Admission Control and Scheduling of Isochronous Traffic with Guard Time in IEEE 802.11ad MAC

Anirudha Sahoo, Weichao Gao, Tanguy Ropitault and Nada Golmie
National Institute of Standards and Technology
Gaithersburg, Maryland, U.S.A.
Email: {anirudha.sahoo, weichao.gao, tanguy.ropitault, nada.golmie}@nist.gov

***Abstract*—An upsurge of low latency and bandwidth hungry applications such as virtual reality, augmented reality and availability of unlicensed spectrum in the millimeter wave band at 60 GHz have led to standardization of the new generation WiFi systems such as IEEE 802.11ad and 802.11ay. Due to the stringent Quality of Service requirement of those applications, IEEE 802.11ad/ay have introduced contention free channel access called *Service Period*. One type of user traffic supported by IEEE 802.11ad is isochronous traffic, which is essentially periodic traffic that requires certain channel time to be allocated before its period ends. In an earlier work, we presented three Admission Control Algorithms (ACAs) which admit isochronous requests to achieve the above goals. One of these ACAs, the proportional fair allocation admission control (PFAAC), offers the best tradeoff across different performance metrics. But it did not consider guard time (GT) overhead, which is essential in a practical system. In this paper, we present two methods to compute upper bounds on GT overhead. We evaluate performance of the modified PFAAC with the two methods and PFAAC with no GT overhead. The modified PFAAC with the method that uses a tighter upper bound on GT overhead, provides the best performance.**

I. Introduction

There has been an upsurge of networking applications that require high throughput and low delay service. Applications such as Virtual Reality (VR), Augmented Reality (AR), wireless backhaul, high bandwidth connectivity with 8K TVs are some of the examples that demand Quality of Service (QoS) in terms of delay and throughput. A large amount of bandwidth availability (˜12 GHz in the USA) in the unlicensed millimeter wave (mmWave) band at 60 GHz has led to the formulation of the new generation WiFi standards like IEEE 802.11ad and 802.11ay, also known as *Wireless Gigabit* (WiGig). These standards make use of large channel bandwidth (2.1 GHz) (in both of them), Multiple Input Multiple Output (MIMO) and channel bonding techniques (in IEEE 802.11ay) to provide very high data rate [1], [2]. Due to its probabilistic channel access mechanism, stringent QoS requirement of the above said applications cannot be met by contention based channel access traditionally used in older WiFi standards. Hence, in 802.11ad/ay, the WiGig standard supports contention free channel access referred to as *service period* (SP) which is suitable for QoS based applications. An SP is a dedicated channel duration exclusively reserved for communication between a pair of nodes. However, the standard does not specify how to schedule those SPs for user traffic. This is the broad topic of study (for the IEEE 802.11ad system) in this paper.

In IEEE 802.11ad, there is a particular type of user traffic called *isochronous* traffic. Isochronous traffic is essentially periodic traffic that needs certain amount of channel time before its period ends. So, isochronous traffic needs guaranteed channel time allocation with stringent deadlines. Some of the new applications in the AR/VR domain have high throughput and low latency requirement and need almost a constant bit rate (CBR) service. As mentioned in [3], application Riftcat on servers and VRidge on smartphones use a setting in the hardware-accelerated H.264 encoding that generates traffic which is almost CBR. The authors in [3], [4] provide the traffic trace of freely available AR/VR application *Virus Popper* (see Figure 1 in [3], [4]), which shows that the downlink traffic of that application is almost CBR. In addition, many Internet of Things (IoT) applications have periodic traffic [5]. Some of those applications may be time critical, i.e., there may be a deadline associated with the traffic. Aforementioned applications can make use of isochronous traffic support by IEEE 802.11ad to provide the strict QoS in terms of latency and throughput. As pointed out in [6], TP-Link Talon AD7200 [7] was the first IEEE 802.11ad based router[1] followed by Netgear Nighthawk X10 Smart Wifi Routers [8]. As per the finding by [6], current firmware in those commercial products only supports Contention Based Access Periods (CBAP) based channel access using Enhanced Distributed Channel Access (EDCA). It was also pointed out in [6] that the CBAP channel access leads to inefficiencies due to deafness problems in channel sensing and that TCP throughput drops when the channel is crowded with more stations. Thus, it is perceivable that CBAP based channel access may not be adequate for high throughput low latency applications like AR/VR. In the early days of IEEE 802.11ad, applications with such real time stringent QoS requirements were yet to come into existence. Hence, we argue that such application would drive the industry to implement contention free channel access using SP. Due to this requirement, IEEE 802.11ad system needs an admission control algorithm that can determine whether or not a new request can be admitted such that the new as well as the existing requests can be guaranteed their requested channel times. A scheduling algorithm allocates channel time for

[1]The identification of any commercial product or trade name does not imply endorsement or recommendation by the National Institute of Standards and Technology, nor is it intended to imply that the materials or equipment identified are necessarily the best available for the purpose.

each admitted request such that each request gets its requested channel time before its deadline, which is the end of its period. The major challenge for the admission control and scheduler is to be able to guarantee SP duration to admitted user traffic (or requests) before their respective deadlines, while admitting high number of requests and achieving high channel utilization. The algorithms also should be fair while allocating channel time to different users. In [9], we proposed three Admission Control Algorithms (ACAs) which have the above properties. Since isochronous traffic has deadlines to meet, the scheduling algorithm presented in that work is based on Earliest Deadline First (EDF) scheduling algorithm used in scheduling tasks in real time systems [10]. Among the three ACAs, the Proportional Fair Allocation Admission Control (PFAAC) offers the best tradeoff across different performance metrics. But in that work, the ACAs and the scheduling algorithm did not account for guard time (GT) overhead which is required in a real IEEE 802.11ad system. In fact, to the best of our knowledge, there has been no study of admission control and scheduling of IEEE 802.11ad in the literature that includes GT overhead. We believe our work presented in this paper is the first to consider GT overhead in IEEE 802.11ad admission control and scheduling. A GT is inserted between two adjacent allocations and is calculated based on worst case clock drift and maximum allowed number of lost beacons. A GT prevents adjacent transmissions to overlap (or interfere) in time if a station does not receive few beacons and its clock drifts during that period. In this paper, we show methods to compute two upper bounds on GT overhead. Then, we present a modified PFAAC admission control algorithm that accounts for GT overhead based on the computation of upper bounds on GT overhead. The modified PFAAC admission control algorithm still retains the same computational complexity as the original PFAAC algorithm. Hence, it is suitable for a real IEEE 802.11ad system. The main contributions of this work are: i) we present methods to compute two upper bounds on GT overhead, ii) we propose modification to PFAAC admission control algorithm to account for GT overhead, while retaining the same computational complexity, and iii) we present detailed simulation results of the modified PFAAC admission control algorithm to show the effect of GT overhead in terms of different performance metrics.

II. Design of IEEE 802.11ad Admission Control and Scheduling

A. IEEE 802.11ad Medium Access

The medium access time in IEEE 802.11ad consists of an infinite sequence of time durations called Beacon Intervals (BI). A BI is expressed in Time Units (TU), where 1 TU=1024 μs. Each BI consists of a Beacon Header Interval (BHI) followed by a Data Transmission Interval (DTI). The DTI is used for data exchanges and beamforming training among IEEE 802.11ad Stations (STAs) and Personal Basic Service Set (PBSS) Control Point/Access Point (PCP/AP). During a DTI, channel access is specified in two ways. A CBAP duration implies that STAs should access the channel using contention based scheme called EDCA [11]. A SP type of channel access is intended for communication between two STAs or between a STA and a PCP/AP without any contention. CBAP and SP schedules in a DTI are announced by the PCP/AP in a Directional Multi Gigabit (DMG) Beacon frame in the Beacon Transmission Interval (BTI) or Announce frame in the Announcement Transmission Interval (ATI) of BHI (before DTI period starts) [11].

B. IEEE802.11ad Traffic

IEEE 802.11ad has two types of user traffic: *isochronous* and *asynchronous*. Isochronous traffic is suitable for applications that require periodic data transmission with certain QoS requirements. Asynchronous traffic, on the other hand, is a *one time* request, although the requested duration may be granted in multiple allocations. Asynchronous traffic may be *best effort* or may have certain QoS requirements. A station sends *Add Traffic Stream* (ADDTS) requests to its PCP/AP to request resources for its isochronous or asynchronous traffic. The Traffic Specification (TSpec) element in the ADDTS request carries the traffic parameters for which resources need to be allocated. Because of the periodic nature of isochronous traffic, scheduling of this type of traffic is more challenging. The most important traffic parameters of isochronous traffic are [11]:

- Allocation Period (P): Period over which allocation repeats. Allocation period can only be an integer multiple or integer fraction of the 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.
- Minimum Duration: Minimum duration in microseconds in each allocation period. An allocation may be split into multiple chunks. Each chunk must be larger than or equal to this duration. The user can set this value to zero to indicate that this parameter should not be considered. In this study, we assume this parameter to be zero to keep the problem simple.

Let C_i^{op}, called the *operational duration*, be the duration allocated to isochronous request T_i whose traffic parameters are (C_i^{min}, C_i^{max}, P_i). The allocated duration C_i^{op} may change over the lifetime of the request, but it must always satisfy $C_i^{min} \leq C_i^{op} \leq C_i^{max}$. Since this is a periodic request, C_i^{op} must be allocated in every P_i interval. At the beginning of every period, the request is ready to be served, i.e., SP duration can be allocated anytime after the beginning of the period and the corresponding deadline is its period. We refer to the beginning of every period as the *release time* of the request. Denoting R_{i_n} as the n^{th} release time of isochronous request T_i, we have $R_{i_0} = 0$, $R_{i_1} = P_i$, $R_{i_2} = 2P_i$ and so on. In IEEE 802.11ad, when a request arrives, if admitted, it is scheduled in the next BI. Hence, release time $R_{i_0} = 0$ of a request refers to the start

of the next BI. These periodic releases of a request contribute to the load on the system and are captured as its *demand*. So, demand of a request T_i is represented as a series of triples ($C^{op}_{i_n}$, P_i, R_{i_n}), $n = 0, 1, ..$, where $C^{op}_{i_n}$ is the duration to be allocated between R_{i_n} and $R_{i_{n+1}}$. Each triple in a demand is referred to as a *job* of the request.

C. Central Processing Unit Scheduling of Periodic Tasks

Scheduling of isochronous traffic is very similar to Cental Processing Unit (CPU) scheduling of periodic tasks. CPU scheduling of periodic task has been extensively studied in the literature [10], [12]–[14]. In this context, a periodic task T_i is modeled with two parameters *(C_i, P_i)*, where C_i is the duration of the task and P_i is the period as well as deadline of the task. So, the task must be allocated CPU time C_i in every P_i time duration. In [10], the authors present two preemptive scheduling algorithms for periodic tasks: Rate Monotonic Scheduling (RMS) and EDF scheduling. Priority of a task in an RMS scheduler is static and a task with lower period is assigned higher priority. The EDF scheduler sets higher priority to a task with earlier deadline. The priority, in this case, is dynamic. Although RMS is a simpler scheduler than EDF, the maximum utilization that can be achieved while guaranteeing that every task meets its deadline is approximately $\ln 2 \approx 69\,\%$ for a large set of tasks [10]. EDF scheduler can achieve maximum utilization of $100\,\%$ while guaranteeing the deadline of each task. Hence, we choose EDF scheduler to schedule isochronous traffic.

