

A Novel AID Shuffle Mechanism for RAW Slot Assignment in IEEE 802.11ah Networks

Xin Zhang and Kwan L. Yeung
 Department of Electrical and Electronic Engineering
 The University of Hong Kong
 Email: {zhangxin, kyeung}@eee.hku.hk

Abstract—To implement the new restricted access window (RAW) medium access control mechanism in IEEE 802.11ah, stations (STAs) must be assigned to RAW slots based on their association IDs (AIDs). In this paper, we first identify the *fixed subgroup* problem with the current round-robin slot assignment. To address it, a novel AID shuffle mechanism is proposed for each STA to distributedly find a different and distinct *temporary AID* in each RAW. Each STA then reuses the round-robin slot assignment with its temporary AID. As a result, the set of STAs assigned to each subgroup/slot are randomized, yet with the same subgroup size. Simulation results show that the transmission fairness problem with the original round-robin assignment is solved and the system throughput is also improved.

I. INTRODUCTION

IEEE 802.11, or Wi-Fi, is one of the most widely adopted wireless local area network (WLAN) technologies. Wi-Fi targets at offering high throughput to a small number (tens) of devices/stations located at a short distance from each other (tens of meters). With a huge success in WLAN market, IEEE 802.11 is recently extended to also support low-power machine-to-machine (M2M) communications, or Internet of Things (IoT), resulting in a new amendment called IEEE 802.11ah [1]. Unlike human-oriented communications, M2M communications usually do not require high throughput, but high connectivity and high power efficiency are essential. Unlike the existing Wi-Fi technologies that work on either 2.4GHz or 5GHz ISM bands, IEEE 802.11ah extends Wi-Fi into sub-1GHz ISM band. With the propagation characteristics of sub-1GHz band, an AP can cover an area with a radius up to one kilometer and support up to 8,191 devices. (Note that throughout the paper, we will use stations (STAs), devices and sensors interchangeably.) Using the distributed coordination function (DCF) (§9.3.4 in [2]), which is essentially a CSMA/CA protocol, having thousands of STAs competing for the channel will lead to a high chance of frame collisions [3] even under ideal channel conditions (i.e., no transmission errors and hidden terminals). Based on a more realistic channel model of sub-1GHz [4], it is found [5] that the propagation loss between two STAs is much higher than that between a STA and AP. In other words, most of the STAs in an IEEE 802.11ah WLAN are hidden to each other. As a result, the performance of DCF in an 802.11ah WLAN will be much worse than that in an 802.11 WLAN.

To address the problem, a new medium access control mechanism called restricted access window (RAW) is introduced in

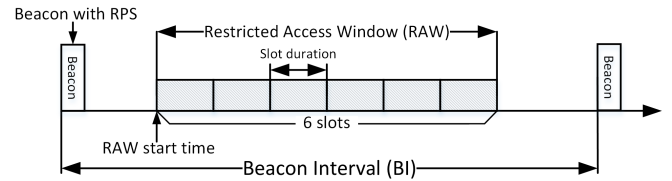


Fig. 1. Restricted access window (RAW) mechanism.

IEEE 802.11ah. The basic idea (§10.22.5 in [1]) is to organize STAs into groups. Only STAs of a particular group are allowed to access the channel in a reserved time interval, i.e., restricted access window. By doing so, the number of STAs competing for the channel in a RAW is limited, collision can thus be reduced and the throughput can be enhanced. To further reduce the number of competing STAs, a RAW can be further divided into slots, and each slot is assigned to a specific subgroup of STAs. Besides, new grouping schemes [6], [7] are also proposed for organizing STAs not hidden from each other to the same group.

To inform STAs of the RAW allocation in a beacon interval (BI), including the number of RAWs in a BI, the number of slots in each BI, and which STAs can access each slot, a RAW parameter set (RPS) element (§9.4.2.192 in [1]) is added to the beacon frame as shown in Fig. 1. A single RPS element can announce multiple RAWs. For each RAW, RPS contains the RAW start time, the number of slots in the RAW and the slot duration. The group of STAs allowed to access this RAW are identified by RAW Start AID and RAW End AID. AID, or association ID, is a unique 13-bit value¹ AP assigned to a STA during the association process. Any STA whose AID value is in the specified range can access this RAW. To determine k , the index of the slot a STA is assigned to, each STA uses the following mapping function (§10.22.5.3 in [1]):

$$k = (\text{AID} + N_{\text{offset}}) \bmod m, \quad (1)$$

where m is the number of slots in the RAW, and N_{offset} is an offset value to improve fairness (with respect to the slot index) and equals to the two least significant octets of the frame checksum (FCS) field of the beacon frame. From (1), we can see that STAs with consecutive AIDs will be assigned to

¹In IEEE 802.11ah, AP can update/change the AID of a STA, which is not allowed in IEEE 802.11.

slots in a round-robin fashion. We thus denote this STA-to-slot mapping as *round-robin assignment*. Let the number of STAs assigned to a RAW be N and assume N is an integer multiple of m . Round-robin assignment ensures each subgroup/slot has exactly N/m STAs. If N is not an integer multiple of m , the number of STAs assigned to different RAWs can differ by at most one.

