Look Who's Talking: A Practical Approach for Achieving Scheduled WiFi in a Single Collision Domain [*]

Chao-Fang Shih, Yubing Jian, and Raghupathy Sivakumar
Georgia Institute of Technology
{cshih,yubing,siva}@ece.gatech.edu

ABSTRACT

We ask the following question in this paper: *Can the goals of centralized WiFi scheduling be achieved using purely distributed operations?* We present a solution called *Look Who's Talking (LWT)* that allows for arbitrary schedules to be distributed to nodes in a WiFi network. The nodes in the network then use purely local and distributed operations to achieve the prescribed schedule. The scope of LWT in this paper is restricted to a single collision domain (single or multiple cells), but we discuss how LWT can be extended to multiple collision domains. We use both experimental evaluations (using a WARP-based testbed) and simulation-based analysis (using ns3) to evaluate LWT.

CCS Concepts

•Networks → Network protocols; Wireless local area networks;

Keywords

centralized WiFi, scheduled WiFi, media access control

1. INTRODUCTION

Most WiFi deployments today use the distributed coordination function (DCF) mode of the IEEE 802.11 standard [1]. The DCF mode of operation is simple and scalable, and requires a participating node to listen to the channel and make contention decisions purely on locally available information. The contention algorithm in turn is controlled by a set of parameters including the maximum contention window that are adaptively adjusted based on local information. While the approach is simple and scalable, the goal of DCF is to achieve coarse-level fairness and efficiency in the network. Any finer-level goals are beyond the scope of DCF.

The IEEE 802.11 point coordination function (PCF) mode on the other hand relies on centralized scheduling by the access-point (AP) [1]. Theoretically, the scheduling algorithm at the AP can be arbitrarily defined. The problems with PCF are two-fold: i) it uses a polling process that incurs heavy overheads, especially in dynamic load conditions, and ii) the standard does not specify how APs should coordinate with each other to prevent collisions across cells.

There has been interest lately on the problem of achieving the benefits of centralized scheduling while retaining the simplicity and scalability benefits of distributed operations[1] [3]. The benefits of centralized scheduling are the following:

- Predictability: Applications and services that require predictable service can expect to receive it in a setting with centralized scheduling. The central scheduler has complete control over what is transpiring in the network, and hence provide assurances.
- Differentiation: Applications and services can be provided with different resource allocations depending on their requirements. While there are distributed approaches to accomplish this goal (e.g., IEEE 802.11e [1]), they are quite coarse in the differentiation they provide.
- Efficiency: Finally, in environments where operational efficiency is an issue (e.g., in high-density WiFi deployments), centralized scheduling can lead to higher efficiencies.

On the other hand, the advantages of purely distributed operations are the lack of a single point of failure or bottleneck, scalability with the number of nodes, and backward compatibility with how WiFi is predominantly used today [4].

The context for this paper is this bridge between centralized scheduling and purely distributed operations. We ask

[*]This work was funded in part by the National Science Foundation under grants IIP-1343435 and CNS-1319455, and the Wayne J. Holman Endowed Chair.

CoNEXT '15 December 01-04, 2015, Heidelberg, Germany
© 2015 ACM. ISBN 978-1-4503-3412-9/15/12…$15.00
DOI: http://dx.doi.org/10.1145/2716281.2836116

[1]In a related domain, software-defined networks (SDNs) aim to accomplish a similar goal (e.g., OpenFlow [2]).

the following question:*Can the goals of centralized scheduling be achieved using purely distributed operations?* We present a solution called *Look Who's Talking* (LWT) that allows for arbitrary schedules to be distributed to nodes in a WiFi network. The nodes in the network then use purely local and distributed operations to achieve the prescribed schedule. The scope of this paper is restricted to a single collision domain (single or multiple cells), but we briefly discuss how LWT can be extended to multiple collision domains.

The core research contributions are however in how LWT tackles (i) work conservation under dynamic load conditions; (ii) schedule tracking when transmissions are outside decodable (but detectable) ranges; and (iii) expansion of local information when hidden terminals exist. We use both experimental evaluations and simulations-based analysis to study the performance of LWT, and show that not only are the mechanisms in LWT quite practical and achievable, but the performance of LWT in terms of adherence, efficiency, and protocol overhead is attractive.

