Rhythm: Achieving Scheduled WiFi Using Purely Distributed Contention in WLANs

Chao-Fang Shih, Bhuvana Krishnaswamy, and Raghupathy Sivakumar
Georgia Institute of Technology, Atlanta, GA, USA
{cshih,bhuvana,siva}@ece.gatech.edu

***Abstract*—The ubiquitous adoption of WiFi implicitly introduces large diversity in types of application requirements and topological characteristics. Consequently, considerable attention is being devoted to making WiFi networks controllable without compromising their scalability. Within this broad paradigm, we propose Rhythm, a MAC protocol that achieves scheduled WiFi efficiently and that is subject to the following constraints: (i) It does not need fine-grained time synchronization, (ii) it adds no "active listening" time, (iii) it does not need to gather the queue status from clients, and (iv) it requires no additional hardware. It also has the following properties: (i) low overhead, (ii) work conservation, (iii) robustness to partial connectivity, and (iv) backward compatibility.**

I. Introduction

WiFi has become de-facto "zeroth"-mile connectivity in a wide-swathe of Internet settings. High data-rate achievability, cheap deployability, and near-universal availability on mobile devices have all contributed to its tremendous ubiquity. WiFi is routinely used today in many environments that exhibit diversity both in terms of service requirements and topological characteristics. Therefore, researchers are devoting considerable attention to *improving WiFi network controllability without compromising its scalability.* WiFi controllers that centralize the management of access points (APs) within an enterprise network may be the simplest example of such efforts. Functionalities such as radio resource management, wireless policy management, and authentication services are moved to the central controller. However, while WiFi controllers allow for configurability, they do so only for macro-level "control plane" parameters such as authentication settings and channel assignments, and they do not control "data plane" functions such as media access control.

The media access control (MAC) protocol used by WiFi is an important function with regard to controllability and scalability. The distributed MAC protocol has many benefits, including low complexity, scalability, and robustness to dynamic traffic loads. However, it is not controllable, and its efficiency decreases as contention levels increase. Some approaches improve efficiency by optimizing the contention window of the distributed coordination function (DCF) [1], [2]. However, without centralized network-wide information, the resulting efficiency does not achieve what a MAC protocol with centralized information achieves.

While distributed MAC protocols are less controllable and often have low efficiency, centralized MAC protocols [3] are more controllable and enable cooperation among networks. However, such approaches require fine-grained time synchronization, which is not practical in many situations [4]. They also introduce high overhead or inefficiency when collecting micro-level network-wide information such as the queue status of clients (e.g., the polling mechanism in the 802.11 PCF [5]).

Thus, this paper focuses on how to *enable the controllability of WiFi networks without compromising their scalability*. To achieve this goal, we introduce a MAC protocol called "*Rhythm*". Specifically, Rhythm furnishes all nodes in WiFi networks with a *target schedule* that has been determined by a central entity. The nodes in the networks then operate in a *purely distributed fashion* to meet the target schedule. We refer to such a network behavior as "*scheduled WiFi*."

The specific contributions of our work are two-fold:

- We present Rhythm, a solution for scheduled WiFi. We show that Rhythm enables scheduled WiFi subject to the constraints of (i) no fine-grained time synchronization, (ii) no additional "active listening" time, (iii) no need for gathering the queue status from clients, and (iv) no additional hardware; and with the properties of (i) low overhead, (ii) work conservation in the presence of non-backlogged nodes, (iii) robustness to partial connectivity scenarios, and (iv) backward compatibility with non-Rhythm nodes in the environment.
- We evaluate the performance of Rhythm using simulations and real-time experiments carried out in a software defined radio platform: the Wireless Open-Access Research Platform (WARP) [6].

This paper is not the first to introduce the notion of scheduled WiFi. A MAC protocol, Domino [4], is perhaps closest to the approach used in this paper and best exemplifies an approach to achieving a schedule without fine-grained time synchronization. We detail the differences between Rhythm and Domino later in the paper and compare their performance. Briefly, Rhythm is more practical with minimal software-only changes to WiFi implementation. It does not change WiFi frame formats, and requires much less operating overheads, enhancing performance and fairness.

The rest of the paper is organized as follows: Section II presents the problem definition and Section III the design of the Rhythm solution. Section IV evaluates Rhythm, Section V presents related works, and Section VI concludes the paper.

This work was supported in part by the National Science Foundation under grants IIP-1343435 and CNS-1319455.

II. Problem

Let us consider multiple infrastructure WLANs, containing n nodes, all using the same channel. In these WLANs, a central controller communicates with all APs. After gathering network-wide information (only macro-level information such as client lists from APs, not the queue status, are gathered), the central controller determines a target schedule $S = \{s_0, \cdots, s_{k-1}\}$, where $s_i \in N = \{0, \cdots, n-1\}$ indicates the scheduled node ID in position i. (Note that the length of schedule k can be larger than the number of nodes n if a node is scheduled multiple times.) S is delivered to all nodes through the APs. *With S, how can nodes in these networks efficiently contend distributedly (without any other communication with the central controller) to follow the transmission order in S?*

At a high-level, Rhythm involves changes to the contention mechanism in the 802.11 DCF. A Rhythm node listens to ongoing transmissions and determines a *virtual schedule pointer* within S. It then contends based on the relative distance between the virtual pointer and its position within S. The non-trivial aspects of the Rhythm solution lie in how *work conservation* is achieved when it is in the presence of non-backlogged nodes (Note that the queue status is unknown to the central controller) and how *partial connectivity* (i.e., nodes cannot overhear each other) and other practical challenges such as *backward compatibility* are handled.