With round-robin assignment, for any two STAs, denoting their AIDs as AID_i and AID_j , if

$$AID_i \equiv AID_j \pmod{m},$$

according to the property of congruence, for any N_{offset} , it is also true that,

$$AID_i + N_{\text{offset}} \equiv AID_j + N_{\text{offset}} \pmod{m}.$$

That means these two STAs will always be assigned to the same subgroup, independent of the value of N_{offset} . In other words, the membership of every subgroup is *fixed* because it always contains the same set of STAs. We call it *fixed subgroup problem*. Consider an example of $N = 8$ STAs with AIDs from 1 to 8. If $m = 4$, the 8 STAs will always be divided into four fixed subgroups, $\{1,5\}$, $\{2,6\}$, $\{3,7\}$ and $\{4,8\}$. With a different N_{offset} , each subgroup, as a whole, can be mapped to a different time slot k (but the membership within each subgroup remains the same). Since STAs of the same subgroup always compete for channel access in the same slot, if a STA is assigned to a heavily loaded subgroup, i.e., the aggregated traffic generated by other STAs in the group is heavy, it will always experience higher frame collision probability than another STA assigned to a lightly loaded subgroup. This causes transmission unfairness among STAs of different subgroups. Besides, the uneven traffic loads in different subgroups can also hurt the throughput performance.

To address the fixed subgroup problem, we propose a novel AID shuffle mechanism in this paper to randomize the set of STAs in each subgroup while keeping the same subgroup size. To be more specific, each STA is pre-installed with a set of judiciously designed shuffle patterns. Based on the FCS field of the beacon frame, STAs will shuffle/exchange their AIDs in a distributed fashion and according to the set of shuffle patterns chosen. In the end, each STA will get a different and distinct *temporary AID* in each RAW it is assigned to. The round-robin slot assignment is then conducted using the temporary AID instead of the original/permanent one. We can show that with our proposed FCS-based AID shuffle mechanism, the traffic load in different slots is balanced, the transmission fairness among STAs is improved, and the system throughput is also increased.

The rest of the paper is organized as follows. In Section II, our AID shuffle mechanism is presented. In Section III, our shuffle pattern design is detailed. In Section IV, some related work on shuffle pattern design is discussed. Performance evaluation is conducted in Section V and we conclude the paper in Section VI.

II. AID SHUFFLE MECHANISM

As pointed out in Section I, when the number of STAs in the target group (N) and the number of slots in a RAW (m) are given, mapping STA-to-slot using the round-robin function in (1) will cause the fixed subgroup problem. Since each STA has a different traffic load, the aggregated traffic load in different slots/subgroups will not be the same/balanced. As a result, if a STA is stuck in a heavily loaded subgroup, it will always experience much higher frame collision probability than those in other subgroups.

Under the assumption that the traffic load of each STA is known a priori, to balance the aggregated traffic load in different slots/subgroups, one way [8] is to assign STAs to subgroups such that the inter-subgroup load difference is minimized. But with this *traffic-aware* STA-to-slot assignment, the number of STAs assigned to each subgroup may not be the same – a determining factor of DCF performance, and getting the traffic load in advance is difficult. Besides, since the STA-to-slot mapping is conducted by AP, AP must announce the mapping/grouping results to all STAs, e.g. using the RPS element in beacon. But the current RPS element, as reviewed in Section I, does not support arbitrary grouping.

Another approach is to let each STA randomly choose a slot from the m slots in the assigned RAW. We call it *random assignment*. Intuitively, random assignment eliminates the need to announce the grouping results (as needed by the traffic-aware assignment), solves the fixed subgroup problem (with round-robin assignment), does not need a priori knowledge of traffic loads yet can balance the load in different slots (in the long run). Since each STA will compete with a different set of STAs each time it is assigned to a slot, the transmission fairness problem can also be easily solved. But a major problem with random assignment is that the number of STAs assigned to each slot in the same RAW can fluctuate significantly. An extreme case can be that all STAs (randomly) choose the same slot to access the channel while leaving the rest of $m-1$ slots empty. That means with random assignment, the traffic load in different slots of the *same* RAW can be poorly balanced, adversely affecting the system throughput.

Our goal is to design a STA-to-slot mapping mechanism to solve the transmission unfairness caused by the fixed subgroup problem, while keeping the system throughput as high as possible. The proposed mechanism should be compliant with the IEEE 802.11ah specifications and easy to implement. Without loss of generality, we focus on a specific group of N STAs, which are always allowed to access channel in the same RAW. We call this group as the *target* STA group. We denote the STAs in the target STA group by $STA_1, STA_2, \dots, STA_N$, and their corresponding AIDs by $AID_1, AID_2, \dots, AID_N$. AIDs are assigned consecutively such that $AID_i - AID_j = 1$, if $i - j = 1$. An AID list $L = [l_i]$ ($i = 1$ to N) is formed by setting $l_i = AID_i$.

Our proposed FCS-based AID shuffle mechanism functions as follows. Each STA is pre-installed with a set of 32 judiciously designed shuffle patterns (to be detailed in the next

section), which can be obtained from AP during association. Each bit in the 32-bit frame checksum (FCS) field of the beacon corresponds to a specific shuffle pattern. If a particular bit in FCS is '1', the corresponding shuffle pattern will be applied to shuffle/exchange the location of the AIDs in L . If the bit is '0', the current shuffle pattern will be skipped. When all bits in FCS have been used to shuffle AIDs in L , each STA_{*i*} obtains a temporary AID (or AID'_{*i*}), which is stored at $l_i = \text{AID}'_i$. STA_{*i*} will then apply AID'_{*i*}, instead of its original/permanent AID_{*i*}, to the round-robin assignment in (1) to determine its slot index k .