The rest of the paper is organized as follows: Section 2 provides some background information on WiFi and presents the problem definition. Section 3 outlines the LWT solution. Section 4 discusses the performance of LWT, and Section 5 concludes the paper.

2. BACKGROUND AND PROBLEM DEFINITION

2.1 WiFi DCF - A Primer

The Distributed Coordination Function (DCF) mode of IEEE 802.11 ([1]) is a Carrier Sense Multiple Access (CSMA) MAC protocol. It belongs to the *listen before talk* family of protocols. Before a transmitter node (Tx) transmits, it senses the channel to determine if there are other nodes transmitting. If the channel is busy, the Tx [2] defers until the channel becomes idle. If the channel is idle for a specified duration (the DCF interframe space (DIFS)) the Tx infers the channel to be idle and randomly selects a backoff number in [0, cw], where cw is the the contention window. Then, the Tx counts down the backoff number in terms of backoff slots. If the channel becomes busy before the backoff timer expires, the Tx freezes its backoff and defers until the channel becomes idle again. Otherwise, the Tx transmits when the backoff number becomes zero. If the DATA transmission is successfully received, the receiver node (Rx) sends an ACK after a short interframe space (SIFS) duration. Fig. 1 shows the timeline for a DCF transmission. An optional mechanism, exchanging short control frames (RTS and CTS frames) before the data transmission, can be used to decrease the probability and impact of collisions, but is rarely used due to its associated overheads.

2.2 Scheduled WiFi

[2]In this paper, we use Tx to refer transmitter node and Rx to refer receiver node.

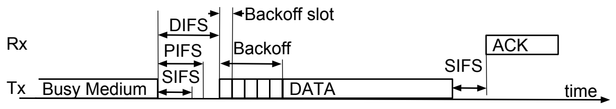


Figure 1: Time line of a DCF transmission

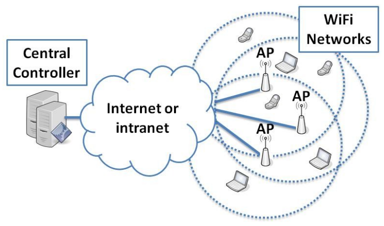


Figure 2: System architecture for scheduled WiFi

Scheduled WiFi is the notion of making WiFi nodes transmit according to an order specified in a target schedule S. Fig. 2 shows a possible system architecture for scheduled WiFi consisting of a central controller and a multi-cell WiFi network. The central controller (which can either be an onsite controller in an enterprise WiFi deployment or a cloud controller for APs that do not belong to the same administrative domain) communicates with all APs in the WiFi network. Protocols such as CAPWAP([5]) and LWAPP([6]) facilitate the communication between the APs and the central controller. WiFi-related network information is sent to the central controller periodically. Based on the collected information and other configured policies, the central controller determines a target schedule $S = \{s_0, \cdots, s_{k-1}\}$, where $s_i = (tx_i, rx_i)$ indicates the scheduled link in schedule position i (tx_i and rx_i are the MAC addresses of the Tx and Rx of the link). The schedule is then pushed to the nodes in the network through the APs.

For the work in this paper, we define a metric called *adherence*, adh, to measure how well the WiFi nodes track the prescribed schedule. The transmission pattern of WiFi nodes is $T = \{t_0, \cdots, t_{m-1}\}$, where t_i can be a successful transmission ((tx_i, rx_i)) or a transmission without ACK (Col). We partition T into several regions $\{T'_0, \cdots, T'_l\}$ using $t_i = Col$ (Fig. 3) as a separator. The adherence is $adh = \frac{1}{m}\Sigma_{i=0}^{l} \max\{HCC(T'_i, S)\}$, where HCC is the hamming cross-correlation function. Note that $adh \geq 0$, and $adh = 1$ means perfect adherence.

The ability to make nodes transmit while following a prescribed schedule has many benefits as outlined in Section 1. Specifically, the benefits center around predictability of service, efficiency under heavy load conditions, and weighted differentiation when applications and services require different resource allocations. More generically, once nodes in a network can be made to follow a schedule, *any MAC problem (e.g. energy-efficient scheduling, transport-layer aware scheduling, usage-based scheduling, etc.) can now be solved in a much easier fashion, since only a centralized solution to the problem needs to be constructed.* The output of the centralized scheduler is simply furnished to the network nodes that then achieve that schedule.

