sched[$D_{max}$ * BI] = 0 ;
    for $i$ = 1 to $D_{max}$ do
        $Schedule[1 \dots BI] = 0$ ;
        // process ISO requests whose periods are integer fraction of a BI
        $J^{iso}_{frac}$ = {array of jobs of ISO requests in $\mathcal{S}^{temp}_{iso}$ in the current BI whose periods are integer fraction of a BI} ;
        $J^{iso}_{sort\_frac}$ = {sorted array of jobs in $J^{iso}_{frac}$ in a non-decreasing order of their current deadlines} ;
        k = 0 ;
        while ($k <$ len($J^{iso}_{sort\_frac}$)) do
            R = $J^{iso}_{sort\_frac}[k].release$; D = $J^{iso}_{sort\_frac}[k].P$; $C_{min}$ = $J^{iso}_{sort\_frac}[k].C_{min}$ ;
            while ($C_{min} > 0$) do
                $avail$ = number of contiguous slots available from position 'R' in Schedule[] ;
                $avail$ = $\min(avail, C_{min})$ ;
                $Schedule[R \dots (R + avail] = 1$ ;
                $C_{min} -= avail$ ;
                R = next empty slot position in Schedule[] ;
                if ($R > D$) then print error and exit; // this should not happen, since we have done the
                  utilization test for ISO requests ;
            $k^{++}$ ;
        // process ISO requests whose periods are integer multiple of a BI
        $J^{iso}_{mult}$ = {array of jobs of ISO requests in $\mathcal{S}^{temp}_{iso}$ in the current BI whose periods are integer multiple of a BI} ;
        $J^{iso}_{sort\_mult}$ = {sorted array of jobs in $J^{iso}_{mult}$ in a non-decreasing order of their current deadlines} ;
        k = 0 ;
        while ($k <$ len($J^{iso}_{sort\_mult}$)) do
            R = 1 ; D = $J^{iso}_{sort\_mult}[k].D_{curr}$ ; $C_{remain}$ = $J^{iso}_{sort\_mult}[k].C_{remain}$ ;
            while ($C_{remain} > 0$) do
                $avail$ = number of contiguous slots available from position 'R' in Schedule[] ;
                $avail$ = $\min(avail, C_{remain})$ ;
                $Schedule[R \dots (R + avail] = 1$ ;
                $C_{remain} -= avail$ ;
                R = next empty slot position in Schedule[] or (-1) if Schedule[] is full (all 1);
                if ($R == -1$) then print error and exit; // this should not happen, since we have done the
                  utilization test for ISO requests ;
            $k^{++}$ ;
        long_schedule[i] = schedule[] ;
        for each $T_i \in \mathcal{S}^{temp}_{iso}$ do
            if ($T_i.t_{remaining\_life} == i$) then $\mathcal{S}^{temp}_{iso} = \mathcal{S}^{temp}_{iso} \setminus \{T_i\}$ ; // Request $T_i$ leaves the system
        update current BI to the next BI ;
    tot_async_dur = 0 ;
    for each $T_i \in \mathcal{S}_{async}$ do
        tot_async_dur += $T_i.C_{remain}$ ;
    num_empty_slots = number of empty slots in long_schedule[] ;
    if (tot_async_dur > num_empty_slots) then // not enough empty slots to schedule all the ASYNC
      requests
        Goto RJ ;
    // process ASYNC requests
    $\mathcal{S}_{sort}^{async}$ = {sorted array of ASYNC requests in a non-decreasing order of their current deadlines} ;
    k = 0; tot_c_async = 0 ;
    while ($k <$ len($S_{sort}^{aysnc}$)) do
        R = 1 ;
        D = $\mathcal{S}_{sort}^{async}[k].D_{curr}$ ;
        $C_{remain} = \mathcal{S}_{sort}^{async}[k].C_{remain}$ ;
        while ($C_{remain} > 0$) do
            $avail$ = number of contiguous slots available from position 'R' in Schedule[] ;
            $avail = \min(avail, C_{remain})$ ;
            $long\_schedule[R \ldots (R + avail] = 1$ ;
            $C_{remain} -= avail$ ;