Since all STAs start with the same initial AID list L and they shuffle AIDs in L based on the same FCS, the shuffle process at different STAs is synchronized and the same final L will be obtained. This guarantees that each STA_{*i*} will get a *different* and *distinct* AID'_{*i*} in each RAW. Applying AID'_{*i*} to (1), the set of STAs assigned to each slot will be randomized, yet the set size can be differed by at most one. If N is an integer multiple of m , each slot will have exactly N/m STAs.

III. SHUFFLE PATTERN DESIGN

We denote a shuffle pattern by (A, B) . To shuffle the AIDs in L with pattern (A, B) , L is first split into two sub-lists L_1 and L_2 , where L_1 consists of the first $|L_1| = \lceil \frac{N \cdot A}{A+B} \rceil$ AIDs in L , and L_2 contains the remaining AIDs. The AIDs from L_1 and L_2 are then interleaved by strictly alternating between the two such that A AIDs from L_1 (first) and B AIDs from L_2 (second). Note that if the number of AIDs left in L_1 (or L_2) is less than A (or B), all of them will be used. Consider an example of applying pattern $(A = 1, B = 1)$ to shuffle $L = [1, 2, 3, 4, 5, 6, 7, 8]$. L is first split into $L_1 = [1, 2, 3, 4]$ and $L_2 = [5, 6, 7, 8]$. L_1 and L_2 are then interleaved to form $L = [1, 5, 2, 6, 3, 7, 4, 8]$.

By varying the values of A and B , we can get different shuffle patterns. Since there are only 32 bits in the FCS field, we only need 32 shuffle patterns, one corresponding to each bit. Without loss of generality, let s_i denote the shuffle pattern corresponding to the i -th bit in FCS. We need to map s_i ($i = 1$ to 32) to different (A, B) tuples. Without loss of generality, we first sort the shuffle patterns according to the sum of A and B (i.e., $A + B$) in ascending order. So shuffle patterns with smaller $A + B$ will be ahead of the patterns with larger $A + B$. Then, for the patterns with same $A + B$, they are further sorted in ascending order of A . From the ordered list thus obtained, the first 32 shuffle patterns will form the shuffle pattern set $S = [s_i]$, or

$$\begin{aligned} S = \{ & s_1 = (1, 1), \\ & s_2 = (1, 2), s_3 = (2, 1), \\ & s_4 = (1, 3), s_5 = (2, 2), s_6 = (3, 1), \\ & \dots, \\ & s_{29} = (1, 8), s_{30} = (2, 7), s_{31} = (3, 6), s_{32} = (4, 5) \}. \end{aligned}$$

It is interesting to note that, no matter which shuffle pattern s_i is used, the top A AIDs from L_1 are always the top A AIDs in L . (In the study of shuffling playing cards [9], this is

called an out shuffle.) This is undesirable because we want to randomize the location of AIDs in L through shuffle. We thus define a reverse shuffle pattern $s_i^{-1} = (A, B)^{-1}$. The only difference between s_i and s_i^{-1} is that, with s_i^{-1} , the AIDs from L_2 will be placed into L first.

To summarize, when (active) STA_{*i*} is allowed to access a RAW in the current beacon interval (as indicated by the RPS element in beacon), it first finds a temporary AID'_{*i*} and then applies AID'_{*i*} to the round-robin assignment in (1) to determine k , its assigned RAW slot index. To find AID'_{*i*}, STA_{*i*} first constructs the original AID list L using the RAW Start AID and RAW End AID from the RPS element. It then checks the FCS field starting from the first/least-significant bit. If the i -th bit is '0', no shuffle will be conducted. If the i -th bit is '1' and there are odd number of '1's in the first i bits, L is shuffled with s_i . If the i -th bit is '1' and there are even number of '1's in the first i bits, L is shuffled with s_i^{-1} . When all the FCS bits are checked, STA_{*i*}'s temporary AID (AID'_{*i*}) is the AID currently stored at location l_i .

Algorithm 1 summarizes the above shuffle mechanism. An example is illustrated in Fig. 2, where $L = [1, 2, 3, 4, 5, 6, 7, 8]$ are shuffled according to the first four bits of a FCS, or '1101' (where the least-significant-bit is ranked first). After conducting three shuffles, we have $L = [6, 1, 7, 4, 3, 5, 8, 2]$, representing the temporary AIDs for STA₁ to STA₈. Assume $m = 4$ and $N_{\text{offset}} = 0$. Apply the temporary AIDs to (1), we get four new subgroups, $\{6, 3\}$, $\{1, 5\}$, $\{7, 8\}$ and $\{4, 2\}$. As compared to the original subgroups (obtained using permanent AIDs), $\{1, 5\}$, $\{2, 6\}$, $\{3, 7\}$ and $\{4, 8\}$, the fixed subgroup problem is solved because each subgroup has a different/randomized set of STAs.