An EDF scheduler can be preemptive or non-preemptive. The feasibility or admissibility of a set of n preemptive tasks for an EDF scheduler is given by [10]:

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

The feasibility of a set of n non-preemptive tasks, in addition to the condition in Eq. (1), needs another condition to be satisfied as given in Theorem 4.1 in [12]. This second condition for non-preemptive task involves finding least upper bound on the processor demand between periods of the tasks and is more complex than the condition given in Eq. (1). Hence, the admission control for non-preemptive EDF scheduler is more complex than its preemptive counterpart. In the case of IEEE 802.11ad Medium Access Control (MAC), when a new request arrives, it is scheduled in the next BI. Hence, in a given BI, requests are scheduled as per their priority based on their deadline (which is their period). This schedule is not perturbed (i.e., does not change) by the arrival of a new request, unlike what happens in CPU scheduling of periodic tasks. This property of IEEE 802.11ad MAC leads to a static schedule in each BI (even though EDF is a dynamic scheduling algorithm), i.e., the schedule does not change in a given BI due to arrival of a new request. So, we choose preemptive EDF as our scheduler in IEEE 802.11ad MAC to take advantage of simpler admission control and scheduling.

D. Admission Control

We borrow the basic principles of admission control from the *feasibility* test of CPU scheduling of periodic tasks. However, the TSpec of isochronous traffic has a range of duration from C^{min} to C^{max}, unlike the CPU scheduling of a task which has a single duration. So, in our case, an admission control algorithm not only determines whether a new request can be admitted or not, but also computes C^{op}, $C^{min} \leq C^{op} \leq C^{max}$, the exact operating allocation duration of the newly admitted request. Let us assume that there are $(n-1)$ requests already in the system. These requests were admitted at their corresponding C^{op}s. Let the operational duration of the newly arriving n^{th} request, T_n, having a period P_n, is computed to be C^{op}_n. The newly arriving isochronous request is admitted if and only if

$$U + \frac{C^{op}_n}{P_n} \leq 1, \tag{2}$$

where $U = \sum_{i=1}^{(n-1)} \frac{C^{op}_i}{P_i}$ is the utilization of the system due to already admitted requests. Depending on the ACA used, while admitting a new request, C^{op} of the existing requests may or may not change during the life of the requests. If C^{op}s of existing requests may change, then utilization of the system due to existing requests, U, needs to be recalculated. In this case, the complexity of the admission control algorithm depends on the complexity of computing U and $C^{op}s$. Also note that ACAs in which C^{op}s of existing requests may change, C^{op}s change only when a new request is admitted and when an existing request leaves the system. To keep the notations simple, we do not make C^{op} a function of time. Hence, when we refer to C^{op}, it refers to its value at that instant. However, if C^{op} of existing requests cannot change throughout the life of the requests, then U does not need to be computed every time a new request is admitted. U can be updated and maintained in the system as and when new requests are admitted or existing requests leave the system. The complexity of the admission control algorithm, in this case, depends only on complexity of computation of C^{op}.

E. Our EDF Based Scheduler

As mentioned earlier, when admitting a request, the admission control algorithm computes the C^{op} of the request. The responsibility of the scheduler is to allocate C^{op} duration to the request before its deadline which is equal to its period. The allocation may be a contiguous duration or a set of non-contiguous fragments[2]. The flowchart of our preemptive EDF scheduler is shown in Figure 1. The figure illustrates how the schedule is computed for a given BI. The algorithm starts with ordering the jobs of all the requests in a non-decreasing order of their deadlines, i.e., from earliest to latest deadline and initializing few variables (Box A). It picks up the ordered jobs

[2]Throughout this paper, we refer to an allocation as a *fragment*. A fragment may refer to an allocation equal to the fragmented part of a job or to an allocation of an unfragmented job.

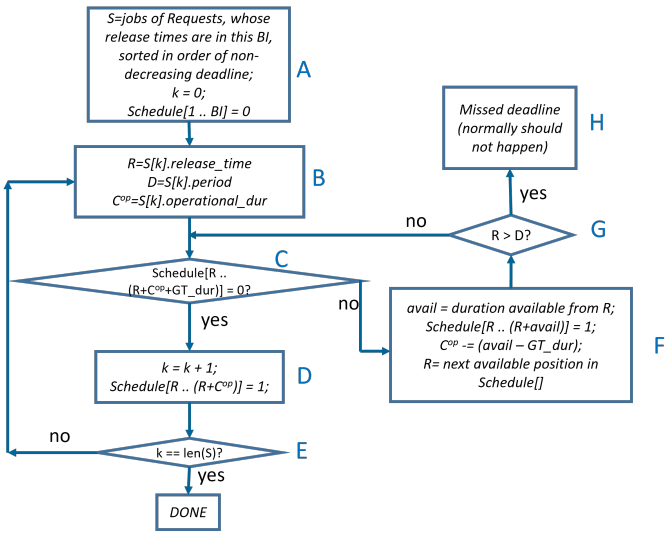


Figure 1: Flowchart of the Proposed EDF Scheduler

one at a time, extracts release time (R), deadline (D) (which is same as the period of the job), and the allocation duration (C^{op}) (Box B). It then checks if the BI has unallocated contiguous duration $(C^{op} + GT\_dur)$ available starting from release time of the job (Decision Box C), where GT_dur is the duration of a GT. If so, then that part of the BI is allocated to the job (Box D) and then the Algorithm loops back to schedule the next job, if there is one (Decision Box E). Otherwise, whatever duration (which is less than C^{op}) is available, is assigned to the request, C^{op} is decremented, taking the GT into account for the fragment just allocated, to determine the remaining duration to be allocated and the allocation point is advanced to the next unallocated (or empty) location in the BI (Box F). Note that if Box F is reached during scheduling of a job, then that job is fragmented, which is akin to preemption in CPU task scheduling parlance. If the next unallocated location is greater than the deadline (Decision Box G), then the request has missed its deadline (Box H). This is an error condition and should not happen in a correct implementation. Otherwise, the algorithm repeats the process of finding unallocated contiguous duration for the reduced C^{op} (Decision Box C). Each request will search for free locations in a BI once (for all its jobs). Hence, the time complexity of our EDF based scheduler is $\mathcal{O}(BI \cdot n)$, where n is the number of requests in the system.

Figure 2 illustrates our EDF based scheduling algorithm using an example. There are three requests, T_1, T_2 and T_3, whose C^{op} values are represented by the length of the filled rectangles (on the top left corner of the figure). The release times of the requests are indicated by color-coded vertical arrows. Their periods, P_1, P_2 and P_3 are also shown. In this example, the periods are integer fraction of a BI. These parameters define jobs of the three requests in one BI duration. Request T_1 has eight jobs, T_2 has three jobs and T_3 has two jobs in one BI. The job number for each request is shown inside the rectangle representing the

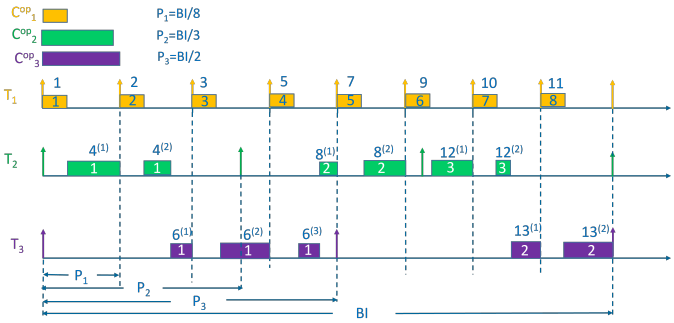


Figure 2: An Example Illustrating Our EDF Based Scheduler

job. These jobs are first ordered in a non-decreasing order of their respective deadlines (Box A in the flowchart given in Figure 1). These allocation order numbers are shown as integer numbers on top of each job. When the C^{op} of a job cannot be allocated contiguously, then the allocation is *fragmented* and the fragment number is shown in parenthesis as a superscript to the order number. For example, the first three jobs of T_1 come first in the order, then the first job of T_2 (order number 4), since its deadline is before the fourth job of T_1 (order number 5). Since contiguous durations equal to C_1^{op} from their respective release times are available in the BI (success in Decision Box C), the first three jobs of T_1 are allocated contiguous durations each equal to C_1^{op} (Box D). Then the first job of T_2 is picked up for allocation (Box B). Since first job of T_1 has already been allocated C_1^{op} from the beginning of the BI, that part is not available for allocation (failure in Decision box C). Hence, T_2 is allocated SP duration immediately following that allocation. However, the duration available is smaller than C_2^{op}. Hence, T_2's allocation is fragmented. The first fragment (shown with a superscript (1)) occupies the empty space in between the first and second allocation of T_1. The second fragment of first job of T_2 is then allocated after the allocation of second job of T_1. These fragmented allocations happens by going through the loop along the Decision Box C, Box F, Decision Box G and back to the Decision Box C. Thus, following the flowchart of Figure 1, we end up with the schedule as shown in Figure 2. We want to reiterate that any new request arriving in the middle of a BI is considered for scheduling in the next BI. Hence, the schedule computed by our algorithm is not perturbed by a newly arriving request and therefore, remains static (or does not change) throughout the duration of the BI. Also, note that, spatial multiplexing of the requests is outside the scope of this study. Hence, we do not consider multiple isochronous requests scheduled simultaneously. Rather, we consider scheduling in one beamformed direction.

III. Handling Guard Time Overhead

In [9], we presented and evaluated three different ACAs which are fair with respect to allocation of C^{op} and standard compliant. Out of the three, we showed that PFAAC offers the best tradeoff across different performance metrics [9]. Hence, in this work, we choose to base our admission control on PFAAC. In PFAAC,

requests are allocated an operational duration (C^{op}) such that surplus allocation over C^{min} expressed as a fraction of the requested allocation range ($C^{max} - C^{min}$) is same for all the requests, i.e., $x_1 = x_2 = \ldots = x_n$, where $x_i, \forall i \in \{1, \cdots, n\}$ is given by

$$x_i \quad = \quad \frac{C_i^{op} - C_i^{min}}{C_i^{max} - C_i^{min}}. \tag{3}$$

Note that allocation durations of admitted requests can change during their lifetimes when this algorithm is used. This algorithm tries to allocate as much duration as possible while maximizing the number of admitted requests. However, PFAAC admission control presented in [9] does not consider GT overhead while admitting requests. A GT is inserted between two consecutive allocations (or fragments), regardless of the type of allocation (SP or CBAP) and is calculated based on the worst case clock drift and maximum allowed number of lost beacons. GT prevents transmissions corresponding to adjacent allocations overlapping (or interfere) in time if a station does not receive few beacons and its clock drifts during that period.