In the following section, we start with the simplest scenario and consider each challenge gradually.

III. Rhythm: Scheduled-WiFi using Distributed Contention

A. Baseline algorithm

In a scenario in which all nodes are in transmission range of one another (i.e., a fully connected topology) and all nodes are usually backlogged, if nodes are to follow the target schedule, we introduce a logic concept: the virtual schedule pointer.

1) Virtual schedule pointer: To follow a common schedule, nodes must be synchronized. Although the most straightforward way is time synchronization, fine-grained time synchronization can be difficult. Thus, Rhythm achieves scheduled WiFi by synchronizing nodes in a "logical schedule position" in which each node in the network maintains a virtual schedule pointer, Pos, that points to the current logical position in the schedule. If the values of Pos in all nodes are the same (i.e., they are synchronized), the nodes follow the target schedule by adjusting the backoff number. For example, let's assume that three nodes X, Y, and Z and a target schedule $S = \{s_0 = X, s_1 = Y, s_2 = Z\}$ initially do a random backoff (the same way as DCF), and one node (assuming Y) wins the contention and transmits successfully. After hearing the successful transmission of Y, all nodes (including Y) update the virtual schedule pointer to 1. Then, instead of continuing the random backoff, Z sets its backoff number to 0 by calculating $D = pos(Z) - Pos = 2 - 1 = 1$, where $pos(Z)$ is the scheduled position of Z, and uses $((D-1) \bmod k)$ as its backoff number. Similarly, X sets its backoff number to 1, and Y sets its backoff number to 2. If Z transmits, X resets its backoff number to 0 after overhearing the transmission of Z ($Pos = 2$, $pos(X) - Pos - 1 = 0 \bmod 3$). If Z does not transmit, X starts transmission when its backoff timer expires (X only waits 1 backoff slot). Such a mechanism allows nodes to follow the target schedule without extra time synchronization. Employing this mechanism, we develop a baseline algorithm of Rhythm: Rhythm-Base.

2) Algorithm: We assume that target schedule $S = \{s_0, \cdots, s_{k-1}\}$ is known to all nodes in the network. All nodes maintain synchronization state ST and record $RC = \{r_0 \in N \bigcup \{Col\}\}$, where r_0 represents the most recent transmission, and Col represents a collision. Algorithm 1 illustrates position synchronization (UpdatePos) and schedule matching (MatchSch) of Rhythm-Base. Initially, every node does random backoff when RC is empty. Upon a successful transmission, the nodes update Pos to a position in S that matches RC. The update mechanism depends on the current synchronization state ST, which has two states: RAN and SYN. If $ST == RAN$, which indicates lack of synchronization before this transmission, the nodes update Pos to *the smallest* matched position in S. If $ST == SYN$, nodes start from previous Pos and update Pos to the nearest matched position in S. After updating, each node sets the backoff number to $((D-1) \bmod k)$, where $D = pos_{nearest}(s_{self}) - Pos$ and $pos_{nearest}(s_{self})$ is the nearest scheduled position of each node. If a collision occurs, ST is reset to RAN, and the nodes perform a random backoff, just as they did in DCF. If a node hears a new transmission before the backoff timer expires, it updates Pos and resets the backoff number. Otherwise, when the backoff timer expires, the node transmits, records its own transmission, and updates Pos.

Rhythm-Base is simple, proffers weighted fairness, as indicated in the target schedule, and yields near optimal channel utilization: i) It wastes only 1 backoff slot (9μs in 802.11g) if a scheduled node does not transmit, ii) it causes no collisions when all nodes are synchronized, iii) it requires only one successful transmission for convergence in Pos. (Matching transmissions to the smallest position after collision avoids ambiguity when some nodes are scheduled multiple times.) In addition, because of its fast-convergence property, Rhythm-Base is robust against disturbances such as the packet error or loss of ACKs (both recorded as Col). Note that although Rhythm utilizes overheard transmissions, the active listening time is the same as that of DCF.

3) Overhead estimation: The use of Rhythm-Base introduces two overheads: *i) The overhead of broadcasting the target schedule:* Every node needs to know the target schedule S. S can be placed into a beacon every few seconds. The extra time for sending S is $T_s = \frac{k \times l}{R_b}$, where l is the length of the MAC address, k the schedule length, and R_b the sending rate of beacons. The overhead is $O_s = \frac{T_s}{T_p}$, where T_p is the period of updating the target schedule. If $R_b = 6Mbps$, $k = 100$, and $T_p = 100ms$, the overhead of broadcasting the target schedule is only $O_s = 0.8\%$, which is very low even when we have a long schedule and update it frequently. *ii) The overhead of re-synchronization resulting from packet error and collision:* Assuming P_{col} is the probability of having a packet collision when n nodes contend using DCF, and P_{err} is the packet error rate, the overhead caused by packet errors and collisions in DCF is $O_e = (1 - P_{col})P_{err} + P_{col}$. In Rhythm, 1 successful transmission can gain synchronization and avoid collisions, so the overhead in Rhythm is $O_e =$