Algorithm 1 AID shuffle mechanism

```

1:  $c \leftarrow 0$ ;  $i \leftarrow 1$ 
2:  $L \leftarrow$  the original ordered AID list
3: while  $i \leq 32$  do
4:   if  $\text{FCS}[i] = 1$  then
5:      $c \leftarrow c + 1$ 
6:     if  $c$  is an odd number then
7:        $L \leftarrow L$  is shuffled with  $s_i$ .
8:     else
9:        $L \leftarrow L$  is shuffled with  $s_i^{-1}$ .
10:    end if
11:  end if
12:   $i \leftarrow i + 1$ 
13: end while
```

IV. RELATED WORK

By shuffling the AIDs in L , our goal is to get a distinct and randomized temporary AID for each STA in each RAW, which is equivalent to generating a random permutation of AIDs in L . The typical way to generate a random permutation is the Knuth shuffle algorithm [10]. This algorithm goes through the list L from locations 1 through $N - 1$. At each location i , we swap the element currently there with a randomly chosen

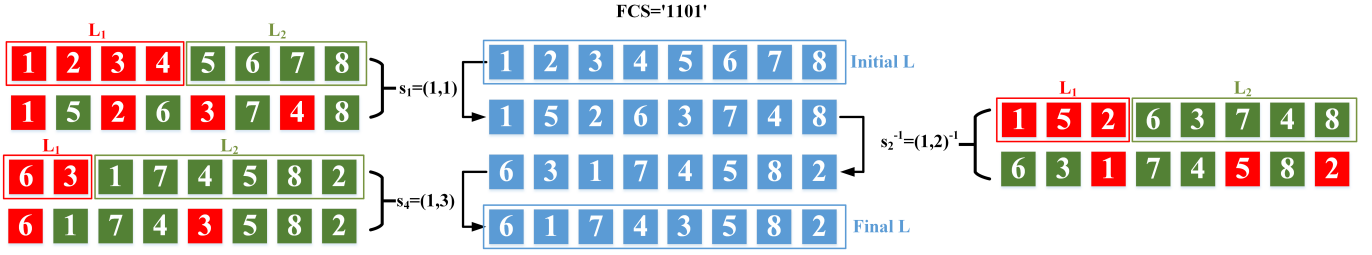


Fig. 2. An example of the proposed FCS-based AID shuffle mechanism.

element from locations i through N , inclusively. It is proved that any permutation of N elements will be produced by this algorithm with probability exactly $1/N!$, thus yielding a uniform distribution over all possible permutations.

Another interesting shuffle mechanism is dovetail, or riffle shuffle. The riffle shuffle is the most commonly used method of shuffling playing cards. The mathematical model of the riffle shuffle is introduced by Gilbert and Shannon [11]. In this model, a deck of N cards is cut into two packs according to a binomial distribution. The two packs are then riffled together in such a way that cards drop from the left or right heap with probability proportional to the number of cards in each heap. Thus, if there are n_1 and n_2 cards remaining in the left and right heaps, respectively, then the chance that the next card will drop from the left heap is $n_1/(n_1 + n_2)$. It is proved by Dave Bayer and Persi Diaconis [12] that, $\frac{3}{2} \cdot \log_2 N + \theta$ shuffles, where θ is a constant, are sufficient to mix up N cards, or each permutation will be produced with probability $1/N!$.

Mark Conger and D. Viswanath further investigated the riffle shuffle problem in Bridge games [13]. In bridge games, there are only 4 players, so it is enough if each player receives a random set of 13 cards. That means the order in which those 13 cards are received by a player is inconsequential. This case is similar to our AID shuffle, since we only need to randomize the set of STAs in each subgroup, while the order of the STAs in a subgroup is not important. In [13], it shows (without a theoretical proof) that only five shuffles are enough to mixing up the 52 cards for bridge games.

While both Knuth shuffle and riffle shuffle are efficient in generating a random permutation, they are based on a probabilistic shuffle model. In Knuth shuffle, at each position, we need to randomly pick a posterior element to swap. In the riffle shuffle, we need to randomly decide how to cut the deck of cards into two packs, and which card should be dropped. Both Knuth and riffle shuffles cannot be directly used in our design because (a) if shuffling is conducted at all STAs independently, due to the probabilistic nature, the final list L obtained by individual STAs will not be consistent, and (b) if shuffling is conducted by AP and the final list L is announced to individual STAs, the consistency issue is solved but IEEE 802.11ah specifications does not provide an announcement mechanism.

Nevertheless, our proposed FCS-based shuffle mechanism

(in Algorithm 1) is inspired by the riffle shuffle. In essence, we turn the probabilistic riffle shuffle into a set of deterministic shuffle patterns. We then use the 32-bit FCS to randomly pick some shuffle patterns to shuffle the AIDs. Since all STAs shuffle the AIDs based on the same FCS, the final list L will be consistent. Simulation results in Section V further show that our FCS-based shuffle can achieve comparable performance as the ideal Knuth shuffle using as little as 16 bits in FCS. Since there are numerous other ways to construct shuffle patterns, it is possible to find shuffle patterns that are more efficient than those in S . We leave that to future research.

V. PERFORMANCE EVALUATION

A. Simulation model

A typical sensor network is dominated by uplink traffic (from STA to AP), where sensors report their status to a server, e.g. the number of cars in a parking lot, temperature, humidity level, etc. Without loss of generality, we only focus on the uplink traffic. In our simulations, we assume STAs are properly grouped [6], [7] such that there is no hidden terminal problem among STAs assigned to the same RAW. We only focus on the target STA group with N STAs, where $N = \text{RAW End AID} - \text{RAW Start AID} + 1$. RAWs with m slots each will be assigned to this target STA group periodically for them to report their status. A STA with a frame to send needs to wake up at every target beacon transmission time (TBTT) and listen to the beacon frame to obtain the RAW allocation information from the RPS element. If the target STA group is allocated a RAW in the current beacon interval, each STA will determine the index of its assigned RAW slot using a specific STA-to-slot assignment.