One might think that GT can be accommodated into an admission control algorithm by simply artificially increasing C^{op} of every request by GT duration (assuming that every job of a request needs a GT) and then use Eq. (1) for admission control. But that will not be correct, since the scheduler may break a job of a request into multiple fragments while following the EDF scheduling scheme (recall the example of T_2 in Figure 2 which was fragmented). Each fragment of the job, in that case, requires a GT and hence, allocating just one GT to the job would be incorrect. Thus, the correct way to include GT is to actually compute the exact schedule of the system assuming the new request is admitted and then assigning GT to every fragment[3]. But this would require computing the schedule for one *hyperperiod* of the requests. Hyperperiod of a set of requests is the least common multiple of individual periods of the requests. The schedule computed in one hyperperiod repeats in subsequent hyperperiods as long as the set of requests in the system remains the same. In that case, the GT overhead in one hyper period will continue to the next. However, if there are many requests with periods which are relatively prime to each other, then the hyperperiod can become too large and a generic admission control algorithm following this method can be computationally expensive. In addition, GT overhead computed for one hyperperiod will need to be recomputed every time a new request is admitted or an existing request leaves the system, since these events will change the schedule. Therefore, we resort to computing the worst case GT overhead of a set of requests and use that in Eq. (1) for admission control, which keeps the computational complexity low. The idea is to compute the worst case GT overhead duration and the corresponding worst case GT overhead utilization. Then Eq. (2) is modified to accommodate GT overhead as:

$$U + GT\_overhead\_util + \frac{C_n^{op}}{P_n} \leq 1, \tag{4}$$

where $GT\_overhead\_util$ is the worst case GT overhead utilization. Requests are admitted using Eq. (4) and EDF schedule is computed using the flowchart shown in Figure 1. The schedule so obtained is then modified such that a GT is inserted between every back-to-back (or adjacent) fragments in the schedule.

In the following subsections we first present the theorems to compute the GT overhead followed by a modified PFAAC ACA which takes GT overhead into account. Note that in the following theorems although we depict scenarios with back-to-back fragments, each such pair of fragments will eventually have a GT between them.

A. Computation of Guard Time Overhead

Consider a IEEE 802.11ad scheduler which has a set of admitted isochronous requests denoted as $\mathbb{S}$. Let k be the size of $\mathbb{S}$. For a request $T_i \in \mathbb{S}$, we denote P_i as the period of the request. Therefore, the number of release times of request T_i in one BI is given by

$$N_i = \begin{cases} BI/P_i & \text{if } P_i = BI/m, \\ 1 & \text{if } P_i = m \cdot BI, \end{cases}$$

where $m \in \mathbb{Z}^+$, the set of positive integers.

If the requests are sorted by their respective number of release times in one BI in a non-increasing order, we have:

$$N_1 \geq N_2 \geq \cdots \geq N_{k-1} \geq N_k. \tag{5}$$

Let a_i be the number of fragments allocated to request T_i by the EDF scheduler in a BI. We are interested in determining the number of GT required in a BI in the worst case. The number of fragments in a BI is also the number of GTs. This is because, in the worst case, each fragment of a request may be scheduled back-to-back[4] with another request, or the scheduler may, after scheduling all the fragments, fill all the remaining idle times in a BI with CBAP requests. Thus, each fragment needs a GT. Let G_i be the total number of fragments allocated to requests from T_1 to T_i, i.e., $G_i = a_1 + \cdots + a_i$, which is also the total number of GTs needed. We will refer to the idle duration between two consecutive fragments as an *empty slot*. Note that, while an empty slot can be allocated to the job(s) of the request(s), a GT, even though is an idle duration, cannot be allocated to any job. Hence, GT is an essential overhead.

Theorem 1. *Let s_i be the number of empty slots in a BI after the requests T_1 to T_i are scheduled. Then an upper bound on s_i is given by*

$$s_i \quad \leq \quad s_{i-1} + N_i - 1. \tag{6}$$

[3]We assume the worst case scenario that if idle period may follow a fragment, the scheduler may schedule CBAPs in those idle periods, which would require each fragment to have a GT.

[4]Even though we say back-to-back, there will be a GT between the two back-to-back fragments.

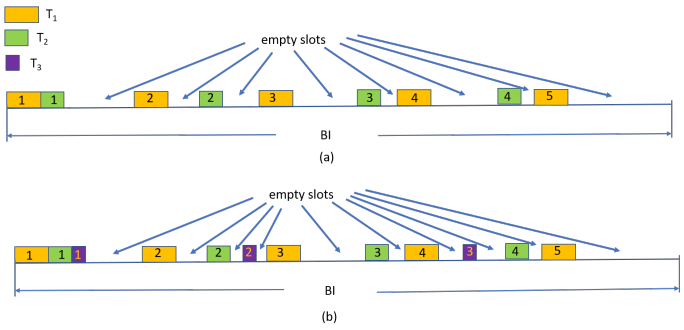


Figure 3: (a) Empty Slots after T_2 is Scheduled (b) Empty Slots after T_3 is Scheduled

Proof. The maximum number of empty slots after requests T_1 to T_{i-1} are allocated is s_{i-1}. Now request T_i is allocated. The number of empty slots after T_i is allocated will be maximum when there are exactly N_i fragments of T_i (that is, none of the jobs in T_i is fragmented) in the schedule and each fragment is allocated in the middle of an empty slot such that it does not lie right before or right after any existing fragment in the schedule. In this scenario, each fragment of T_i splits an existing empty slot into two. Thus, one extra empty slot is created in the BI where a fragment of T_i is scheduled. So, N_i jobs of T_i will create N_i extra empty slots. But the very first fragment will be scheduled right next to the very first fragment of T_{i-1}. Remember that the release times of first job of all the requests happen at the beginning of BI. Hence, the first job of all the requests are scheduled back-to-back at the beginning of a BI. Hence, the total number of empty slots after T_i is scheduled cannot exceed $s_{i-1} + N_i - 1$. □

An example scenario to explain the result from Theorem 1 is shown in Figure 3. Each request is shown with a color and a job of a given request is numbered. After request T_2 is scheduled, there are eight empty slots (Figure 3(a)). When request T_3 is scheduled, maximum number of empty slots will be created when the jobs of T_3 are not fragmented and each job is scheduled in the middle of an empty slot, as shown in the figure (Figure 3(b)). However, the very first job will be right next to first job of T_2. Wherever a job of T_3 is scheduled, except for the first job, it creates an extra empty slot as shown. Since T_3 has three jobs, it creates two extra empty slots. Thus, after scheduling T_3 there are ten empty slots.

Theorem 2. *The sum of number of fragments allocated to T_i and the number of empty slots after T_i is scheduled cannot exceed $s_{i-1} + 2N_i - 1$, i.e.,*

$$a_i + s_i \leq s_{i-1} + 2N_i - 1. \tag{7}$$

Proof. We will use Figure 4 in our proof which shows all possible cases of scheduling a job of a request. The colored

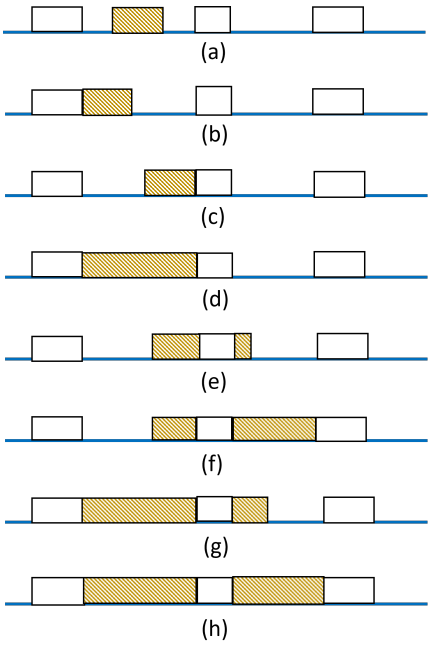


Figure 4: Possible Cases of Scheduling a Job of a Request

rectangles show the allocations for a job of a request being scheduled and the unfilled rectangles represent allocations already done to the jobs of other requests. Theorem 1 gives the maximum value of s_i. This maximum value was obtained assuming that every job of T_i is unfragmented and is scheduled in the middle of an empty slot. In this case, $a_i = N_i$ and hence, using Eq. (6), $a_i + s_i \leq s_{i-1} + 2N_i - 1$. An example of this case is shown in Figure 4(a), where one job of a request which is unfragmented is scheduled in the middle of an empty slot. When this job is scheduled, it splits the empty slot into two, thereby increasing the number of empty slots by one. It also increases number of fragments by one. Thus, the sum of empty slots and fragments goes up by two, i.e., contribution of this job towards $(a_i + s_i)$ is 2. Figure 4(b), (c) and (d) show the other three scenarios of scheduling an unfragmented job. In Figure 4(b) and (c), the number of fragments increases by one, but the number of empty slots remains the same. Hence, the contribution of this job towards $(a_i + s_i)$ is 1. In Figure 4(d), the number of fragments increases by one, but the number of empty slots decreases by one. So, the net contribution of this job towards $(a_i + s_i)$ is 0. When a job is split into two fragments by the EDF scheduler there can be four different cases as shown in Figure 4(e), (f), (g) and (h). In the case of Figure 4(e), the first fragment is scheduled right before and the second fragment is scheduled right after an existing allocation and the second fragment does not fully occupy the empty slot. The first and the second fragment do not change the number of empty slots, but add two to the number of fragments. Thus, the contribution of this job towards $(a_i + s_i)$ is 2. The case shown in Figure 4(f) is similar to Figure 4(e), except that the second fragment exactly fits the empty slot. In this case, the two fragments decrease the

number of empty slots by one, whereas increase the number of fragments by two. Thus, the net contribution of this job towards $(a_i + s_i)$ is 1. For the case in Figure 4(g), the first fragment fully occupies the empty slot whereas the second fragment partially occupies the empty slot. Taking note of the symmetry between Figure 4(f) and 4(g), it is clear that the net contribution of the job to $(a_i + s_i)$ is 1, same as the case in Figure 4(f). Figure 4(h) is the last case where both the fragments exactly fit the respective empty slots. In this case, the number of empty slots decreases by two, but the number of fragments increases by two. Hence, the the net contribution of the job to $(a_i + s_i)$ is 0. Thus, the maximum contribution of a job of a request to $(a_i + s_i)$ is 2 in any possible case. When a job is fragmented into more than two fragments, except for the first and last fragments, all other fragments fully occupy empty slots. These fragments contribute 0 to $(a_i + s_i)$, since each add a fragment and delete an empty slot. The contribution of the first fragment to $(a_i + s_i)$ would be 1 or 0 depending on if it occupies an empty slot partially or fully respectively (this case is similar to the first fragment shown in Figure 4(f) and (g) respectively). Likewise, the last fragment would also contribute 1 or 0 to $(a_i + s_i)$. Thus, all the fragments would contribute at most 2 to $(a_i + s_i)$. Since request T_i has N_i jobs, its maximum contribution to $(a_i + s_i)$ is $2N_i$. But the very first job of T_i will always be scheduled back-to-back at the beginning of a BI right after the first fragment of request T_{i-1}. If the first job is not fragmented, then it may occupy an empty slot partially or fully, in which case its maximum contribution to $(a_i + s_i)$ is 1. If it is fragmented into two, then it is similar to the case shown in Figure 4(g) or (h), in which case the contribution to $(a_i + s_i)$ is 1 or 0 respectively. Hence, the maximum contribution of the very first job of request T_i to $(a_i + s_i)$ is 1 and the maximum contribution of all other $(N_i - 1)$ jobs to $(a_i + s_i)$ is 2. Thus, request T_i has maximum contribution of $2(N_i - 1) + 1 = 2N_i - 1$ towards $a_i + s_i$. Thus, $a_i + s_i \leq s_{i-1} + 2N_i - 1$. □

Theorem 3. *An upper bound on the number of GTs required in a BI for the set of k requests in $\mathbb{S}$ is given by*

$$G_k \leq \begin{cases} N_1 & k = 1, \\ 2\sum_{i=1}^{k-1} N_i - (k-2) & k > 1. \end{cases} \tag{8}$$