Algorithm 1 Rhythm-Base

```
1:  function UpdatePos
2:      if r_0 ==Col || |RC| == 0 then
3:          set RC = {Col}
4:          ST = RAN
5:          return, and do regular random backoff as DCF
6:      else if ST == RAN then
7:          Pos=MatchSch(-1)
8:      else
9:          Pos=MatchSch(Pos)
10:     end if
11:     set RC = {r_0}                                  ▷ clear old record
12:     ST = SYN
13:     set backoff_number=(pos_nearest(s_self) − Pos − 1) mod k
14: end function

15: function MatchSch(prevPos)
16:     j = (prevPos + 1) mod k
17:     while True do
18:         if s_j == r_0 then
19:             return j
20:         end if
21:         j = (j + 1) mod k
22:     end while
23: end function
```

$P_{err} \times (1-P_{col})(1+P_{col} \times 2+P_{col}^2 \times 3+\cdots) \leq P_{err} \times \frac{1}{1-P_{col}}$. If $P_{err} = 1\%$ and $P_{col} = 15\%$, then $O_e = 15.85\%$ in DCF and $O_e \leq 1.18\%$ in Rhythm.

4) Limitations of the baseline algorithm: Although Rhythm-Base yields near optimal channel utilization with low overhead, it cannot perform well in the following situations:

- Non-backlogged nodes lead to inefficiency: In Rhythm-Base, 1 backoff slot ($9\mu s$) is wasted when a scheduled node does not transmit. When many nodes are continuously non-backlogged, the performance of Rhythm-Base might be worse than that of DCF.
- Partial connectivity leads to a lack of information: Rhythm-Base achieves position synchronization by overhearing transmissions. Thus, if nodes are located outside the transmission range of the other nodes (i.e., hidden terminals), they do not overhear transmissions, and Rhythm-Base may not achieve position synchronization.

In the following subsections, we propose mechanisms that deal with these situations.

B. Work conservation with non-backlogged nodes

Non-backlogged nodes generate idle slots, leading to inefficiency. Instead of informing the central controller to change the target schedule, which generates extra overhead and requires small delay between central entity and APs, we introduce a *Shrinking mechanism* in Rhythm (Rhythm-Shrink), in which nodes distributedly adjust the schedule.

1) Schedule shrinking for non-backlogged nodes: Each node maintains a *non-backlogged record*: If some scheduled nodes do not transmit, they are deemed *non-backlogged*. Non-backlogged nodes are ignored in backoff calculation and schedule-matching algorithms; that is, the schedule shrinks among these non-backlogged nodes. Schedule shrinking reduces idle slots and thus avoids the inefficiency caused by non-backlogged nodes. However, when nodes receive new packets, how do they get back on schedule? One simple way is to generate a collision that stops schedule shrinking. However, if 30 nodes are non-backlogged and only one of them receives

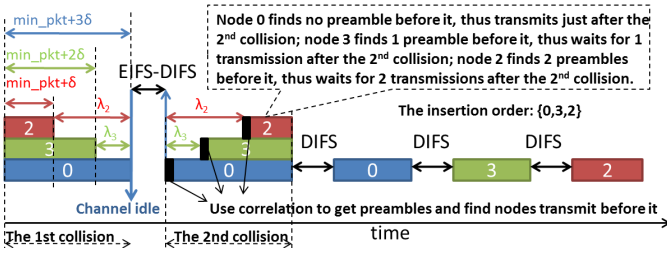


Fig. 1. Time line of mirror insertion

new packets, a collision and 29 idle slots ($261\mu s$) is generated every time a node gets back on schedule. Thus, we introduce an efficient insertion mechanism that reinserts nodes on the schedule.

2) Mirrored collisions during insertions: If non-backlogged nodes exist, 1 backoff slot, referred to as an *insert slot*, is left at the end of each schedule cycle for these nodes to win the contention and start a transmission, which stops schedule shrinking among the nodes. If only one node is inserted, it is recorded as backlogged when its transmission is heard. *However, how can several non-backlogged nodes be simultaneously inserted with only 1 backoff slot?* We introduce "*mirror insertion*," which generates two customized collisions for efficiently inserting multiple nodes simultaneously.

The main goal of mirror insertion is to determine an instant *insertion order* at which point all inserting nodes will be inserted so that no time is wasted waiting for non-backlogged nodes that are not inserting. Fig. 1 illustrates the time line of mirror insertion. When nodes want to insert, they transmit a packet of a special length (achieved by packet splitting and aggregating) in the base rate. The length is $min\_pkt + (n - I) \times \delta$, where min_pkt is a predefined minimum packet size known by all nodes, δ the sum of the preamble transmission time and the transmission switching delay, and I the order of their first scheduled position in S. (For example, if $S = \{0, 3, 1, 3, 2\}$, the order of the first scheduled position is $\{0, 3, 1, 2\}$, and the insert packet length is min_pkt for node 2, and $min\_pkt + 3 \times \delta$ for node 0.) When multiple nodes are inserted simultaneously, a collision occurs. Each inserting node records λ_i ($i \in N$), the time from the end of its transmission to the time when the channel becomes idle. Then, each inserting node transmits at $EIFS - DIFS + \lambda_i$ after the channel becomes idle, creating a second collision, a "mirror image" of the first. After the first collision, all inserting nodes listen to the channel before transmitting. Since the start of each transmission is separated by at least δ, each node is able to hear the preambles of all prior transmissions before starting its own. Each node then applies preamble correlations to the heard signal and figures out how many nodes start transmitting before it start to transmit during the second collision. This transmission order determines the insertion order for all inserting nodes, as illustrated in Fig. 1.