            R = next empty slot position in long_schedule[];
            if ($R > D$) then goto RJ ;
        $k^{++}$ ;
    tot_c_iso = 0; $\Delta c_{tot} = 0$ ;
    for each $T_i \in \mathcal{S}_{iso}$ do
        end_BI = $\min(T_i.t_{remain\_life}, D_{max})$ ;
        tot_c_iso += $(T_i \cdot C_{min}) \cdot \lceil \frac{end\_BI}{T_i.P} \rceil$ ;
        $\Delta c_{tot}$ += $(T_i.C_{max} - T_i.C_{min}) \cdot \lceil \frac{end\_BI}{T_i.P} \rceil$
    surplus_c = $D_{max}$ - (tot_c_iso + tot_async_dur) ;
    if (surplus_c $\leq$ 0) then return ACCEPT, long_schedule[] ;
    // allocate surplus duration to ISO requests in a proportionally fair manner
    $\mathcal{S}_{iso}^{sort}$ = {sorted array of all ISO requests in a non-decreasing order of their periods} ;
    for each $T_i \in \mathcal{S}_{iso}^{sort}$ do
        $c\_extra\_per\_job = \min(1, \frac{surplus\_c}{\Delta c_{tot}}) \cdot (T_i.C_{max} - T_i.C_{min})$; // this is extra duration, over $C_{min}$,
            available for each job of request $T_i$
        for each job $J$ of $T_i$, whose release time falls within long_sched[] do
            allocate as much of c_extra_per_job to $J$ before its deadline in long_schedule[]; // it may not always be
                possible to allocate all of c_extra_per_job before the job's deadline
    return ACCEPT, long_schedule[] ;
    RJ:
    if ($T_n$.reqType == ISO) then $\mathcal{S}_{iso} = \mathcal{S}_{iso} - \{T_n\}$ ;
    else $\mathcal{S}_{async} = \mathcal{S}_{async} - \{T_n\}$ ;
    return REJECT
```

schedule. Hence, the time complexity of our EACIAR algorithm is $\mathcal{O}(BI \cdot D_{max} \cdot N_{req}^{iso})$, where N_{req}^{iso} is the number of ISO requests in the system. Thus, it is a linear time algorithm.

F. Correctness of the Admission Control and Scheduling

In this section, we discuss the correctness of our EACIAR algorithm and its associated scheduling in terms of guaranteeing that no request misses its deadline. When there is no ASYNC request in the system, the system follows an EDF schedule. EDF schedule guarantees that every periodic (or ISO) request meets its deadline since they were admitted based on Eq. (1) [5]. When there is at least one ASYNC request, the system follows the schedule computed in *long_schedule[]* using Algorithm 1. In this algorithm, while admitting a new request, it is made sure that the total utilization[4] of the ISO requests, based on their respective C_{min}, is less than or equal to 1, which, as per [5], guarantees that the ISO requests meet their respective deadlines when EDF schedule is followed. The algorithm gives priority to ISO requests over ASYNC request while allocating empty slots. Between *ISO*$_M$ and *ISO*$_F$, *ISO*$_F$ requests are picked up first (since they have shorter deadline than *ISO*$_M$) and allocated empty slots equal to their respective C_{min}, based on EDF. Then the *ISO*$_M$ requests are scheduled in a similar manner. Since, the utilization was calculated based on C_{min} and allocation amount was also C_{min}, the ISO requests would meet their respective deadlines since they follow an EDF schedule. After scheduling all ISO requests, ASYNC requests are scheduled based on EDF. It is made sure that each ASYNC request meets its deadline when allocating slots to it. Finally, any surplus empty slots are proportionally distributed to all the ISO requests before their respective deadlines. Obviously, this operation does not change the ability of ISO requests to meet their deadlines. Thus, all the requests meet their respective deadline.