In our simulations, we adopt an event-driven traffic model which can be usually seen in smart parking systems, thermostat control systems, etc. In these scenarios, sensors will only report its reading when it is triggered by some event, e.g. a car just parked or the temperature just fell below/above a certain threshold. To model this, we assume that during the interval between two consecutive RAWs allocated to the target STA group, a STA (STA_i) generates a new reading/frame with probability p_i . So p_i is the traffic load of STA_i . Since a sensor only needs to report its most up-to-date reading, if an old reading has not been reported (e.g. due to a frame collision in the last assigned slot) when the new one is generated, we

assume the old reading will be overwritten. In other words, each STA can have at most one buffered frame for sending.

When the target STA group is allowed to access the channel in a RAW, all the STAs with buffered frames, or *active* STAs, will compete the channel in their assigned slots. When an active STA receives the beacon frame and calculates the index of its assigned RAW slot, it will go back to sleep and wake up again when its assigned slot starts. According to the specifications [1], active STAs waking up in the same slot will compete for the channel based on the DCF (distributed coordination function) mechanism. Specifically, each STA chooses a backoff timer with its value chosen randomly from $\{0, 1, \dots, CW_{min}\}$ and starts to count down after a distributed inter-frame space (DIFS) time. (In our simulations, $CW_{min} = 15$, the default value in IEEE 802.11 networks (§14.10 in [2]), is used.) A STA will reset its backoff timer at the end of its assigned slot (§10.22.5.5 in [1]). In other words, a STA will invoke a new backoff procedure for accessing the channel in each RAW slot assigned to it. If the channel is sensed idle, the timer will count down. If the channel is sensed busy (due to other transmissions), the timer will be frozen until the channel is sensed idle again. When the timer counts down to zero, the corresponding STA will immediately send its buffered frame. Since we have assumed there are no hidden terminals in our target STA group and the STAs are perfectly synchronized, frame collision can only occur if two or more STAs choose the same backoff timer and simultaneously transmit when their timers expire.

We assume the uplink data frames are of identical size and are transmitted with the same PHY data rate. Let T_{data} be the transmission time of a data frame and T_{ACK} be the transmission time of an ACK frame. The duration of each RAW slot is set as

$$T_{slot} = \text{DIFS} + CW_{min} \cdot aSlotTime + T_{data} + \text{SIFS} + T_{ACK},$$

where $aSlotTime$ is the system slot time and SIFS is the short interframe space (§9.3.7 in [2]). That means the slot duration is just enough for sending one data frame. If a collision occurs in a slot, the remaining slot duration will not be enough for another transmission attempt. (Note that the AID shuffle mechanism proposed in this paper works with any slot size.) When a slot ends, all active STAs assigned to the slot will go back to sleep and wake up again at next TBTT. An example of channel access is shown in Fig. 3, where 4 STAs are partitioned into two equal subgroups and each subgroup is mapped to a specific slot. In slot 1, STA_a sends a frame successfully. In slot 2, a frame collision occurs.

According to the transmission status, a slot can be classified as empty (i.e., no active STAs), collided (such as slot 2 in Fig. 3) or successful (such as slot 1 in Fig. 3). Let $P(\text{slot}_{succ})$ be the probability of having a successful slot, or *slot utilization*. $P(\text{slot}_{succ})$ is determined by the number of active STAs in a slot. If there are no active STAs, $P(\text{slot}_{succ}) = 0$. If there are too many active STAs, $P(\text{slot}_{succ}) \approx 0$ because the