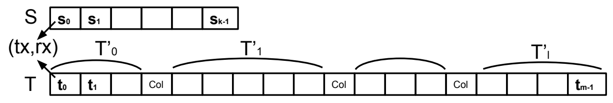


Figure 3: Adherence of a transmission pattern

2.3 Related Works

It is possible to achieve scheduled WiFi using a purely centralized MAC protocol such as the point coordination function (PCF) [1] or Soft-TDMAC [3]. However, these protocols require queue status of nodes or use polling for scheduling. They either assume that nodes always transmit or incur large overheads in collecting queue status from all the nodes. Also, the protocols require tight time synchronization for good efficiency, adding to the burden on the network operations. On the other hand, the requirement of time synchronization for LWT is the same as that of the 802.11 standard. Thus, general WiFi nodes can support LWT's required time synchronization level.

Hybrid MAC protocols that try to schedule WiFi transmissions have also been proposed more recently: CENTAUR [7], Chain [8], and DOMINO [9]. For example, CENTAUR schedules certain conflict links to avoid collisions generated by hidden terminals. However, it only controls downlink traffic. Chain [8], on the other hand, only schedules uplink traffic.

DOMINO [9] is an approach that has some properties similar to that of LWT. Both approaches utilize relative scheduling to workaround the tight time synchronization requirement. In order to efficiently collect queue information from all nodes, DOMINO uses the sub-carriers of WiFi. LWT also relies on subcarriers for its flash signals [10]. DOMINO uses Gold codes [11] while LWT utilizes flash signals. However, there are three main differences between LWT and DOMINO in the context of a single collision domain: (i) *DOMINO exactly follows the schedule specified by the central entity.* DOMINO collects queue information from all the nodes and updates the schedule frequently. It is thus impacted by the delay between the central entity and APs. The collection of queue information also incurs non-negligible overhead. On the other hand, LWT doesn't need any queue information and can dynamically adjust the slot usage without help from the central entity. (ii) *The schedule decision of DOMINO is restricted by the network topology.* This increases the schedule design complexity. On the other hand, LWT can support any schedule decision. LWT also saves on the overheads involved in gathering network-wide topology information. (iii) *DOMINO cannot handle hidden terminal problems under certain scenarios* while LWT can handle all scenarios in a single collision domain. We illustrate this difference further in Section 4.

2.4 Scope and Challenges

The focus of this paper is on WiFi networks in a single collision domain, in which any two simultaneous transmissions cause a collision. WiFi network deployments typically have auto-channel-selection mechanisms (3 to 12 orthogonal channels depending upon the spectrum used). For a given channel, most networks are practically either in a single collision domain or are totally disconnected, and hence can operate independently. We discuss how to extend LWT for multiple collision domains in Section 3.5.4 briefly, but leave its in-depth exploration for future work.

Thus, the problem addressed by this paper can be stated as follows. Consider a multi-cell WiFi network containing n nodes in a single collision domain. The central controller decides a target schedule S, which is delivered to all nodes. Having S, how can the nodes achieve scheduled WiFi efficiently? We present below a list of non-trivial challenges that need to be addressed by any scheduled WiFi solution:

Non-backlogged Nodes: A common problem in scheduled WiFi is to deal with non-backlogged nodes. When nodes do not always have packets to transmit, it is hard to determine whether to schedule the node. While collecting queue status from all the nodes is one solution to the problem, this collection incurs non-negligible overheads and delays, especially when there are a large number of APs or large delays between the APs and the central controller. Thus, it is desirable to deal with non-backlogged nodes without collecting queue status.

Decodability vs. Detectability: The WiFi PHY layer uses multiple rates for data transmissions. Thus, overheard packets cannot always be decoded correctly. This prevents nodes from fully relying on information from overheard transmissions. To deliver control-plane information, one possible solution is to use extra signals or control frames, which in turn increases overheads. Hence, it is desirable to construct a solution to tackle such non-decodable scenarios while incurring minimal overheads.

Partial Connectivity: Due to any partial connectivity caused by network topology or obstacles, hidden terminals exist even in single collision domains. A well-known solution for the hidden terminal problem in WiFi is the exchange of RTS/CTS before data transmissions. However, RTS/CTS introduce considerable overheads. Also, the mechanism cannot solve unfairness problems under certain scenarios (we illustrate this in Section 3.4). Thus, it is desirable to achieve scheduled WiFi even in the presence of hidden terminals.