[4]utilization of a request T_i based, on its C_{min}, is $\frac{T_i.C_{min}}{T_i.P}$

IV. Related Work

A few analytical channel access models for IEEE 802.11ad exist in the literature. A 3D Markov chain based model for SP and CBAP mode of channel access has been proposed in [9]. [10] introduces a model based on Markov chain for CBAP channel access, in the presence of SP channel access as well as deafness and hidden node problems due to directional antennas. An analytical model of SP channel access is presented in [11] which is used to compute worst case delay of packets. It also studies optimal channel sharing between SP and CBAP. An analytical model for SP channel access with channel errors for multimedia traffic is presented in [12].

However, experimental study of admission control and scheduling of SP and CBAP channel access is quite limited in the literature. Authors in [13] design a max-min fair admission control and scheduling algorithm which can only handle isochronous traffic (no asynchronous traffic). They consider two very simple application scenarios; one in which all applications have the same traffic parameter values and the other in which applications choose one set of parameter values out of two pre-configured values. It only handles isochronous requests having periods which are integer fraction of a BI. It has a cubic run time complexity in terms of number of requests. [14] presents scheduling SP channel access using reinforcement learning (RL). It interacts with the network deployment scenario and uses Q-learning to find the optimal SP duration. The number of error free packets received is used as *reward* and the number of packets in the MAC layer queue is used as *states*. [2] proposes three admission control algorithms for isochronous traffic that are fair and compliant with the IEEE 802.11ad. It also proposes an EDF based scheduler which guarantees appropriate SP durations to the admitted isochronous requests before their respective deadlines. These algorithms have a linear run time complexity even when the requests choose any arbitrary values for their parameters and thus, is more efficient and free of restrictions on parameters compared to [13]. Guard time overhead for isochronous traffic in IEEE 802.11ad MAC is accounted for in the admission control and scheduling algorithm presented in [3]. An admission control and scheduling algorithm which can handle both isochronous and asynchronous traffic is presented in [4]. It treats asynchronous traffic as though they are periodic. This keeps the algorithm simple, but results in overallocation of resources to asynchronous traffic and hence, leads to performance loss. In contrast, the admission control and scheduling algorithm presented in this paper allocates only required resources to asynchronous traffic and hence, is more efficient.

Periodic and aperiodic CPU tasks are quite similar to isochronous and asynchronous traffic respectively. Admission control and scheduling of periodic and aperiodic CPU tasks have been studied in the literature [5]–[8]. [5] presents admissibility of a set of preemptive periodic task in an EDF scheduler, but it does not handle aperiodic tasks. Schedulability conditions for non-preemptive periodic tasks are studied in [6], but it also does not consider aperiodic tasks. Joint scheduling of periodic and aperiodic tasks with hard deadline is studied in [7] using a concept called *slack stealing*. Basically, aperiodic tasks use the idle time or slack time left by the periodic task. However, the periodic task have a single parameter for execution time, whereas in IEEE 802.11ad system, the isochronous requests have a range of channel time (C_{min} and C_{max}), so the their algorithm cannot be applied directly. Also, in their system, the priorities of requests are static, whereas in an IEEE 802.11ad system, the priorities of requests can change as new requests arrive. [8] also handles scheduling of both periodic and aperiodic requests, but using EDF which is a dynamic priority algorithm. However, their algorithm assumes that when a new aperiodic request arrives, all previously accepted aperiodic requests have finished, which is a severe limitation. In addition, their algorithm requires computing exact schedule of the periodic requests in every hyper period of the existing periodic requests[5]. The hyper period can potentially become very large, when the periods are relatively prime. In contrast, our algorithm needs to compute the exact schedule until D_{max}, which is the maximum deadline among all the asynchronous requests.