The key mechanism of mirror insertion is *identifying preambles in collision using correlation*. A preamble is a pseudo-random sequence that can be identified using correlation, even under high interference. Studies have proven this mechanism valid with multiple random collided packets [7]. Since mirror collision is designed in a way in which preambles are separated in the integer number of δ (we use $26\mu s$ when

Algorithm 2 Backoff Calculation of Rhythm-Shrink

```
1: function CALBK(Pos)
2:     if IsInserting==False then                          ▷ node is not inserting
3:         Sch_Pos=pos_nearest(s_self)
4:     else if IsInserting==True then                      ▷ node is inserting
5:         if InstST==Before_Mirror_Coll then              ▷ before mirror collisions
6:             Sch_Pos=k
7:         else if InstST==After_Mirror_Coll then          ▷ after mirror collisions
8:             BK =(n_front_insert)-(n_insert)
9:             return BK
10:        end if
11:    end if
12:    BK = (Sch_Pos−Pos − 1) mod k;
13:    BK = BK−NonBkLogCount(Pos,Sch_Pos)   ▷ skip non-backlogged nodes
14:    if Sch_Pos≤ Pos & NonBkLogCount(0,k)> 0 then
15:        BK++
16:    end if
17:    return BK
18: end function
19: function NONBKLOGCOUNT(p1,p2)
20:    return the number of non-backlogged nodes between position p1 and p2
21: end function
```

the preamble transmission time is 16μs), its identification is even easier.

Mirror insertion does not require fine-grained time synchronization. As long as the preambles in the second collision are separated far enough for identification, an insertion order is determined. Since mirror collisions are two collisions within $EIFS$, they are easily recognized, and are not treated as normal collisions.

3) Algorithm: As the position synchronization and schedule-matching algorithm of Rhythm-Shrink closely resembles that of Rhythm-Base (requiring only skipping mirror collisions when matching and setting $Pos = k$ during insertion), this work omits it.[1] Algorithm 2 illustrates the backoff calculation of Rhythm-Shrink. When a non-inserting node calculates the backoff number, it skips non-backlogged nodes (line 13) and adds an insert slot if required (line 14 to 15). After creating mirror collisions, inserting nodes learn the insertion order, continue counting the insertion transmissions, and set up the backoff number accordingly (line 8). Since $Pos = k$ during insertion, non-inserting nodes will continue adding an extra backoff slot (line 14 to 15), and inserting nodes win the contention after creating mirror collisions.

4) Overhead estimation: Mirror collisions result in the main overhead of the shrinking mechanism: $O_{mirr} = \frac{2\times T_{col}\times P_{insrtCol}}{\lceil \frac{T_{inact}}{T_{sch}} \rceil \times T_{sch}}$, where $T_{col} = T_{min\_pkt} + (n - I_{avg}) \times \delta$ is the time spent in each mirror collision, I_{avg} is the average minimum first scheduled position of the insert nodes, $P_{insrtCol}$ is the probability of having more than two inserting nodes, T_{inact} is the average period in which a node runs out of packets, T_{sch} is the average period of the target schedule, and T_{min_pkt} is the transmit time of a packet with size min_pkt.

T_{col} slightly increases as the number of node increases: If $T_{min\_pkt} = 52\mu s$, $\delta = 26\mu s$, $n = 50$, $I_{avg} = 25$, $P_{insrtCol} = 1$ and $\lceil \frac{T_{inact}}{T_{sch}} \rceil \times T_{sch} = 50ms$, we have $O_{mirr} = 2.81\%$. If $n = 300$, $I_{avg} = 150$, we have $O_{mirr} = 15.81\%$. Compared to 40%, which results from collisions when DCF is used with $n = 300$, this value is still reasonable. The traffic dynamic and the schedule period also affect O_{mirr}. If $T_{sch} = 200ms$ or $T_{inact} = 200ms$, $O_{mirr} = 3.95\%$ in the $n = 300$ case.

In summary, the overhead grows only when the three following conditions are satisfied: i) the number of nodes is large, and ii) traffic is strongly dynamic, and iii) the average schedule period is short. Because it is unlikely to have both short schedule period and a large number of nodes, Rhythm can yield near optimal channel utilization in most practical situations with various traffic loads.

[1]More details can be found in [8].

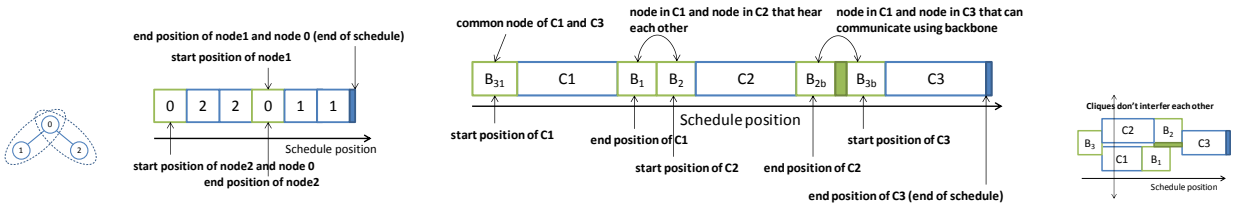
C. Handling partial connectivity

Partial connectivity hinders position synchronization in Rhythm. Instead of transmitting RTS/CTS, which generates large overhead, we introduce a *clique mechanism* in Rhythm (Rhythm-Clique).