Backward Compatibility: Since it is unrealistic that all devices in a target deployment can be updated, backward compatibility is an important property of any newly designed MAC protocols. Some MAC protocols (e.g., DOMINO [9]) deal with legacy nodes by separating their transmissions into different time period. This introduces delay for legacy nodes and requires extra control overheads. It is thus desirable to construct a solution that allow legacy nodes to operate normally without additional overheads and delay.

Sleeping Nodes: WiFi radios can be put to sleep to conserve energy. Nodes whose radios were put to sleep need to be able to rejoin the network and sync with the schedule dynamically. A naive solution is to use a fixed duration for each transmission (e.g., DOMINO [9]), so that nodes can use the time elapsed to estimate the current schedule slot after sleeping. However, WiFi supports multiple rates for data transmissions, and packet sizes can vary for different

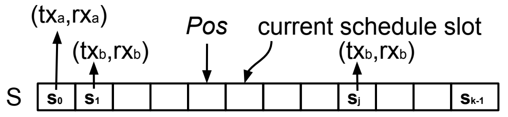


Figure 4: Position synchronization

applications. The fixed transmission times require packet aggregation and fragmentation, which in turn introduces extra delays. Hence, it is desirable to address sleeping while allowing for different transmission durations.

Schedule Changes: It is also important to allow for changes in the target schedule. Unicasting the schedule to each node on any update is reliable but introduces large overheads. Broadcasting the schedule is efficient but can cause problems if the schedule is not delivered reliably to some nodes. Thus, it is desirable to design a solution that allows for updates to the target schedule.

In the following section, we start with an idealized scenario that does not have several of the above challenges, and then progress systematically in presenting a solution that addresses all the challenges.

3. LWT: SCHEDULED-WIFI USING DISTRIBUTED CONTENTION

3.1 Baseline algorithm

Consider a simple scenario with a fully connected topology, in which overheard packets can be successfully decoded and nodes are always backlogged. Under this scenario, we introduce the baseline algorithm of LWT, LWT-Baseline.

3.1.1 *From Centralized Scheduling to Distributed Contention*

Scheduled WiFi is achieved by centralized scheduling protocols through the following steps in each node: i) access to the target schedule , ii) get the current schedule slot, iii) trigger a specific transmission, iv) know when the current transmission ends, v) adjust to the next schedule slot, and get back to step iii). Similarly, LWT achieves scheduled WiFi through the following steps in each node: i) receive the broadcast of the target schedule, ii) use “position synchronization” to get the current schedule slot, iii) reuse backoff mechanism for self-triggering, iv) track transmission completions, v) move the “position pointer” to the next schedule slot, and get back to step iii). Below, we describe the detail of each step in LWT.

i) Broadcast of Schedule

In LWT, APs broadcast the target schedule S to all nodes. S can be put into a beacon every few seconds. Broadcasting generates negligible overheads , which is: $\frac{k\times 48\times 2}{R_b\times T_p}$ (the time percentage used to transmit S), where 48 is the length of a MAC address, k is the length of S, R_b is the sending rate of beacons, and T_p is the period of updating S. Consider $R_b =$ 6Mbps, $k = 100$, and $T_p = 1$s, the overhead of broadcasting S is only 0.2%. However, broadcasting is unreliable and

Algorithm 1 LWT-Baseline

```
1: function PosSYNC
2:     if ($r_0 == Col$) or ($r_0 == Null$) then
3:         ST=RAND, Pos=Null
4:         return
5:     else if ST==RAND then
6:         Pos=PosMatch(-1)
7:     else
8:         Pos=(Pos+1)$\bmod k$
9:     end if
10:    if Pos! =Null then
11:        ST=SYNC
12:    end if
13: end function
14: function PosMatch(prevPos)
15:    $i$ = (prevPos+1) mod $k$
16:    for $j = 1$ to $k$ do
17:        if $s_i == r_0$ then
18:            return $i$
19:        end if
20:        $i = (i + 1) \bmod k$
21:    end for
22:    return $Null$
23: end function
```

can cause problem if the schedule is not reliably delivered to some nodes. We will deal with this issue in Section 3.5.3 when considering the problem of change to schedule.

ii) Position Synchronization

In order to follow a schedule, nodes need to know the current schedule slot. In LWT, this is achieved by synchronization in a schedule position pointer *Pos*. Each node in the network maintains a schedule pointer *Pos* that points to a position (0 to $k-1$) in the schedule; the next position where *Pos* points to is the current schedule slot (Fig. 4).