V. Conclusion

We presented a comprehensive admission control algorithm, EACIAR, which handles both isochronous and asynchronous requests in an IEEE 802.11ad MAC. The algorithm is efficient in the sense that it allocates resources exactly as per the traffic requirements of the requests and that it has a linear run time complexity. In addition, the isochronous requests get proportionally fair allocation of SP channel time. We discussed the correctness of the algorithm in terms of guaranteeing required SP duration allocation to every admitted request before their respective deadlines. We believe, this is the first comprehensive algorithm for IEEE 802.11ad MAC which handles both the traffic types and any IEEE 802.11ad compliant traffic parameters.

References

[1] "Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications," 802.11 Working Group of the LAN/MAN Standards Committee of the IEEE Computer Society, Dec. 2016.

[2] A. Sahoo, W. Gao, T. Ropitault, and N. Golmie, "Admission control and scheduling of isochronous traffic in ieee 802.11 ad mac," in *Proceedings of the 24th International ACM Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, 2021, pp. 125–134.

[3] A. Sahoo, W. Gao, T. Ropitault and N. Golmie, "Admission control and scheduling of isochronous traffic with guard time in IEEE 802.11ad MAC," *IEEE Transactions on Mobile Computing*, pp. 1–10, 2022, doi:10.1109/TMC.2022.3207969.

[4] A. Sahoo, P. Tian, T. Ropitault, S. Blandino, and N. Golmie, "Admission control and scheduling of isochronous and asynchronous traffic in ieee 802.11 ad mac," in *2023 IEEE 97th Vehicular Technology Conference (VTC2023-Spring)*. IEEE, 2023, pp. 1–7.

[5] C. L. Liu and J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment," *Journal of the Association for Computing Machinery*, vol. 20, no. 1, pp. 46–61, January 1973.

[6] K. Jeffay, D. F. Stanat and C. U. Martel, "On Non-Preemptive Scheduling of Periodic and Sporadic Tasks," in *IEEE Real-Time Systems Symposium (RTSS)*, December 1991, pp. 129–139.

[7] S. R. Thuel and J. P. Lehoczky, "Algorithms for Scheduling Hard Aperiodic Tasks in Fixed-Priority Systems Using Slack Stealing," in *RTSS*, 1994, pp. 22–33.

[8] H. Chetto and M. Chetto, "Some Results of the Earliest Deadline Scheduling Algorithm," *IEEE Transactions on software engineering*, vol. 15, no. 10, pp. 1261–1269, 1989.

[9] Q. Chen, J. Tang, D. T. C. Wong, X. Peng, and Y. Zhang, "Directional Cooperative MAC Protocol Design and Performance Analysis for IEEE 802.11ad WLANs," *IEEE Transactions on Vehicular Technology*, vol. 62, no. 6, pp. 2667–2677, 2013.

[10] C. Pielli, T. Ropitault, N. Golmie, and M. Zorzi, "An Analytical Model for CBAP Allocations in IEEE 802.11ad," *IEEE Transactions on Communications*, 2020.

[11] C. Hemanth and T. Venkatesh, "Performance Analysis of Service Periods (SP) of the IEEE 802.11ad Hybrid MAC Protocol," *IEEE Transactions on Mobile Computing*, vol. 15, no. 5, pp. 1224–1236, 2015.

[12] E. Khorov, A. Ivanov, A. Lyakhov, and V. Zankin, "Mathematical Model for Scheduling in IEEE 802.11ad Networks," in *2016 9th IFIP Wireless and Mobile Networking Conference (WMNC)*. IEEE, 2016, pp. 153–160.

[13] M. Lecci, M. Drago, A. Zanella, and M. Zorzi, "Exploiting scheduled access features of mmwave wlans for periodic traffic sources," in *2021 19th Mediterranean Communication and Computer Networking Conference (MedComNet)*. IEEE, 2021, pp. 1–8.

[14] T. Azzino, T. Ropitault, and M. Zorzi, "Scheduling the Data Transmission Interval in IEEE 802.11ad: A Reinforcement Learning Approach," in *2020 International Conference on Computing, Networking and Communications (ICNC)*. IEEE, 2020, pp. 602–607.

[5]Hyper period of a set of periodic tasks is the Lowest Common Multiple of their individual periods.