Proof. In the very simple case, when there is just one request in the system, the number of empty slot as well as the number of fragments (which is also the number of GTs required) equals N_1, i.e.,

$$\begin{aligned} s_1 &= N_1, \\ a_1 &= N_1. \end{aligned}$$

Hence, $G_1 = N_1$. Now for $k > 1$, we proceed as follows. Using the above identities and Eq. (7) and noting that the number of fragments of the very last request T_k would be less than or equal to s_{k-1}, we have

$$\begin{aligned} a_1 &= N_1, \\ a_2 + s_2 &\leq s_1 + 2N_2 - 1, \\ a_3 + s_3 &\leq s_2 + 2N_3 - 1, \\ &\cdots \\ a_{k-1} + s_{k-1} &\leq s_{k-2} + 2N_{k-1} - 1, \\ a_k &\leq s_{k-1}. \end{aligned}$$

Adding the above equalities and inequalities we have

$$\begin{aligned} &(a_1 + \cdots + a_k) + (s_2 + \cdots + s_{k-1}) \leq \\ &N_1 + (s_1 + s_2 + \cdots + s_{k-1}) + 2(N_2 + \cdots + N_{k-1}) - (k-2). \end{aligned} \tag{9}$$

With simple mathematical manipulation and using $s_1 = N_1$, we get

$$G_k \leq 2\sum_{i=1}^{k-1} N_i - (k-2). \tag{10}$$

□

Theorem 4. *A tighter upper bound, compared to the one given in Theorem 3, on the number of GTs required in a BI for the set of k requests in $\mathbb{S}$ can be found if some of the N_i's are repeated. Let $\mathbb{D}$ be the set of all unique values of N_i's, $1 \leq i \leq (k-1)$. Then a tighter upper bound is given by*

$$G_k \leq \begin{cases} N_1 & k = 1, \\ \sum_{i=1}^{k-1} N_i + 1 + \sum_{d_i \in \mathbb{D}} (d_i - 1) & k > 1. \end{cases} \tag{11}$$

Proof. The same argument as in Theorem 3 proves the trivial case of $k = 1$. For $k > 1$, we proceed as follows. For an incoming request T_i, first consider the case when its N_i is equal to N_j of request T_j which has already been admitted into the system. In this case, all the jobs of T_i will be scheduled right next to the respective jobs of T_j (as per the EDF scheduling scheme) and hence, none of the fragments of T_i will start in the middle of an empty slot. If none of the jobs of T_i is fragmented, i.e., there are exactly N_i fragments, then the number of empty slots after T_i is scheduled will not exceed s_{i-1} and $(a_i + s_i)$ will not exceed $(s_{i-1} + N_i)$. If a job of T_i is split into two fragments, then the first fragment must completely occupy an empty slot, since it has to start right next to a fragment of T_j (similar to the first fragment shown in Figure 4(g) and (h)). The first fragment decreases the number of empty slots by one, but adds a new fragment, with no net contribution to $(a_i + s_i)$. Therefore, $(a_i + s_i)$ is decided by the second fragment. The second fragment will always start right after (i.e., back-to-back) an existing fragment. If the second fragment partially occupies an empty slot (similar to the second fragment shown in Figure 4(g)), then it does not change the number of empty slots, but adds one fragment. So, its contribution to $(a_i + s_i)$ increases by one, i.e., $s_i = s_{i-1}$ and the fragment contributes one towards a_i. If, on the other hand, the second fragment fully occupies an empty slot (similar to the second fragment shown in Figure 4(h)), then it adds a fragments and deletes an empty slot, i.e., $s_i = s_{i-1} - 1$ and the job contributes one towards

a_i. This means it has no net effect on $(a_i + s_i)$. So, when a job is split into two fragments, its contribution to $(a_i + s_i)$ can increase at most by one. This argument can also be extended to a job of T_i which is split into more than two fragments. So, in any scenario, a job of T_i can contribute at most one towards $(a_i + s_i)$. Since T_i has N_i jobs, it can contribute at most N_i to $(a_i + s_i)$. Thus, in this case $(a_i + s_i) \leq s_{i-1} + N_i$. Now consider the case $N_i \neq N_j$, $\forall T_j$, i.e., period P_i of request T_i is not equal to the period of any of the requests currently admitted. In this case, the results of Theorem 2 to compute $(a_i + s_i)$ applies. Therefore,

$$a_i + s_i \leq s_{i-1} + N_i + b_i(N_i - 1), \tag{12}$$

where

$$b_i = \begin{cases} 0 & \text{if } N_i = N_j, i \neq j \text{ for some } T_j \in \mathcal{S}, \\ 1 & \text{otherwise.} \end{cases} \tag{13}$$

Note that $b_1 = 1$.

Thus,

$$\begin{aligned} a_1 &= N_1, \\ a_2 + s_2 &\leq s_1 + N_2 + b_2(N_2 - 1), \\ a_3 + s_3 &\leq s_2 + N_3 + b_3(N_3 - 1), \\ &\cdots \\ a_{k-1} + s_{k-1} &\leq s_{k-2} + N_{k-1} + b_{k-1}(N_{k-1} - 1), \\ a_k &\leq s_{k-1}. \end{aligned}$$

Adding the above equalities and inequalities we have

$$\begin{aligned} &(a_1 + \cdots + a_k) + (s_2 + \cdots + s_{k-1}) \leq \\ &N_1 + (s_1 + s_2 + \cdots + s_{k-1}) + (N_2 + \cdots + N_{k-1}) + \\ &b_2(N_2 - 1) + \cdots + b_{k-1}(N_{k-1} - 1). \end{aligned} \tag{14}$$

Noting $s_1 = N_1$ and performing simple mathematical manipulations we have

$$G_k \leq \sum_{i=1}^{k-1} N_i + 1 + \sum_{d_i \in \mathbb{D}} (d_i - 1).$$

□

B. Admission Control with Guard Time

PFAAC call admission control presented in [9] is suitably modified to account for the GT overhead using the upper bounds on GT overhead presented in Section III-A. The modified algorithm is presented in Algorithm 1, which we will refer to as PFAAC_GTO, henceforth. The method of computing GT overhead using Theorem 3 and Theorem 4 are referred to GT Algorithm 1 (GTA1) and GT Algorithm 2 (GTA2) respectively.

Note that when PFAAC is used, C^{op}s of existing requests may change. Hence, the PFAAC_GTO algorithm, in addition to deciding whether the new request be admitted or rejected, computes new C^{op}s of existing requests as well as the new request, if the new request is admitted. The algorithm rejects the new request if the utilization of the system, assuming all the existing requests and the new request are allocated their respective minimum allocation (C^{min}) plus the GT overhead,

Algorithm 1 Admission_Control_PFAAC_GTO

1: **input:** C^{min}, C^{max} and P of all existing $(n-1)$ requests and the new request.
2: **output:** Accept or Reject; C^{op} of each request if the new request is accepted.
3: $U_n^{min} = \sum_{i=1}^{n} \frac{C_i^{min}}{P_i}$
4: G_n = upper bound on the number of GT using Eq. (8) (if using GTA1) or using Eq. (11) (if using GTA2)
5: $GT\_overhead\_util$ = $(G_n \cdot GT\_dur)$/BI
6: **if** $(U_n^{min} + GT\_overhead\_util) > 1$ **then return** Reject
7: $U_{surplus} = 1 - (U_n^{min} + GT\_overhead\_util)$
8: $\Delta u_{tot} = 0$
9: **for** i=1 to n **do**
10: $\quad \Delta u_{tot} = \Delta u_{tot} + \frac{C_i^{max} - C_i^{min}}{P_i}$
11: **for** i=1 to n **do**
12: $\quad C_i^{op} = C_i^{min} + \min(1, \frac{U_{surplus}}{\Delta u_{tot}}) \cdot (C_i^{max} - C_i^{min})$
13: **return** Accept

exceeds 1 (Line 6). Otherwise, it computes the *surplus* utilization based on this minimum allocation and GT overhead (Line 7). This surplus utilization ($U_{surplus}$) is then distributed to each individual request in proportion to its difference of utilization between maximum and minimum to the total (over all requests) difference of utilization. Hence, the operating utilization of request T_i is given by

$$u_i^{op} = u_i^{min} + \frac{\Delta u_i}{\Delta u_{tot}} \cdot U_{surplus}, \tag{15}$$

where $u_i^{op} = \frac{C_i^{op}}{P_i}$, $\Delta u_i = \frac{C_i^{max}}{P_i} - \frac{C_i^{min}}{P_i}$ and $\Delta u_{tot} = \sum_{i=1}^{n} \Delta u_i$. Multiplying both sides of Eq. (15) by P_i and inserting the term $\min(1, \frac{U_{surplus}}{\Delta u_{tot}})$ to take care of the fact that $\frac{U_{surplus}}{\Delta u_{tot}}$ could be greater than 1, we get

$$C_i^{op} = C_i^{min} + \min(1, \frac{U_{surplus}}{\Delta u_{tot}}) \cdot (C_i^{max} - C_i^{min}), \tag{16}$$

which is the expression in Line 12. Note that when $\frac{U_{surplus}}{\Delta u_{tot}}$ is greater than 1, it implies that there is enough surplus for every request, so that every request can be allocated its C^{max}. In this case, it can be verified from Eq. (16) that $C_i^{op} = C_i^{max}$. This also explains why the term $\min(1, \frac{U_{surplus}}{\Delta u_{tot}})$ should be used in Eq. (16) to prevent C_i^{op} going above C_i^{max}. Noting the for loops in Line 9 and Line 11 of Algorithm 1, each of which iterates n times, it is obvious that the time complexity of this ACA is $\mathcal{O}(n)$.

To account for the GT overhead, we make minor modification to the EDF scheduler presented in Figure 1. After the flowchart ends, we identify allocations which are right next to each other. We insert a GT between such allocation pairs.

IV. Performance Results

A. Performance Metrics

In this section, we define the performance metrics used to evaluate our admission control algorithms.

- Acceptance Ratio (AR) : This is the fraction of total requests that are admitted by the ACA. Higher AR implies that an IEEE 802.11ad MAC can support a greater number of flows or applications.
- Allocation Efficiency (AE): This metric represents efficiency of an ACA in terms of allocation of SP duration. For a request T_i, it is defined as $\frac{C_i^{op}-C_i^{min}}{C_i^{max}-C_i^{min}}$. Note that AE is a value between 0 and 1. AE is 0 when the ACA allocates C_i^{min} to T_i and is 1 when it allocates C_i^{max}. An ACA with higher AE is preferable to an application since that translates to higher throughput for the application.
- BI Utilization (BU): This metric is the fraction of a BI duration that has been allocated to requests for SP channel access by the corresponding STAs.
- Degree of Fragmentation (DoF): When C_i^{op} of a request T_i cannot be allocated in one chunk, then the allocation is said to be *fragmented*. DoF_i of a request T_i in a given duration is given by $\frac{N_{chunk}-N_{job}}{N_{job}}$, where N_{chunk} is the number of chunks or fragments allocated to the request (during scheduling) and N_{job} is the number of jobs of the request in that duration. Thus, if there is no fragmentation of a request T_i, then $N_{chunk} = N_{job}$, and $DoF_i = 0$. In IEEE 802.11ad, a GT is inserted between two consecutive allocations which is an overhead for the system. Hence, DoF is an indicator of GT overhead of the system. Note that regular EDF scheduler cannot guarantee an upper bound on the number of fragments of a job or equivalently it cannot guarantee a minimum fragment size. In some cases, e.g., if the *minimum duration* traffic parameter is to be considered, such a guarantee may be required. In such cases, a variation of EDF called *limited preemption EDF* scheduler can be used [15].
- Normalized Delay: The delay of a job of a request is the difference of time instance of end of allocation of the job and the release time of the job. Note that if the job is fragmented, the end of allocation of the job is the end of allocation of its last fragment. This metric essentially measures the time it would take to finish transmitting a job after it is available (released) at the MAC and is an indicator of packet level delay. Normalized delay is the delay normalized with respect to the period of the request, i.e., it is the ratio of delay of a job to the period of the request.
- Normalized Jitter: Jitter is the absolute difference of delay of two consecutive jobs of a request. In another words, it is the variation in two successive delays of two consecutive jobs of a request. This is normalized with respect to the period of the request to represent normalized jitter. Hence, normalized jitter is the ratio of jitter to the period of the request.