1) Separation and connection: The main concept of the clique mechanism is *separating nodes that cannot overhear each other into different groups*. Nodes inside the same group overhear each other. Thus, each group can operate Rhythm-Shrink by itself. Then, we use *nodes that can hear different groups to connect groups.* We illustrate this concept using an example in Fig.2(a), where nodes 1 and 2 cannot hear each other. The target schedule separates the two nodes and connect them using node 0: $S = \{0, 2, 2, 0, 1, 1\}$. The start and end positions of each group are the positions of the adjoining connecting nodes (indicated in Fig. 2(b)). When Pos points to positions inside a group (referred to as *active positions*), nodes in the group operate Rhythm-Shrink. Otherwise, they set up a long backoff. After the first successful transmission of node 0, all nodes have $Pos = 0$. Node 1 sets up a long backoff and node 2 transmits twice. Then, node 0 transmits again, which updates $Pos = 3$. Node 2 sets up a long backoff and node 1 transmits twice. Then, node 0 transmits and updates $Pos = 0$ again. Repeatedly, the nodes follow the schedule even when they cannot overhear each other. Based on the same concept, we describe the clique mechanism below.

2) Cliques and bridges: The clique mechanism involves a target schedule that separates nodes that cannot overhear each other into different portions, or cliques, of the schedule (all nodes inside a clique overhear each other) and places "bridges" to connect the cliques. The start and end positions of a clique are the positions of the adjoining bridges, which define the *active positions* of a clique. Fig. 2(c) shows a schedule with three cliques, C_i, four bridges, B_j, and start/end positions.

To trigger Pos updates, bridges should always transmit, and the transmissions of bridges must be heard by their adjoining cliques. Accordingly, bridges can be set up in three ways: *i) Using a node that hears both cliques:* A node that hears both cliques can act as a bridge connecting them; *ii) using nodes from both cliques that hear each other:* shown in Fig. 2(c); after overhearing the transmission of B_1, the Pos update of $C1$ is triggered, and B_2 transmits and triggers the Pos update in $C2$; *iii) using nodes from both cliques that can communicate with each other using the backbone:* Some nodes can communicate using a backbone connection (such as APs). In Fig. 2(c), when B_{2b} transmits, it sends a packet containing the transmission time of its wireless transmission to B_{3b} through the backbone. Based on information carried in this packet, B_{3b} learns the end time of transmission of B_{2b}, and transmits to trigger the Pos update in $C3$. We can extend the concept of the clique mechanism to scheduling multiple cliques that do not interfere with each other in parallel (Fig. 2(d)).



(a) Example (b) Schedule of three nodes (c) Different bridges (d) Parallel cliques

Fig. 2. Example schedules and topologies of the clique mechanism

3) Algorithm: Assume that the conflict graph of WLANs is known by the central controller. The controller designs the target schedule S with cliques and bridges (each client belongs to at least one clique formed by its AP and itself). Since bridges always need to transmit, only APs are selected as bridges. If a bridge runs out of data packets, it transmits a CTS-to-Self with NAV$= 0$. The first position s_0 is always a bridge. S with information about the cliques and bridges is then delivered to all nodes.

The schedule matching of Rhythm-Clique is the same as that of Rhythm-Shrink. The position synchronization and the backoff calculation of Rhythm-Clique are similar to those of Rhythm-Shrink; thus, we illustrate only their differences. Initially, all nodes (except s_0) set a long backoff time. s_0 starts the first transmission, and nodes update Pos when they overhear transmissions. If Pos belongs to its active positions, a node operates as Rhythm-Shrink. Otherwise, it sets a long backoff time. When doing insertion, each clique works independently. That is, nodes maintain only a non-backlogged record for the cliques they belong to. In addition, an insert slot is added at the end of each clique portion.

4) Overhead estimation: Rhythm-Clique contains three major overheads: *i) Broadcasting extra information and dynamically generating conflict graphs:* Similar to broadcasting S, broadcasting additional information about cliques and bridges result in little overhead. For generating conflict graphs dynamically, we can utilize PIE [9], which is an online passive interference estimation for WLANs. PIE, which has very low overhead, fast convergence time, and fine-grained interference estimation, can handle mobile clients at walking speed (0.25 m/s). *ii) Transmitting CTS-to-self packets:* The overhead of transmitting CTS-to-self packets is $O_B = \frac{P_{AP} \times n_B \times CTS}{T_{sch}}$, where n_B is the number of bridges, P_{AP} the probability that a bridge (which is also an AP) runs out of packets, CTS the transmission time of a CTS packet, and T_{sch} the average period of the schedule cycle. Typically, because the downlink traffic is heavy [10], P_{AP} is small; since $k \geq n >> n_B$, $T_{sch} >> n_B \times CTS$. Thus, O_B is small. *iii) Backbone transmission delay between bridges:* The overhead introduced by delay in the backbone is $O_d = \frac{d \times n_{Bd}}{T_{sch}}$, where d is the average delay deviation between two bridges and n_{Bd} the number of bridges that communicate through the backbone. Consider $k = 30$ and $n_{Bd} = 2$, $O_d = 9.17\%$, even when $d = 0.5ms$. (The average delay d between two APs with one switch is 0.1ms with a standard deviation of 0.09ms [11].)