Pos synchronization is achieved by transmission overhearing. Initially, nodes perform random backoff as in DCF. Once a node wins the contention and finishes a transmission successfully, all nodes learn the MAC addresses of the Rx and Tx of this transmission through overhearing. Then, all nodes find *the smallest position of this transmission* in S, and set *Pos* to that position. For example, in Fig. 4, the smallest position of link (tx_b,rx_b) is 1. Matching a link to the smallest position avoids ambiguity when a link is scheduled multiple times [3]. *Pos* synchronization gives all nodes a common current schedule slot. After the first synchronization, it is sufficient to maintain *Pos* synchronization by incrementing the position of *Pos* after each transmission. Note that since DCF requires nodes to always listen to the channel for the backoff mechanism, nodes can track the start and end time of schedule slot. Also, the active listening time of LWT is the same as that of DCF.

Algorithm 1 illustrates the *Pos* synchronization of LWT-Baseline. Assume that a target schedule $S = \{s_0, \cdots, s_{k-1}\}$ is known to all nodes in the network. Nodes record the most recent transmission r_0 (Can be its own transmission or an overheard transmission). $r_0 \in \{s_i\} \bigcup Col$, where Col represents a transmission without ACK. Nodes update *Pos* according to r_0 and a synchronization state *ST*. There are two

[3]We will talk about how to handle sleep nodes or new nodes in Section 3.5.

states of *ST*: *RAND* and *SYNC*. Initially, *ST* is *RAND*. If r_0 is $Null$ (never heard any transmissions) or Col, nodes set *ST* to *RAND* and *Pos* to $Null$. If $r_0 \in \{s_i\}$ and *ST* is *RAND*, which means there is no *Pos* synchronization before this transmission, nodes set *Pos* to the smallest position matching r_0 in S and set *ST* to *SYNC*. If $r_0 \in \{s_i\}$ and *ST* is *SYNC*, nodes simply increment the position of *Pos* by 1 (line 8).

iii) Self-Triggering

The backoff mechanism is used to trigger the transmission of the current scheduled link. Note that, as mentioned in Section 2.1, since nodes need to freeze the backoff timer once the channel becomes busy, nodes constantly listen to the channel when counting down the backoff number. Thus, once a new transmission is overheard, a node freezes the backoff timer, updates *Pos*, and determines a new backoff number according to the state *ST*. If *ST* is *SYNC*, it uses *Pos* to find the current schedule slot. If the node is scheduled in the current schedule slot, it sets the backoff number to 0. Otherwise, it sets a large backoff number and waits for *Pos* update. If *ST* is *RAND*, the node carries out random backoff as DCF.

iv) Transmission Completions

Nodes need to learn the completion of the current transmission to start the next transmission in schedule. In DCF, nodes always track the completion of each transmission in order to start the backoff mechanism. LWT utilizes the same mechanism to track transmission completions.

v) The Next Schedule Slot

In LWT, nodes go to the next schedule slot simply by incrementing the position of Pos by 1.

3.1.2 Efficiency Estimation

We give a brief efficiency estimation of LWT-Baseline by estimating the overhead caused by collisions and packet errors. Assume the success of transmissions when n nodes contend using DCF are independent Bernoulli trials, and P_{col} is the probability of having a collision (failure) for each trial. In LWT-Baseline, one successful transmission achieves *Pos* synchronization and avoids collisions. Thus, the average overhead caused by collision in m transmissions is $O_{col} = \frac{(P_{col}+\cdots+(m-1)P_{col}^{(m-1)})}{m}(1-P_{col})+P_{col}^{(m)} = \frac{\frac{1}{1-P_{col}}-1}{m}$. When $m \to \infty$, $O_{col} \to 0$. That is, the overhead is 0 in long term when there is no packet error. Assume P_{err} is the packet error rate. The overhead caused by packet errors and collisions in DCF is $O_{err\_col} = (1-P_{col})P_{err} + P_{col}$; this overhead in LWT-Baseline is $P_{err} \times (1-P_{col})(1 + P_{col} \times 2 + P_{col}^2 \times 3 + \cdots) \leq P_{err} \times \frac{1}{1-P_{col}}$. Considering $P_{err} = 1\%$ and $P_{col} = 15\%$, $O_{err\_col} = 15.85\%$ in DCF, and $O_{err\_col} \leq 1.18\%$ in LWT-Baseline.