B. Simulation Experiment Design

Performance of our admission control and scheduling algorithms is evaluated by using an in-house simulator developed by us. In this paper, we are interested in performance of our SP based admission control and scheduling algorithm at different system loads. SP operates in interference free condition in an IEEE 802.11ad network. Hence, in this paper, we did not consider channel conditions, beamforming and interference due to side lobes. Therefore, developing our own simulator is a much faster approach than using some existing network simulators. Using simulation makes it much easier to simulate different scenarios and operate the network at high loads, which is much harder in a testbed. In addition, a testbed built using commercial off the shelf devices would not have SP implementation nor would they provide knobs to implement a new algorithm.

Our simulation experiments are designed as follows. ADDTS requests arrive with a Poisson distribution having mean arrival rate λ, which is varied from 5 to 50 requests per BI in steps of 5. BI duration is set at 102 400 μs. The GT duration used in our experiments is 10 μs. This value is also used in the 802.11ad implementation in the ns3 network simulator [16]. The maximum allocation duration of a request (C^{max}) is uniformly distributed between 10 μs and 100 μs and is considered a per BI value. Thus, if the request period is an integer fraction of BI, then the randomly generated C^{max} value is scaled down by that fraction. If the request period, on the other hand, is an integer multiple of BI, then C^{max} value is scaled up by that integer multiple. The allocation interval ratio, C_{min}/C_{max}, is uniformly distributed between 0.5 and 1.0. The lifetime of each request follows a normal distribution with the mean duration of 100 BIs and standard deviation of 10 BIs. Thus, about 95 % of lifetimes are between 80 BIs to 120 BIs. Lifetime of a request is rounded down to its nearest period. The integer n that defines the integer fraction or integer multiple of period of a request is uniformly distributed between 1 to 5. The experiments are carried out in three *scenarios*: i) when all the requests have periods which are an integer multiple of BI (Scenario 1), ii) when all the requests have periods which are an integer fraction of BI (Scenario 2) and iii) when 30 % requests have periods which are integer multiple of BI and 70 % requests have periods which are integer fraction of BI (Scenario 3). Scenario 1 simulates applications requiring low bandwidth (e.g., IoT applications), whereas Scenario 2 represents applications requiring high bandwidth and low delay (e.g., streaming video). Scenario 3 is appropriate for a mix of these two types of applications. The experiments are run for a duration equal to 1000 BIs. For a given random parameter, the same sequence of random numbers are used to represent the values of the parameter across the three scenarios, i.e., same seed is used across the three scenarios to generate the random parameter. This ensures that all the scenarios are fed with the same values of input parameters and makes the comparison across scenarios fair. In each scenario, three cases with respect to GT are considered: "No Guard Time" (NGT) (no GT overhead was considered in the ACA), GTA1 and GTA2. NGT case is considered as a baseline case that does not require the complex

analysis of worst case GT overhead during admission control and could offer better performance. However, NGT is only suitable for a theoretical IEEE 802.11ad system where all the stations in the Wireless LAN are perfectly synchronized and hence, its scheduler does not add any GT. Accordingly, in our experiments no GT is added by the scheduler when using NGT algorithm. Note that if NGT admission control is implemented in a real IEEE 802.11ad system, its scheduler will insert GT wherever required and that can lead to undesired consequence of some requests missing their deadlines.

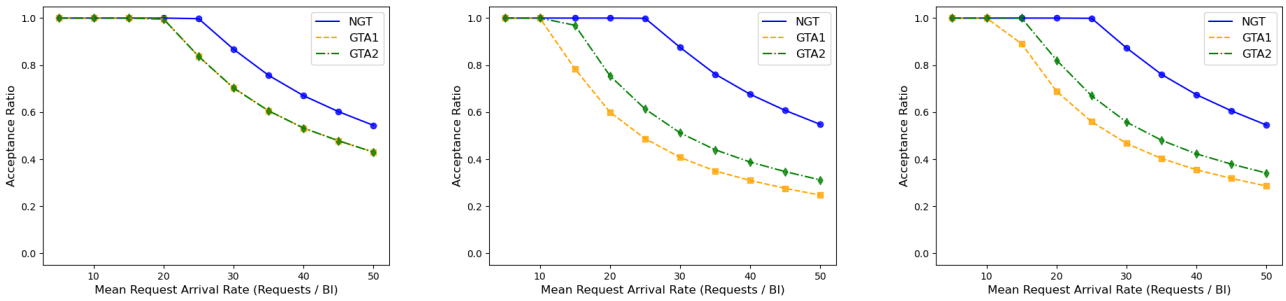
C. Experiment Results

Before we present our results, we explain the *box and whisker plots* used to depict the statistics of some of the performance metrics. The central mark of each box is the median, the edges of the box are the 25^{th} and 75^{th} percentiles. The upper *whisker* represents the largest data point that is within 1.5 times the interquartile range (distance between the upper and lower quartiles) above the upper box edge. Similarly, the lower *whisker* represents the smallest data point that is within 1.5 times the interquartile range below the lower box edge. Outliers beyond the whiskers are not shown.

1) Performance in Terms of Acceptance Ratio: Figure 5 shows how AR changes as the mean request arrival rate λ increases for the three scenarios. At low λ, the AR is $100\,\%$ for all scenarios and all GT algorithms, since the system has low utilization. As λ increases, the load on the system increases and more requests are rejected. Hence, AR starts to decrease. AR for NGT is higher than GTA1 and GTA2 at high λ in all the scenarios. At high λ, system load is high and GT overhead, in case of GTA1 and GTA2, occupy enough duration in a BI to prevent some requests to be admitted. But NGT does not have any GT overhead, hence, it admits more requests leading to higher AR. AR of GTA1 and GTA2 are identical in Scenario 1. In Scenario 1, the periods of all the requests are integer multiple of BI. Thus, in a BI a request will have at most one fragment scheduled. Thus, GTA1 and GTA2 produce the same GT overhead. This can also be verified by putting $N_i = 1, \forall i$ into Eq. (8) and Eq. (11). But in Scenario 2, GTA2 performs better than GTA1 when request arrival rate is high. In this scenario, periods are integer fraction of a BI and there are requests whose periods are the same, hence GTA2 is able to get tighter upper bound on GT overhead. Same is the case for Scenario 3, in which the requests with periods equal to fraction of BI have identical periods and those help reduce GT overhead when GTA2 is used.

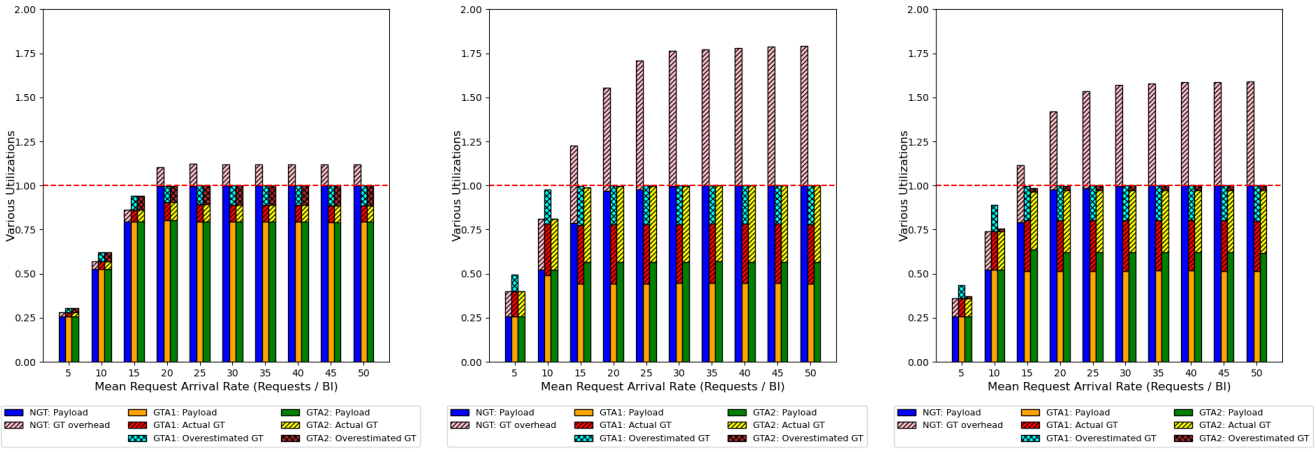
2) Performance in Terms of Various Utilizations: Performance graphs of various utilizations (expressed as a fraction of a BI duration) versus mean request arrival rate are presented in Figure 6. Once the system reaches steady state (after a few BIs), these different utilizations are averaged over all BIs until the end of the experiment duration to obtain the respective average utilizations. The graphs plot actual payload utilization and actual GT utilization (which is the overhead due to provisioning of GT) and overestimated GT which is the difference between computed GT (using either GTA1 or GTA2) and the actual GT. Note that our scheduler would only insert a GT between a pair of fragments which are placed back-to-back (or adjacent). Hence, the estimated GT (either using GTA1 or GTA2), which is computed based on the worst case scenario, will be higher than or equal to the actual GT inserted by our scheduler. Also note that the overestimated GT is *wasted* as idle duration in a BI due to overestimation in GTA1 or GTA2. In Scenario 1, as λ increases, payload utilization as well as GT overhead and overestimation of GT increases until a certain λ, but after $\lambda = 20$, as the system utilization goes to $100\,\%$ all those metrics become almost constant. At high utilization, a new request is admitted only when an existing request leaves. Thus, the schedule of admitted requests does not change much and hence various utilizations do not change much. Overestimated GT for GTA1 and GTA2 are equal, since, as mentioned before, for Scenario 1, $N_i = 1$ and that leads to actual GT and overestimated GT for GTA1 and GTA2 to be equal. In Scenario 2, as λ increases, generally the average payload utilization, actual GT and overestimated GT increase initially. But after $\lambda = 20$, they all remain constant. As more requests arrive, more requests are admitted, which leads to higher payload along with higher GT as well as overestimated GT. However, after the system reaches its limit, new requests could only be admitted when there is room created by an existing request leaving. The overall schedule does not change much, which leads to almost constant utilization of these three parameters. The relative performances of various parameters in Scenario 3 are almost similar to Scenario 2, since the mix of request is dominated by requests having periods equal to fraction of a BI.