D. Other challenges and considerations

1) System architecture: The system architecture of WiFi networks operating Rhythm contains a central controller that periodically communicates with these networks. The central controller can be a WiFi controller or a server set up in the cloud. WiFi network information, such as the conflict graph and the client list, are periodically sent to the central controller. The conflict graph can be obtained by using PIE [9], and the client list can be monitored by each AP. Protocols such as CAPWAP [12] and LWAPP [13] allow APs to communicate with a central controller. Based on the collected information, the central controller determines the target schedule S with cliques and bridges, and delivers it to all nodes through APs.

2) Backward compatibility: Rhythm is backward compatible with legacy WiFi. The operation of Rhythm and non-Rhythm WiFi nodes can be separated into different time duration using network allocation vectors (NAVs) (which is similar to how PCF/DCF coexist). At the end of each schedule cycle, all bridges, one after another (following the order indicated in S), transmit a packet (e.g., a beacon) with an unused bit set (e.g., a power saving bit is not used in packets from APs [5]). This packet contains NAV=P, where P is the estimated transmission time of all Rhythm nodes. While non-Rhythm nodes set up NAV=P, Rhythm nodes recognize the unused bit and ignore the NAV. The estimation of P can be coordinated by bridges using backbone transmissions and adjusted to ensure fairness between Rhythm and non-Rhythm nodes.

3) Schedule updates and membership changes: The update of S is indicated by a sequence number carried in ACK sent by AP (ACK contains four unused bits [5]). If nodes receive or overhear a new sequence number, they switch to using non-Rhythm duration until they receive or overhear the new S from AP. When a node wishes to join a network, it first tries to overhear S from the APs, and then uses the non-Rhythm period for registration. Note that since Rhythm automatically shrinks the schedule, a delay in updating the target schedule does not decrease reduce channel utilization.

4) Scheduling decision: Currently, the central controller simply uses greedy algorithms to locate cliques in the conflict graph and places all nodes on the target schedule. However, since Rhythm actually controls relative scheduling and weighted fairness, with more intelligent scheduling decisions, it is possible for Rhythm to solve other MAC issues, such as QoS, energy-efficient, and higher layer traffic aware scheduling. In future work, we will explore the potential application of Rhythm in other MAC issues.

IV. Evaluation

In this section, we present experimental results carried out by WARP, and use ns2 simulations to evaluate each mechanism in Rhythm.[2] Then, we compare Rhythm to a closed related work, Domino [4]. Table II shows the simulation parameters, which follows 802.11g.

[2]Since the WiFi backoff timer duration is "lazily calculated" in ns-3 [14], and the backoff timer is vital for Rhythm, we evaluate Rhythm using ns-2.

TABLE I.
THROUGHPUT(MBPS)

Link Number	Rhythm	DCF
1	9.00	6.62
2	8.18	8.98
3	9.00	6.62
4	8.18	9.00
Total	34.36	31.22

TABLE II. NS2 PARAMETERS

Parameter	Value
Frame size	1500byte
Basic transmission rate	6Mbps
Data transmission rate	54Mbps
Slot time	$9\mu s$
SIFS	$10\mu s$
DIFS	$28\mu s$
δ	$26\mu s$

A. WARP experiments

We implement Rhythm in a software-defined radio platform: the Wireless Open-Access Research Platform (WARP) v3 [6]. We set up a fully connected topology in a typical indoor environment with three WARP nodes, one that acts as an AP, and the other two that act as clients, operating in 5.18GHz with 54Mbps data transmission rate. Iperf is used to generate UDP traffic with frame size 1500byte. Table I shows the throughput of each link (two uplinks and two downlinks) from the experimental evaluation. Rhythm produces a throughput which is 95% of the theoretical optimal (36Mbps) and better fairness than DCF (the target schedule S is: {AP, client1, AP, client2}). Given the limited number of hardware, the throughput difference between Rhythm and DCF is not large. In scenarios with more nodes in the following sections, large improvement will occur. We consider these experimental results as a proof of concept and evaluate the performance of Rhythm in more complicated situations using ns-2 simulations.

B. Baseline algorithm

Fig. 3 shows the time usage (channel utilization) of Rhythm in a topology with 20 nodes (all nodes are backlogged and the topology is fully connected). As expected, Rhythm spends almost no time in collision or backoff idle, and the small portion of idle time results from Interframe Space (IFS), which is an unavoidable protocol overhead, yielding channel utilization of almost 90%. Compared to DCF, Rhythm leads to 20% improvement in channel utilization. We also examine the ability of Rhythm to provide weighted fairness. Using the same topology, we randomly assign schedule weights to each node and calculate the weighted Jain's fairness index. Fig. 4 shows that Rhythm produce correct weighted fairness as indicated in the target schedule.

C. Shrinking mechanism

We examine the shrinking mechanism of Rhythm using fully-connected topologies and a variety of traffic. First, we present the performance of Rhythm-Base at determining the effect of the shrinking mechanism in a topology with 50 nodes. The two APs are always backlogged, and the clients are randomly backlogged periodically during a certain percentage of time. In Fig. 5, while the channel utilization of Rhythm-Base decreases as the average backlogged time decreases, Rhythm maintains channel utilization.