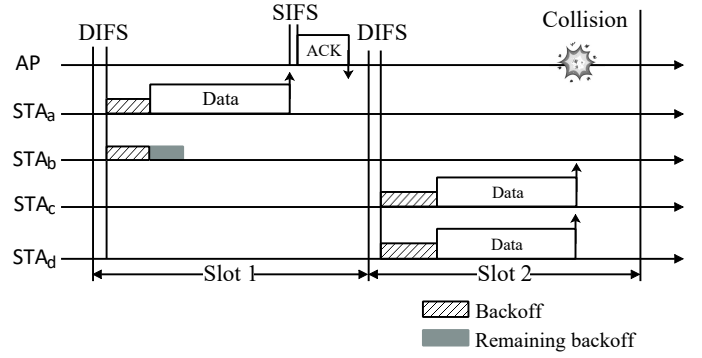


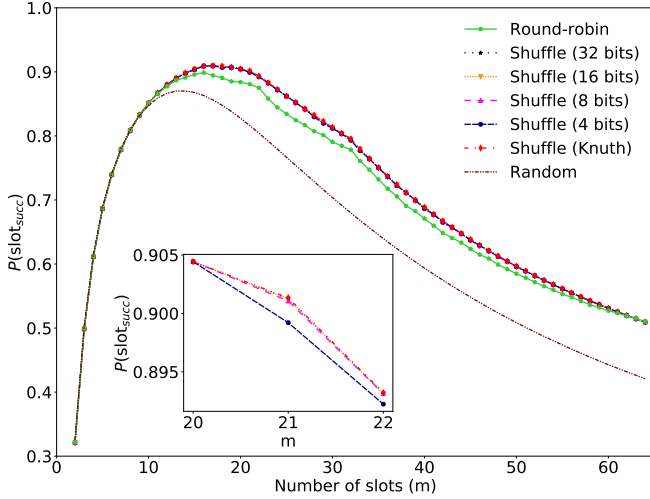
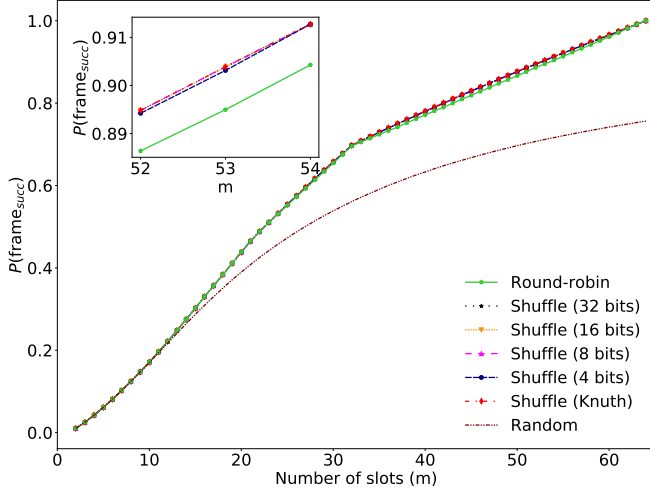
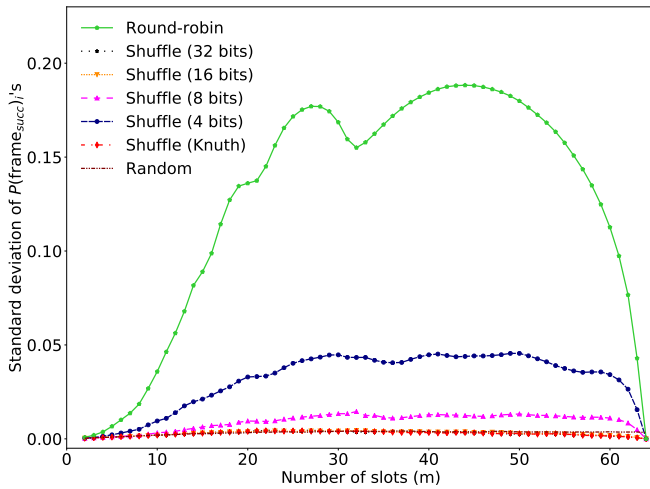
Fig. 3. An example of channel access in RAW slots.

collision probability will be so high that no frame can get through. Note that, for a given number of STAs (N) and traffic load (p_i), the number of slots in a RAW (m) can be designed to maximize $P(\text{slot}_{succ})$. Although finding the optimal value of m is interesting, we focus on maximizing $P(\text{slot}_{succ})$ when m is given. When $P(\text{slot}_{succ})$ is maximized, so is the system throughput.

For each STA_{*i*} in the target STA group, we define $P(\text{frame}_{succ})_i$ as the probability that a frame is sent successfully by *active* STA_{*i*} in its assigned slot, e.g. STA_{*a*} in Fig. 3. To provide fair channel access, our goal is to have the same $P(\text{frame}_{succ})_i$ for all STAs. As expected, if STA_{*i*} always needs to compete with many other active STAs, $P(\text{frame}_{succ})_i \approx 0$ because of high collision probability. On the other hand, if STA_{*i*} is the only active STA in an assigned slot, $P(\text{frame}_{succ})_i = 1$. A trivial way of getting $P(\text{frame}_{succ})_i = 1 \forall i$, is to have a dedicated slot for each STA, or $N = m$. But the slot utilization, given by $P(\text{slot}_{succ}) = \frac{\sum_i p_i}{N}$, can be very low (as we will see in next subsection). Besides, a large m implies less channel time left for STAs *not* belonging to the target STA group.

Based on above definitions and analysis, three performance metrics are adopted to evaluate the performance of STA-to-slot assignment schemes, average slot utilization ($P(\text{slot}_{succ})$), average frame successful probability ($P(\text{frame}_{succ}) = \sum_i P(\text{frame}_{succ})_i / N$) and the standard deviation (STD) of $P(\text{frame}_{succ})_i$'s among different STAs. $P(\text{slot}_{succ})$ provides the system throughput performance, $P(\text{frame}_{succ})$ provides insight to STAs' transmission efficiency, and STD measures the transmission fairness among STAs. In our simulations, $P(\text{slot}_{succ})$ is obtained by dividing the total number of frames successfully sent by the total number of slots simulated. $P(\text{frame}_{succ})_i$ is obtained by keeping two counters for each STA_{*i*}, X_i and Y_i . When STA_{*i*} is active in a slot, $X_i = X_i + 1$. When STA_{*i*} has successfully sent a frame, $Y_i = Y_i + 1$. We have $P(\text{frame}_{succ})_i = Y_i / X_i$.

In our simulations, three STA-to-slot mapping mechanisms are simulated and compared, *Round-robin* the original round-robin assignment given by (1), *Shuffle* the round-robin assignment with our FCS-based AID shuffle mechanism, and *Random* the random assignment presented in Section II. To

(a) Average slot utilization $P(\text{slot}_{succ})$.(b) Average frame successful probability $P(\text{frame}_{succ})$.(c) Standard deviation of $P(\text{frame}_{succ})_i$'s.Fig. 4. Performance of different STA-to-slot mapping mechanisms with $N = 64$ STAs and varying number of RAW slots m .

study the efficiency of our FCS-based shuffle mechanism with different number of bits in FCS, we specify the number of bits used in the bracket, e.g. *Shuffle (8 bits)*. We also implement the ideal Knuth shuffle algorithm, *Shuffle (Knuth)*, as a benchmark. In simulation results, each data point is obtained by averaging over 100 simulation runs. In each simulation run, 100,000 RAWs are allocated to the target STA group to get a stable performance. At the beginning of each simulation run, each STA_i chooses its traffic load p_i from the interval $(0, 1)$ uniformly at random. To implement our FCS-based AID shuffle mechanism, the 32-bit FCS is generated randomly.