Notice that the overestimated GT for GTA2 algorithm in Scenario 2 is almost zero. This means that the upper bound on estimation of GT using GTA2 for this scenario is very tight. GTA2 distinguishes between the cases when N_i of the new request is equal or not equal to N_j of any existing requests (see Eq. (12)). This makes the upper bound on GT tight for GTA2. GTA1 does not makes this distinction and hence, it incurs significant overestimation of GT. However, when N_i is not equal to N_j of any existing requests and N_i is not relatively prime with all the N_j's of the existing requests, then some of the jobs of those requests (that are not relatively prime with each other) will be scheduled back-to-back and hence, GTA2 may overestimate the GT overhead. So, in general, if the unique N_i's of the requests are relatively prime to each other then GTA2 would result in a tight upper bound in Scenario 2. The tightness worsens as the number of unique N_i's that are not relatively prime with each other increases. In Scenario 2, N_i is the exact value of number of release times of the request T_i in a BI. However, in Scenario 1, N_i is equal to 1 for every request, which is an overestimation when the period of a request is more than a BI. This, in turn, leads to overestimation of GT. As explained before, this overestimation of GT in Scenario 1 is same for GTA1 and GTA2. Hence, GTA1 and GTA2 do not produce tight upper bound in Scenario 1.



(a) Scenario 1 (b) Scenario 2 (c) Scenario 3

Figure 5: Acceptance Ratio vs. Mean Request Arrival Rate (BI=102.4 ms)

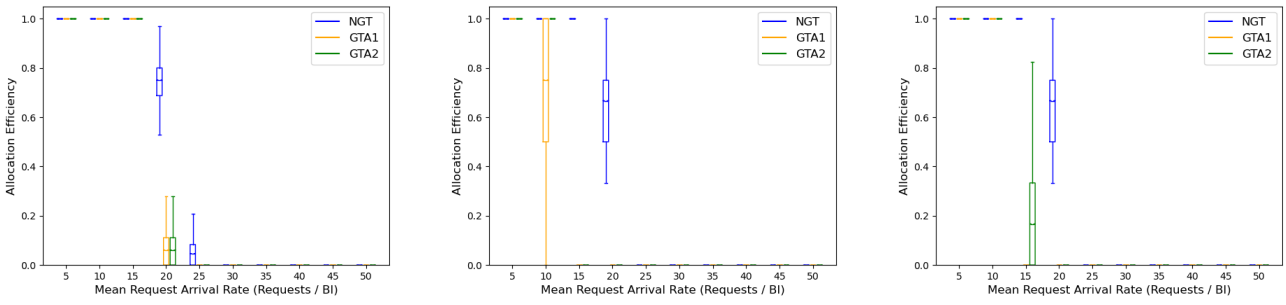


(a) Scenario 1 (b) Scenario 2 (c) Scenario 3

Figure 6: Various Utilizations vs. Mean Request Arrival Rate (BI=102.4 ms)

For the NGT algorithm, which is meant for a theoretical system (in which the scheduler does not insert any GT), we show the GT overhead only to illustrate the problems with this admission control algorithm if implemented in a real IEEE 802.11ad system. The GT overhead, for this illustration purpose, is computed assuming that the real IEEE 802.11ad MAC scheduler inserts a GT between every pair of jobs scheduled back-to-back. In all the scenarios, for the NGT algorithm the total utilization goes above 1.0 at high λ. This implies that the schedule of the requests is not feasible. Hence, some requests would miss their deadlines (more discussion on this in Section IV-C8). The GT overhead in Scenario 1 is smaller than in Scenario 2. When periods of requests are fraction of a BI, at high λ they will have more fragments compared to when periods are multiple of a BI, which leads to more GT overhead.

3) Performance in Terms of Allocation Efficiency: Figures 7 shows the performance of different GT algorithms in terms of average AE of the requests as λ increases. To compute the average AE of a given request, AE is computed in every period and averaged over its lifetime. Then average AEs of all the requests are plotted as a box and whisker plot. Average AE of NGT is always better or same as GTA1 and GTA2 in all the scenarios as request arrival rate increases. In Scenario 1, at low λ ($\lambda \leq 15$ requests per BI), all the three algorithms provide AE of $100\,\%$. But beyond that, it goes down and eventually becomes zero. As discussed in the performance in terms of AR, GTA1 and GTA2 are always equal in this scenario since a request will have at most one fragment scheduled in a BI. Hence, C^{op} values are identical across GTA1 and GTA2 which leads to same AE for the two algorithms. AE for NGT is higher than GTA1 or GTA2 for medium values of λ ($\lambda = 20$ and 25 requests per BI). Since NGT does not consider GT at all, it has more idle periods in BI which is used to allocate higher C^{op} to the requests. But at high lambda ($\lambda \geq 30$ requests per BI), AE becomes zero for all the algorithms, since PFAAC_GTO tries to admit more requests at the cost of lowering AE. In Scenario 2, the relative performance of the three algorithms is the same as



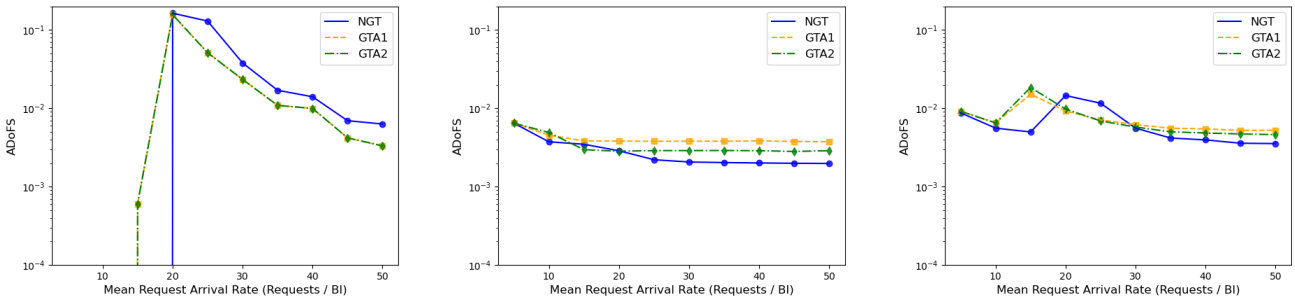
(a) Scenario 1 (b) Scenario 2 (c) Scenario 3

Figure 7: Average Allocation Efficiency vs. Mean Request Arrival Rate (BI=102.4 ms)

Scenario 1. However, for all the algorithms, AE goes to zero at a lower λ compared to Scenario 1. This is because in Scenario 2 allocations are more tightly fitted in a BI since all the requests repeat multiple times. Thus, the ACA reduces AE much faster than in Scenario 1 to admit more requests. As expected, between GTA1 and GTA2, AE for GTA1 goes below 1.0 at a lower λ compared to GTA2, since GTA2 uses a tighter upper bound of GT overhead. In Scenario 3, AE for GTA2 falls to zero at a higher λ than GTA1 and AE for NGT falls to zero at a higher λ than GTA2. This is in line with GT estimation of the three algorithms.

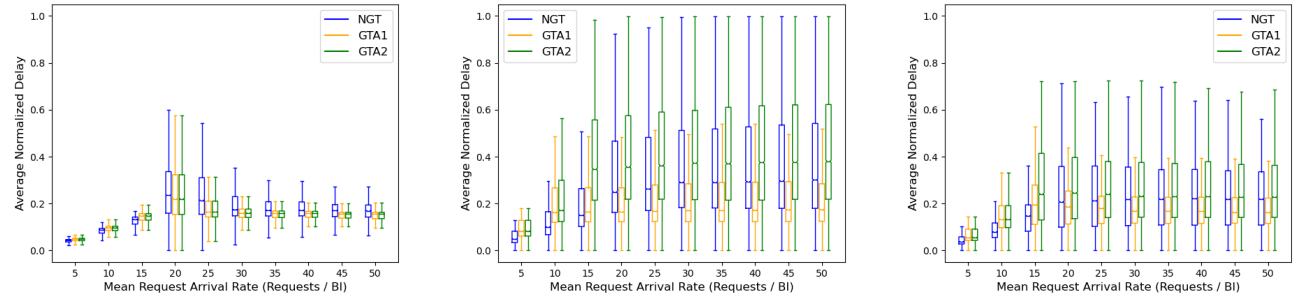
4) Performance in Terms of Degree of Fragmentation: Performance in terms of average DoF of the system (ADoFS) for the three scenarios is captured in Figures 8. To compute ADoFS, DoF of every request is computed in every period and then average DoF is computed over its lifetime. Another averaging is done over average DoF of all the requests to obtain ADoFS. In Scenario 1, at low λ, ADoFS is zero for all the three algorithms. In this scenario, the deadlines of the requests are much longer (than Scenario 2) and there is enough room to schedule the jobs of requests without fragmentation. But at high λ, fragmentation of jobs is inevitable because of high system load. GTA1 and GTA2 overestimate the GT overhead which leads to much idle periods left in an BI (see Figure 6), but NGT fully utilizes the BI ($100\,\%$ payload). Hence, NGT has higher ADoFs than GTA1 and GTA2. In Scenario 2, ADoFS for the three algorithms are very close to each other (note the logarithmic scale) for the entire range of request arrival rates. At high load, whether accounting for GT or not, the fragmentation is almost similar across the GT algorithms. In Scenario 3, ADoFS of the three algorithms are almost equal throughout except for λ values between 15 to 25 requests per BI, where NGT deviates slightly from the other two. Since Scenario 3 predominantly has requests having periods equal to integer fraction of a BI (like the requests in Scenario 2), the performance is similar to Scenario 2 except for λ values between 15 to 25 requests per BI. At those λ values utilization of the system moves towards $100\,\%$ and fragmentation for all the three algorithms starts to increase. The difference of performance between the three algorithms is due to slight difference in system utilization around those λ values. Beyond λ value of 25 requests per BI, the utilization of the system remains almost constant for all the three algorithms and hence, their AoDFS remains close to each other.

5) Performance in Terms of Normalized Delay: Figure 9 shows average normalized delay (AvND) of requests in the three scenarios as λ increases. To compute the AvND of a given request, normalized delay is computed in every period and averaged over its lifetime. Then AvND of all the requests are plotted as a box and whisker plot. In Scenario 1, at low λ ($\lambda \leq 15$), NGT algorithm has lower AvND than the other two, since NGT does not have any GT and at low load there is no fragmentation (see Figure 8), jobs of requests finish earlier in NGT, which leads to lower delay. At higher λ ($\lambda \geq 20$), NGT has higher payload allocation than the other two. It achieves high payload by allocating extra fragments to the requests which may already have been allocated at least C_{min}, when it finds idle durations to fill. This leads to more fragmentation (see Figure 8) and higher delay for the request. Median AvND of GTA1 and GTA2 are almost identical throughout, because the two algorithms have the same actual GT and payload utilization as well as the same ADoFS, which transaltes to same delay. In Scenario 2, at low λ ($\lambda \leq 15$), median AvND of NGT is the lowest followed by GTA1 and then GTA2. At low λ, the actual GT allocated to GTA1 and GTA2 is high. These GT add to the delay, which leads to higher delay with GTA1 and GTA2 than NGT. Between GTA1 and GTA2, GTA2 has less overestimated GT (see Figure 6) and hence, actual GT allocated in GTA2 is higher than GTA1 (note that GTA2 has higher payload). This leads to AvND of GTA2 to be more than GTA1. But at high λ ($\lambda \geq 20$), GTA2 has high actual GT overhead (very little overestimation of GT), whereas NGT, although has higher payload, has no GT overhead. The high GT overhead in GTA2 leads to more fragmentation. Hence, AvND for GTA2 is higher than NGT. For GTA1, payload is very low with relatively lower GT overhead (and quite a bit of overestimation) (see Figure 6). Hence, AvND in this case is the lowest. Relative performance



(a) Scenario 1 (b) Scenario 2 (c) Scenario 3

Figure 8: Average Degree of Fragmentation of the System (ADoFS) vs. Mean Request Arrival Rate (BI=102.4 ms)



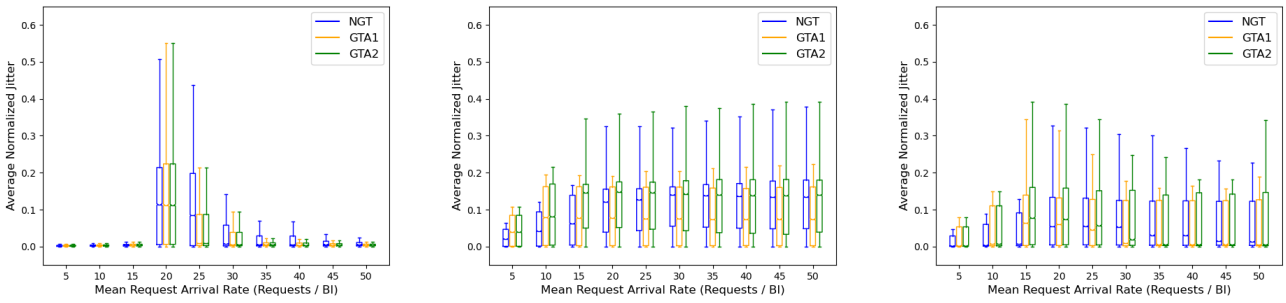
(a) Scenario 1 (b) Scenario 2 (c) Scenario 3

Figure 9: Average Normalized Delay (AvND) vs. Mean Request Arrival Rate (BI=102.4 ms)

of the three GT algorithms in Scenario 3 is similar to those in Scenario 2, since Scenario 3 has much higher percentage of requests with periods equal to fraction of a BI.