Then, to examine the overhead of mirror insertion, we carry out simulations with a large number of nodes. Figure 6 shows channel utilization with a different number of nodes. The two APs are always backlogged, and the clients have random traffic. Initially, the channel utilization of Rhythm slightly decreases since T_{col} slightly increases as the number of nodes increases. However, since T_{sch} and T_{inact} also increase as the number of nodes increases, this decrease slows down. Finally, Rhythm achieves over 200% improvement in channel utilization when the number of nodes reaches 300, indicating that Rhythm improves channel utilization in high-density deployed WiFi.

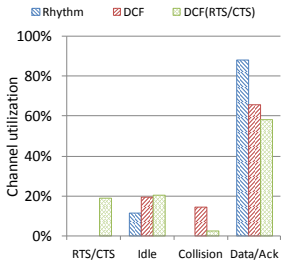


Fig. 3. Time usage of Rhythm

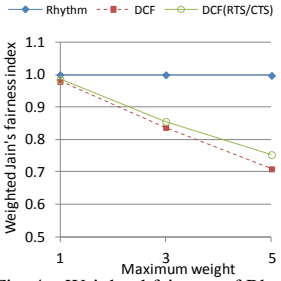


Fig. 4. Weighted fairness of Rhythm

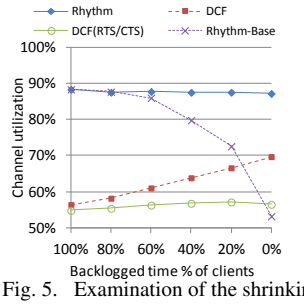


Fig. 5. Examination of the shrinking mechanism

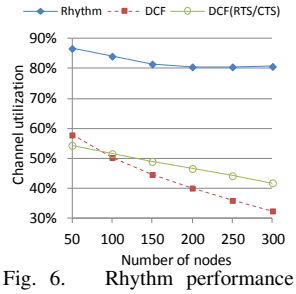


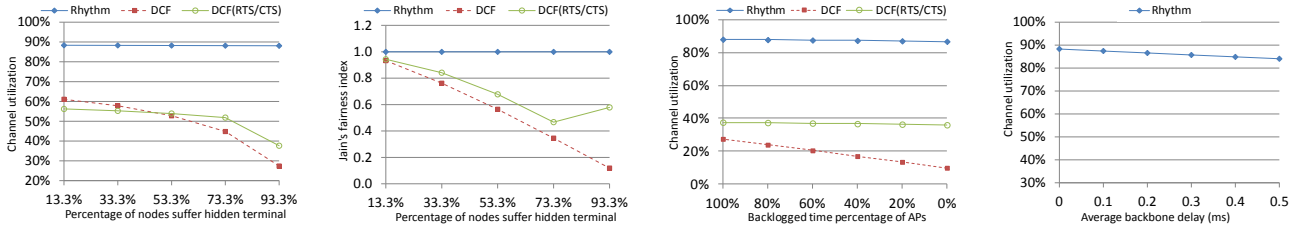
Fig. 6. Rhythm performance in dense-deployed WiFi

D. Clique mechanism

To examine the performance of the clique mechanism of Rhythm, we design various topologies and traffic loads. Since the overhead of the mechanism is not obvious when the number of nodes is large, we decrease the number of nodes to 30. Two APs are located in the center act as bridges; we change the locations of the clients to create hidden terminals. All nodes are continuously backlogged. Figs. 7(a) and 7(b) show channel utilization and fairness under varied percentages of nodes suffering from hidden terminal problems. With the clique mechanism, Rhythm yields near optimal channel utilization and fairness, as indicated in the target schedule. Next, to examine the overhead of transmitting CTS-to-self packets, we change the backlogged time percentage of the two APs. As shown in Fig. 7(c), since $n = 30 >> n_{Barr} = 2$, the overhead is small even when the backlogged percentage of APs is 0%. Finally, to estimate the overhead caused by backbone delays, we separate the two APs and ensure that they cannot hear each other and communicate through the backbone. As shown in Fig. 7(d), channel utilization decreases by 9% when the average backbone delay reaches 0.5ms, which agrees with our overhead estimation.

E. Comparison with Domino

We compare the performance of Rhythm with closely-related Domino [4]. Since the WiFi backoff timer duration is "lazily calculated" in ns-3 [14], we evaluate Rhythm using ns-2. However, as simulation of Domino is available in only ns-3, we implement Rhythm-Base (representing Rhythm) in ns-3 and compare it to Domino in a fully connected topology (no hidden terminal and no exposed terminal) with all nodes continuously backlogged.