B. Simulation results

Fig. 4 compares the performance of different STA-to-slot mapping mechanisms, with $N = 64$ STAs and varying number of RAW slots (m). In Fig. 4a, slot utilization ($P(\text{slot}_{succ})$) is plotted against m . Note that the average traffic load in each (RAW) slot reduces with m . With $m < 10$, each slot is overloaded and all schemes provide comparable throughput performance due to frequent collisions. With $m \geq 10$, the performance is no longer dominated by collisions and we can see that random assignment (*Random*) provides the poorest performance because of the large fluctuation in the number of active STAs assigned to each slot in the same RAW. With both *Round-robin* and *Shuffle*, the number of STAs assigned to each slot is balanced and thus a higher slot utilization can be achieved. Since *Shuffle* solves the fixed subgroup problem with *Round-robin*, better load balancing performance warrants a higher slot utilization. In the close-up window, we can see that with more bits in FCS being used, a slightly higher slot utilization can be obtained. Since such gain is so small, it may be more beneficial to use less FCS bits for a simpler implementation. When $N = m = 64$, each STA is assigned a dedicated slot if *Round-robin* or *Shuffle* is used. In this case, $P(\text{slot}_{succ}) = \sum_i p_i / N \approx 0.5$. Again, random assignment gives the worst performance because it cannot ensure each STA is assigned a dedicated slot.

In Fig. 4b, the average frame successful probability ($P(\text{frame}_{succ})$), or the probability an active STA can have its frame sent in an assigned slot, is provided. As expected, $P(\text{frame}_{succ})$ increases with m because less active STAs will compete in a slot. (That said, slot utilization reduces with m in Fig. 4a.) As compared with the original round-robin assignment, our shuffle mechanism can slightly improve the transmission efficiency. Among the three types of mapping functions, random assignment has the lowest frame successful probability, which is again due to the unbalanced number of active STAs in each slot of the same RAW.

Fig. 4c plots the standard deviation (STD) of frame successful probability ($P(\text{frame}_{succ})_i$) among all STAs against m . We can see that *Round-robin* has the highest STD, or poorest fairness performance, which is due to the fixed subgroup problem in the original round-robin assignment. With our FCS-based AID shuffle mechanism, the performance of *Shuffle* has improved significantly, especially when more bits in FCS

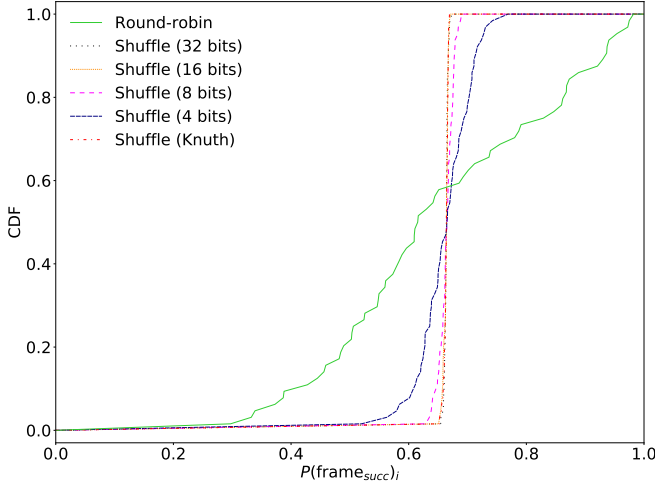


Fig. 5. The CDF of frame successful probabilities ($P(\text{frame}_{\text{succ}})_i$) with $m = 30$ and $N = 64$.

are used for shuffling. In particular, the fairness performances of *Shuffle(16 bits)* is comparable to the ideal performance of *Shuffle(Knuth)* and *Random*. Note that *Random*'s excellent (long-term) fairness performance is at the cost of low slot utilization/throughput in Fig. 4a. Further note that when $m = 1$, i.e., single slot per RAW, all mapping functions have the same ideal STD performance. On the other hand, when $m = 64$, if *Round-robin* or *Shuffle* is used, each STA is assigned a dedicated slot. We thus have $P(\text{frame}_{\text{succ}})_i = 1 \forall i$, and *Round-robin* and *Shuffle* converge to the ideal STD performance. But since random assignment cannot ensure each STA assigned to a dedicated slot when $m = 64$, *Random* cannot get a STD of 0.

To take a closer look at the transmission fairness among different STAs, in Fig. 5, we plot the cumulative distribution function (CDF) of frame successful probabilities ($P(\text{frame}_{\text{succ}})_i$) experienced by all STAs with $m = 30$ and $N = 64$. With our proposed FCS-based shuffle mechanism, when the first 16 or 32 bits of FCS are used to shuffle the AIDs, the step-shaped CDF shows that most STAs have comparable $P(\text{frame}_{\text{succ}})_i$. From our simulation results, we can further see that with *Round-robin*, the minimum and maximum values of $P(\text{frame}_{\text{succ}})_i$ are 0.297 to 0.982. With our *Shuffle(16 bits)*, they become 0.651 and 0.671, where the gap is reduced significantly.