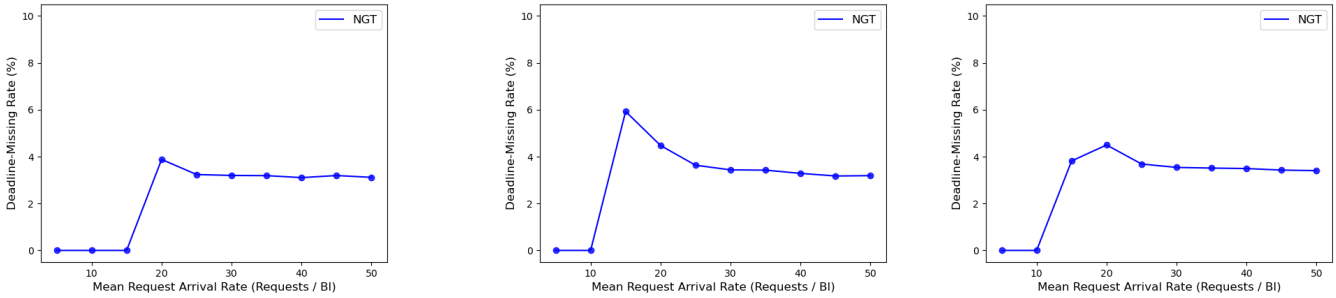
6) Performance in Terms of Normalized Jitter: Figure 10 captures performance in terms of average normalized jitter (AvNJ). To compute the AvNJ of a given request, normalized jitter is computed in every period and averaged over its lifetime. Then AvNJ of all the requests are plotted as a box and whisker plot. In Scenario 1, AvNJ for all the GT algorithms are almost zero for low λ and high λ. At low λ, there is enough idle time that the relative scheduling time of jobs of a request is almost the same across its period, which means negligible jitter. At high λ, new requests are admitted when some existing request leave. So, the new requests do not perturb the schedule much. The relative scheduling time of jobs of the requests does not vary much across periods (since the period or deadline is much more relaxed in this scenario compared to Scenario 2). Thus, the AvNJ is almost zero. But at $\lambda = 20$ and $\lambda = 25$, the system utilization inches towards $100\,\%$, hence schedule of existing requests changes to accommodate new request and that leads to some jitter in all the three algorithms. In Scenario 2, at low λ, median AvNJ of NGT algorithm is lower than the other two. For GTA1 and GTA2, any change in the relative schedule of jobs of a request incurs higher jitter due to the GT. At high λ, NGT and GTA2 run at full capacity, i.e., there is no idle duration left in a BI. Also, requests have much tighter deadline and new requests could change the schedule of existing requests which leads to higher jitter. In the case of GTA1, at high λ, it has overestimated GT (which is basically idle duration in a BI) (see Figure 6), hence, it has more room to not perturb schedules of many existing requests. Therefore, its jitter is lowest at high λ.

7) Uncertainty in the Experiments: The experiments were run for a long duration equal to 1000 BIs, due to which the run times were very high. So, it was not possible to get the uncertainty measurements of AR for every λ value. We measured standard deviation of AR for $\lambda = 5$, 25 and 50 requests per BI for all the three scenarios and all the three GT algorithms to get some sample values. The minimum and maximum standard deviation, across all scenarios and all GT algorithms, were $0\,\%$ and $0.72\,\%$ of the mean respectively. Based on these sample values of standard deviation and the fact that our experiments were run for a long duration of 1000 BIs, we expect the standard deviation of AR and various average utilizations for all λ values to be very very small.



(a) Scenario 1 (b) Scenario 2 (c) Scenario 3

Figure 10: Average Normalized Jitter (AvNJ) vs. Mean Request Arrival Rate (BI=102.4 ms)



(a) Scenario 1 (b) Scenario 2 (c) Scenario 3

Figure 11: Percentage of Requests Missing Deadline vs. Mean Request Arrival Rate

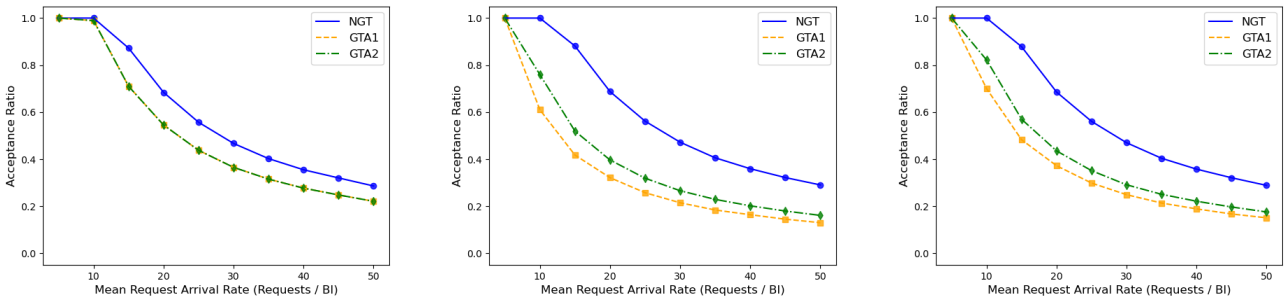
8) Discussion on the NGT algorithm: We want to have some cautionary discussion against using NGT algorithm in a real IEEE 802.11ad system. From the various performance graphs presented above, it might appear that NGT algorithm is a good choice to keep admission control simple and get better performance. But an IEEE 802.11ad scheduler puts GT between two back-to-back allocations. Hence, using NGT algorithm in call admission control can lead to missed deadlines for requests, especially at high load situation. This fact is presented in Figure 11, which plots the percentage of admitted request that miss its deadline at least once in its lifetime as λ increases. For all the three scenarios, at high load situation ($\lambda \geq 15$), some requests miss their deadlines at least once. Even a single missed deadline is a violation of IEEE 802.11ad specification. Therefore, although NGT can sometimes achieve much higher utilization based on payload than GTA1 or GTA2 (for example, see Figure 6(b) at high request arrival rate), it should not be adopted in the call admission control in a real IEEE 802.11ad system.

9) Performance with Shorter BI: We want to observe how the performance of our admission control and scheduling algorithm changes when the BI is shorter. So, in this section, we present few sets of results when BI is set to 51.2 ms, keeping all other simulation parameters unchanged.

Figure 12 presents the AR vs. mean request arrival rate. Comparing them with the corresponding graphs in Figure 5, we observe that the relative performance of the three GT algorithms remains almost the same. However, since the BI duration is shorter, AR falls below 100 % at a smaller value of λ when BI=51.2 ms.

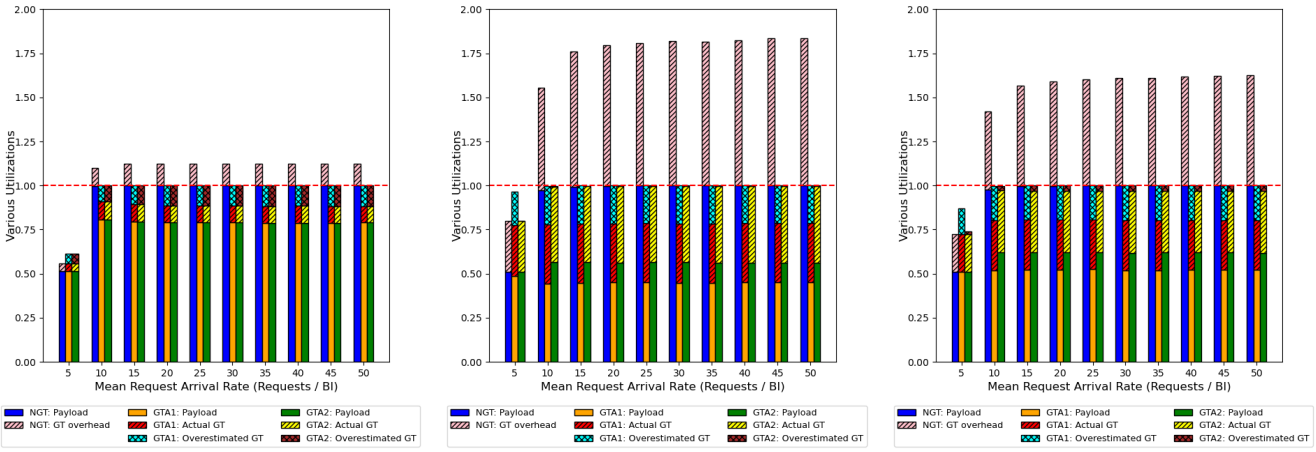
In terms of various utilizations, comparing the Figures 13 and 6, we see that the relative performance of the three GT algorithms does not change much. However, at low λ, total utilization with BI=51.2 ms is higher and also total utilization reaches 100 % at a lower λ which is expected due to the shorter BI. However, the GT overheads are almost identical once the total utilization reaches 100 % at high λ. For the NGT algorithm also we notice that the utilization goes above 1.0 at a smaller λ value than that for BI=102.4 ms due to the shorter BI.

Comparing Figures 14 and 7, it is clear that the relative performance of the three GT algorithms, in terms of AE, does not change much. However, the median AE of respective algorithms falls below 100 % as well as goes to zero at a lower λ due to the shorter BI.