(a) Channel utilization with hidden terminals (b) Fairness with hidden terminals (c) Channel utilization with non-backlogged bridges (d) Channel utilization with backbone transmission delay

Fig. 7. Examination of the clique mechanism

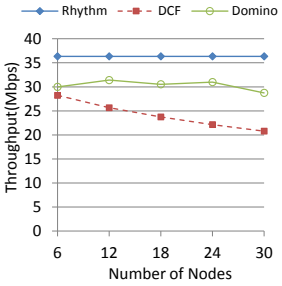


Fig. 8. Throughput in fully connected topologies

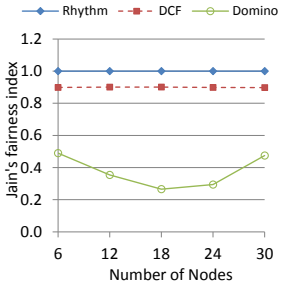


Fig. 9. Fairness when certain flows have a smaller queue size

Similar to Rhythm, Domino [4] triggers the next transmission by overheard transmissions. (The batch design in Domino is similar to the clique mechanism in Rhythm.) While the target schedule in Rhythm is a high-level transmission order in which nodes have a certain degree of distributed adjustment (e.g., the shrinking mechanism), the schedule in Domino is an exact transmission schedule that nodes always follow. Thus, Domino needs to collect the queue status from all nodes and generates a schedule based on the queue status. Since the queue status collection generates certain overheads, shown in Fig. 8, the throughput of Domino is lower, and its channel utilization is 76% while that of Rhythm can reach 88%. Nevertheless, since Domino generates a schedule based on the queue status, the schedule can be biased to flows with a larger queue size, especially when the batch size is large (i.e., it schedules all packets for flows with a smaller queue size and thus extra slots for flows with a larger queue size). Therefore, as indicated in Fig. 9, when certain flows have a smaller queue size, Domino does not ensure as much fairness as Rhythm.

V. Related work

Some papers have employed protocols similar to Rhythm. For example, CENTAUR [10] also utilizes the centralized WiFi architecture for enhanced performance of WiFi. It solves hidden terminal problems by separately scheduling conflicting downlink transmissions. However, it only gives control to downlink traffic, so it does not avoid hidden terminal problems generated in uplink traffic. It also improves performance, but only in cases with hidden/exposed terminal problems. Another protocol, Chain [15] broadcasts order coordination (similar to the target schedule) to achieve high channel utilization. However, it improves performance only when the traffic load is high. In addition, it neither addresses hidden terminal problems nor improves in downlink. Similar to Rhythm, Domino [4] utilizes relative scheduling to avoid overhead from tight time synchronization. While Domino provides an efficient way of collecting the queue status from all nodes, Rhythm provides a distributed adjustment mechanism without having to collect the queue status. However, as indicated in the evaluation section, Domino generates extra overhead and unfairness in certain scenarios. In addition, it requires use of a special address derived from Gold codes, which either limits the number of nodes (127 in its evaluation) in a collision domain or increases overhead by using larger signatures for addressing.

VI. Conclusion

This paper presented Rhythm, a MAC protocol under a theme of *enabling WiFi network controllability without compromising its scalability*. We showed that Rhythm provides near optimal channel utilization and weighted fairness under various traffic loads and connectivity scenarios. By providing controllability of WiFi networks, Rhythm can solve other MAC issues by scheduling, a topic that we will explore in future work.

References

[1] X. Yang and N. Vaidya, "A wireless MAC protocol using implicit pipelining," *IEEE Transactions on Mobile Computing*, 2006.

[2] M. Heusse, F. Rousseau, R. Guillier, and A. Duda, "Idle sense: An optimal access method for high throughput and fairness in rate diverse wireless LANs," *ACM SIGCOMM*, 2005.

[3] P. Djukie and P. Mohapatra, "Soft-TDMAC: A software TDMA-based MAC over commodity 802.11 hardware," *IEEE INFOCOM*, 2009.

[4] W. Zhou, D. Li, K. Srinivasan, and P. Sinha, "Domino: Relative scheduling in enterprise wireless LANs," *ACM CoNEXT*, 2013.

[5] "Part 11:Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications," *IEEE Std. 802.11-2012*, 2012.

[6] Wireless open-Access Research Platform (WARP). [Online]. Available: https://www.mangocomm.com/products/kits/warp-v3-kit

[7] S. Gollakota and D. Katabi, "ZigZag decoding: Combating hidden terminals in wireless networks," *ACM SIGCOMM*, 2008.

[8] C.-F. Shih *et al.*, "Rhythm: Achieving scheduled WiFi using purely distributed contention in WLANs," Tech. Rep., 2014. [Online]. Available: https://dl.dropboxusercontent.com/u/19975402/Rhythm.pdf

[9] V. Shrivastava, S. Rayanchu, S. Banerjee, and K. Papagiannaki, "PIE in the sky: Online passive interference estimation for enterprise WLANs," *NSDI*, 2011.

[10] V. Shrivastava *et al.*, "CENTAUR: Realizing the full potential of centralized WLANs through a hybrid data path," *ACM MobiCom*, 2009.

[11] T. Bansal, B. Chen, P. Sinha, and K. Srinivasan, "Symphony: Cooperative packet recovery over the wired backbone in enterprise WLANs," *ACM MobiCom*, 2013.

[12] P. Calhoun *et al.*, "Configuration and provisioning for wireless access points (CAPWAP) protocol specification," *IETF, RFC 5415*, 2009.

[13] ——, "Light weight access point protocol," *IETF, RFC 5412*, 2010.

[14] ns3: WiFi: The MAC model. [Online]. Available: https://www.nsnam.org/docs/models/html/wifi.html

[15] Z. Zeng, Y. Gao, K. Tan, and P. R. Kumar, "Chain: Introducing minimum controlled coordination into random access MAC," *IEEE INFOCOM*, 2011.