Note that, the total number of permutations of AIDs ($N!$) increases exponentially with the number of STAs (N). It would be more difficult to mix up the AIDs when N is large, e.g. in riffle shuffle, we will need more shuffles to mix up the cards. In Fig. 6, we plot the STD/fairness performance of our *Shuffle* against the number of FCS bits used for shuffle, and varying N from 64, 128 to 256. To ensure the same number of (arbitrarily chosen) four STAs per subgroup/slot as N increases, m is increased from 16, 32 to 64. (Note that the maximum allowed number of slots in a RAW is 64.) From the figure, when the number of FCS bits is less than 16, a larger N leads to a higher

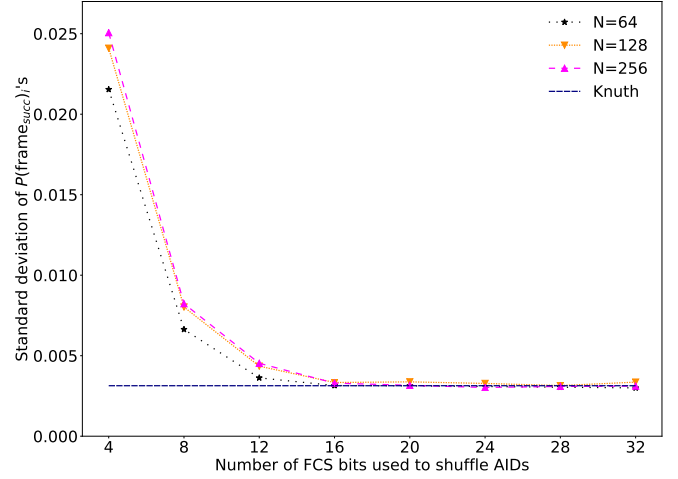


Fig. 6. Standard deviation of $P(\text{frame}_{\text{succ}})_i$'s versus the number FCS bits used with $N/m = 4$.

STD value, or poorer transmission fairness performance. But when the number of FCS bits increases to 16, in all cases, *Shuffle* converges to the ideal performance of *Shuffle(Knuth)*, with a STD below 0.0031. This shows that our FCS-based AID shuffle mechanism is robust with the number of STAs assigned to a RAW.

VI. CONCLUSION

To implement the new restricted access window (RAW) medium access control mechanism in IEEE 802.11ah, stations (STAs) must be assigned to RAW slots based on their association IDs (AIDs). In this paper, we first identified the fixed subgroup problem with the existing round-robin STA-to-slot assignment. We then proposed a novel AID shuffle mechanism to randomize the set of STAs assigned to each subgroup/slot. The idea is to generate a different and distinct temporary AID for each STA in each RAW, based on a set of judiciously designed shuffle patterns and the frame checksum bits of the beacon frame. By reusing the original round-robin assignment with its temporary AID, each STA finds its assigned slot. Our simulation results showed that our FCS-based AID shuffle mechanism solves the fixed subgroup problem, and improves transmission fairness among STAs and the system throughput.

REFERENCES

- [1] "IEEE Standard for Information technology–Telecommunications and information exchange between systems - Local and metropolitan area networks–Specific requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Amendment 2: Sub 1 GHz License Exempt Operation," *IEEE Std 802.11ah-2016*, pp. 1–594, April 2017.
- [2] "IEEE 802.11-2012: Wireless LAN Medium Access Control MAC and Physical Layer PHY Specifications," *IEEE 802.11 LAN Standards*, 2012.
- [3] L. Zheng, M. Ni, L. Cai, J. Pan, C. Ghosh, and K. Doppler, "Performance analysis of group-synchronized DCF for dense IEEE 802.11 networks," *IEEE Transactions on Wireless Communications*, vol. 13, no. 11, pp. 6180–6192, Nov 2014.
- [4] R. Porat, S. Yong, and K. Doppler, "TGah channel model," <https://mentor.ieee.org/802.11/dcn/11/11-11-0968-04-00ah-channel-model-text.docx>, 2015.

- [5] M. Park, "IEEE 802.11ah: Energy efficient MAC protocols for long range wireless LAN," in *2014 IEEE International Conference on Communications (ICC)*, June 2014, pp. 2388–2393.
- [6] M. Ghasemianmadi, Y. Li, and L. Cai, "Rss-based grouping strategy for avoiding hidden terminals with gs-dcf mac protocol," in *2017 IEEE Wireless Communications and Networking Conference (WCNC)*, March 2017, pp. 1–6.
- [7] Z. Zhu, Z. Fan, and F. Zhong, "A station regrouping method for contention based IEEE 802.11ah wireless LAN," in *2017 IEEE 13th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, 2017.
- [8] T. C. Chang, C. H. Lin, K. C. J. Lin, and W. T. Chen, "Load-Balanced sensor grouping for IEEE 802.11ah networks," in *2015 IEEE Global Communications Conference (GLOBECOM)*, Dec 2015, pp. 1–6.
- [9] P. Diaconis, R. L. Graham, and W. M. Kantor, "The Mathematics of Perfect Shuffles," *Advances in Applied Mathematics*, vol. 4, pp. 175–196, 1983.
- [10] D. E. Knuth, *The Art of Computer Programming, Volume 2 (3rd Ed.): Seminumerical Algorithms*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1997.
- [11] E. N. Gilbert, "Theory of shuffling," Bell Laboratories, Technical memorandum, 1955.
- [12] D. Bayer and P. Diaconis, "Trailing the Dovetail Shuffle to its Lair," *The Annals of Applied Probability*, vol. 2, pp. 294–313, 1992.
- [13] M. Conger and D. Viswanath, "Shuffling cards for blackjack, bridge, and other card games," pp. 1–27, 2006. [Online]. Available: <http://arxiv.org/abs/math.PR/0606031>