(a) Scenario 1 (b) Scenario 2 (c) Scenario 3

Figure 12: Acceptance Ratio vs. Mean Request Arrival Rate (BI=51.2 ms)



(a) Scenario 1 (b) Scenario 2 (c) Scenario 3

Figure 13: Various Utilizations vs. Mean Request Arrival Rate (BI=51.2 ms)

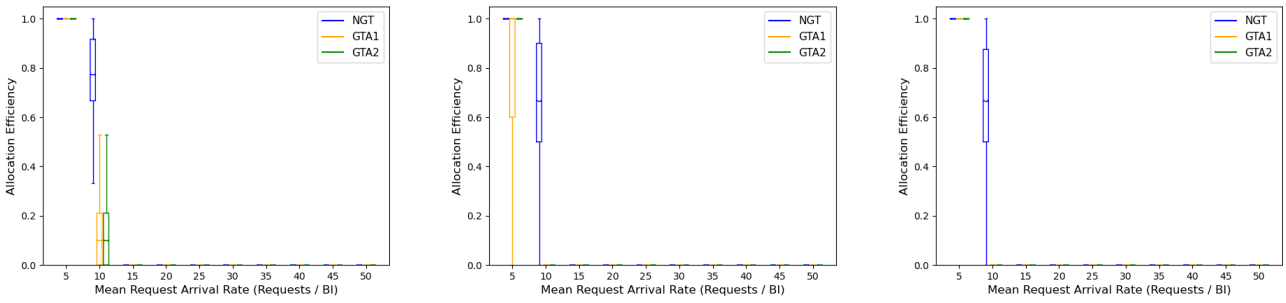
Comparing the normalized delay obtained with BI=51.2 ms (Figure 9) with the earlier results (Figure 15), we notice that for high λ (when total utilization is 100 %), the respective median normalized delays are almost the same in all the scenarios. Note that, in this case, the absolute delay would be lower (in fact, almost half) than that for the BI=102.4 ms case. However, at low λ, when utilization is low, for a given λ, the normalized delay is higher. This is because, in this case, for a given λ, when BI=51.2 ms, the utilization is higher than when BI=102.4 ms. This leads to more fragmentation and therefore, higher delay.

Relative performance of the three GT algorithms in terms of normalized jitter with BI=51.2 ms is very similar to that of BI=102.4 ms, i.e., the respective performance graphs of the three scenarios look very similar. Hence, explanation given for Figure 10 mostly applies to BI=51.2 ms case also. Due to space limitation we are not able to provide the graphs here.

V. Related Work

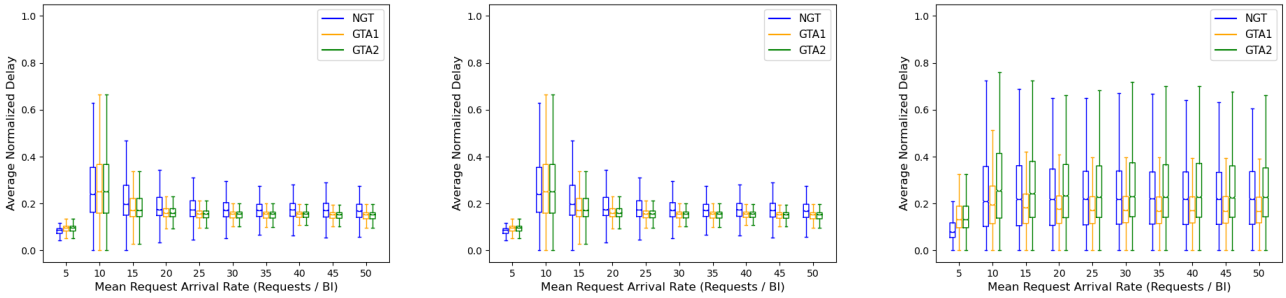
There have been few analytical studies done on IEEE 802.11ad channel access. A 3D Markov chain based analytical model for performance analysis of SP and CBAP mode of channel access is presented in [17]. In [18], authors present a Markov chain based analytical model for CBAP allocation. The model accounts for presence of SPs and deafness and hidden node problems associated with directional antennas during CBAP. An analytical model for SP access is proposed in [19], in which the authors study the worst case delay of SP packets using the model. They also discuss a way to optimally allocate a channel between SP and CBAP access. [20] proposes a scheduling method based on analytical model for a multimedia flow using SP channel access in the presence of channel errors.

There has been very little work reported in the literature in experimental study of admission control and scheduling of SP and CBAP allocations. In [21], the authors present two algorithms for joint admission control and scheduling of periodic traffic streams using SP allocation. However, the authors only consider very simple application scenarios and do not consider GT overhead in the admission control and scheduling. In one scenario, all the applications are assumed to have the same traffic



(a) Scenario 1 (b) Scenario 2 (c) Scenario 3

Figure 14: Average Allocation Efficiency vs. Mean Request Arrival Rate (BI=51.2 ms)



(a) Scenario 1 (b) Scenario 2 (c) Scenario 3

Figure 15: Average Normalized Delay (AvND) vs. Mean Request Arrival Rate (BI=51.2 ms)

parameters. In the other case, there are only two sets of traffic parameters and an application chooses one from the two sets. Also, this study only considers periods which are integer fraction of the BI. Their algorithms can become too expensive if it is to be implemented in a real IEEE 802.11ad system to be able to schedule applications with many different traffic parameters. In [9], we presented three admission control algorithms and a EDF based scheduling algorithm for isochronous traffic, which can handle requests that are integer fraction and integer multiple of a BI. The algorithms are carefully designed so that their run time complexity is not high for any general set of requests having many different set of traffic parameters. However, that study did not consider guard time overhead. In fact, to the best of our knowledge, the work presented in this paper is the first one to consider GT overhead in IEEE 802.11ad admission control and scheduling. Finally, in [22], the authors present a reinforcement learning (RL) based scheduling of SP allocation which finds the optimal duration of each SP. The RL based scheme uses Q-learning and interacts with the network deployment scenario to get the optimal SP duration. Queue size, in terms of number of packets, at the MAC layer represents *states* and *reward* is represented as a function of number of received packets and the *action* taken.

VI. Conclusion and Future Work

One of the main contributions of this study is to compute upper bounds on GT overhead in IEEE 802.11ad scheduling. We used these upper bounds to modify the PFAAC presented in [9] to account for GT overhead in scheduling. We presented two upper bounds on GT overhead. The modified PFAAC, called PFAAC_GTO, shows how to incorporate the two methods, referred to as GTA1 and GTA2, into call admission control. We provide a comprehensive simulation results comparing the performance of these two algorithms along with no GT (NGT) case with respect to different performance metrics. Although NGT case generally performs better than GTA1 and GTA2, we pointed out that it may not be suitable for a practical IEEE 802.11ad system. In fact, if NGT is used for admission control, then we show that when the scheduler actually inserts GT, some requests may miss their deadlines, especially at high load situations. Thus, admission control should use GTA1 or GTA2. Since, GTA2 uses tighter upperbound on GT overhead, it performs better than GTA1 in most of the scenarios and in most of the performance metrics. Hence, PFAAC_GTO with GTA2 is a good choice for admission control for a practical IEEE 802.11ad system.

In terms of future work, we are looking at admission control and scheduling of IEEE 802.11ad MAC when both isochronous

and asynchronous traffic may be present in the system. We would like to study non-preemptive versions of the EDF scheduler which will have reduced performance in terms of acceptance ratio but should have less overhead in terms of GT.

REFERENCES

[1] Y. Ghasempour, C. R. 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, 2017.

[2] T. Nitsche, C. Cordeiro, A. B. Flores, E. W. Knightly, E. Perahia, and J. C. Widmer, "IEEE 802.11ad: Directional 60 GHz Communication for Multi-Gigabit-per-second Wi-Fi," *IEEE Communications Magazine*, vol. 52, no. 12, pp. 132–141, 2014.

[3] M. Lecci, F. Chiariotti, M. Drago, A. Zanella, and M. Zorzi, "Temporal characterization of xr traffic with application to predictive network slicing," *arXiv preprint arXiv:2201.07043*, 2022.

[4] M. Lecci, M. Drago, A. Zanella, and M. Zorzi, "An open framework for analyzing and modeling xr network traffic," *IEEE Access*, vol. 9, pp. 129 782–129 795, 2021.

[5] T. Hoßfeld, F. Metzger, and P. E. Heegaard, "Traffic modeling for aggregated periodic iot data," in *2018 21st Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN)*. IEEE, 2018, pp. 1–8.

[6] H. Assasa, S. K. Saha, A. Loch, D. Koutsonikolas, and J. Widmer, "Medium access and transport protocol aspects in practical 802.11 ad networks," in *2018 IEEE 19th International Symposium on" A World of Wireless, Mobile and Multimedia Networks"(WoWMoM)*. IEEE, 2018, pp. 1–11.

[7] "Talon AD7200 Multi-Band Wi-Fi Router." [Online]. Available: https://www.tp-link.com/us/home-networking/wifi-router/ad7200/

[8] "Netgear Nighthawk ©X10." [Online]. Available: https://www.netgear.com/home/wifi/routers/ad7200-fastest-router/

[9] A. Sahoo, W. Gao, T. Ropitault, and N. Golmie, "Admission Control and Scheduling of IsochronousTraffic in IEEE 802.11ad MAC," in *ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM)*, November 2021.

[10] 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.

[11] "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.

[12] 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.

[13] 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.

[14] 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.

[15] S. Baruah, "The limited-preemption uniprocessor scheduling of sporadic task systems," in *17th euromicro conference on real-time systems (ECRTS'05)*. IEEE, 2005, pp. 137–144.

[16] H. Assasa, J. Widmer, T. Ropitault, and N. Golmie, "Enhancing the ns-3 ieee 802.11ad model fidelity: Beam codebooks, multi-antenna beamforming training, and quasi-deterministic mmwave channel," in *Proceedings of the 2019 Workshop on Ns-3*, ser. WNS3 2019. New York, NY, USA: Association for Computing Machinery, 2019, p. 33–40. [Online]. Available: https://doi.org/10.1145/3321349.3321354

[17] 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.

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

[19] 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.

[20] 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.

[21] 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.

[22] 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.

A nirudha Sahoo (Senior Member, IEEE) received the Ph.D. degree in computer science from Texas A&M University, College Station, TX, USA. He worked as a Software Engineer with Intergraph Corporation, Huntsville, AL, USA, and then as a Senior Software Engineer with Cisco Systems, San Jose, CA, USA. He was an Associate Professor with IIT Bombay, India. He is currently a Computer Scientist with the National Institute of Standards and Technology. His research interests include spectrum sharing, dynamic spectrum access, and other areas of wireless networks.

W eichao Gao received the Doctor of Science Degree in Information Technology from Towson University, Towson, MD, USA in 2021 and MBA from University of Michigan in 2011. He worked as a guest researcher with the Wireless Network Division, National Institute of Standard and Technology, Gaithersburg, MD, USA during his doctorate program, and is currently working as a contractor in the same division. His research interest includes Internet of Things, wireless networks, and data sciences.

T anguy Ropitault received the Ph.D. degree in computer science from the Institut Mines-Télécom, Rennes, France, in 2015. He is currently working as a Contractor with the Wireless Network Division, National Institute of Standards and Technology, Gaithersburg, USA. His current research interest includes mmWave and WiFi sensing system-level performance evaluation.

N ADA GOLMIE (nada@nist.gov) received her Ph.D. in computer science from the University of Maryland at College Park. Since 1993, she has been a research engineer at the National Institute of Standards and Technology (NIST). From 2014 until 2022, she served as the chief for Wireless Networks Division at NIST. She is an IEEE Fellow, and a NIST Fellow in the Communications Technology Laboratory. Her research in media access control and protocols for wireless networks led to over 200 technical papers presented at professional conferences, journals, and contributed to international standard organizations and industry led consortia. She is the author of "Coexistence in Wireless Networks: Challenges and System-level Solutions in the Unlicensed Bands," published by Cambridge University Press (2006). She leads several projects related to the modeling and evaluation of future generation wireless systems and protocols and serves as the NextG Channel Model Alliance chair.