3.1.3 Limitations of Baseline algorithm

Even though LWT-Baseline achieves scheduled WiFi efficiently, it is limited to simple scenarios where nodes are always backlogged and transmissions can always be overheard and decoded. As mentioned in Section 2, we will consider other practical issues sequentially in the following subsections.

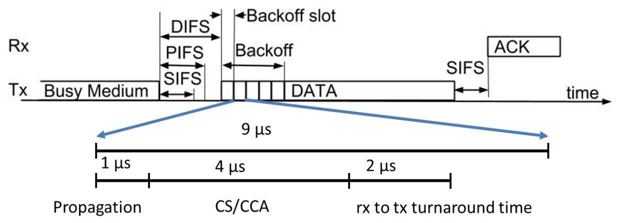


Figure 5: CS/CCA in a backoff slot

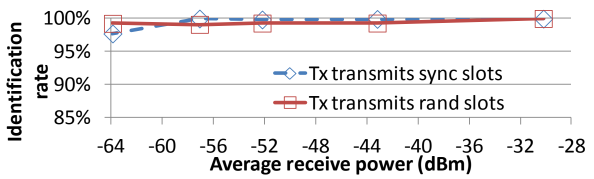


Figure 6: Identification rate of different slots

3.2 Work conservation with non-backlogged nodes

In this section, we consider scenarios with non-backlogged nodes. As mentioned in Section 2, it is hard to determine the schedule when nodes do not always transmit. However, collecting queue status from all nodes brings overheads and delays. We introduce a *work-conserving* mechanism in LWT (LWT-WC), which utilizes the carrier sense/clear channel assessment (CS/CCA) mechanism of WiFi to automatically release unused resources to other nodes when the scheduled node doesn't transmit.

3.2.1 CS/CCA of WiFi

The core insight of LWT-WC lies in the CS/CCA mechanism of WiFi. WiFi packets contain a PLCP preamble in the beginning for synchronization. It is also used for CS/CCA to identify the start of a transmission through auto-correlation or cross-correlation. The IEEE 802.11 standard specifies that an OFDM transmission at a receive level no less than the minimum sensitivity (-82dBm for 20MHz channel) shall cause CS/CCA to indicate channel busy with a probability over 90% within 4 μs (for 20MHz channel)[1]. This timing is sufficient for nodes to identify the start of a transmission within a backoff slot (9 μs). Fig. 5 gives a detailed illustration. A node starts to transmit at the beginning of a backoff slot if its backoff timer expires. The propagation delay is around 1 μs, and nodes take 4 μs to identify the start of a transmission. If a transmission is detected, nodes hold the backoff timer. If no transmission is detected, since the receive to transmit turnaround time of a node is less than 2 μs [1], it is sufficient for a node to prepare to transmit in the beginning of the next backoff slot if its backoff timer expires.

3.2.2 Work-Conserving

Algorithm 2 LWT-WC

```
function PosSYNC
    if ($r_0 == Col$ and $w_t < w_{t\_thd}$) or ($r_0 == Null$) then
        % rand slot does not trigger reset of ST
        ST=RAND, Pos=Null
        return
    else if ST==RAND then
        Pos=PosMatch(-1)
    else
        Pos=$(Pos+1) \bmod k$
    end if
    if Pos$! = Null$ then
        ST=SYNC
    end if
end function
function SelfTrigger
    if ST==SYNC and myself==current scheduled node then
        set backoff to zero
    else if ST==SYNC then
        do random backoff as DCF with $b \geq 1$
    else
        do random backoff as DCF
    end if
end function
```

Utilizing the ability of identifying the start of a transmission within a backoff slot (which is an ability required by IEEE 802.11 standard [1]), LWT-WC classify schedule slots into different types according to the timing of the start of the transmission.

Classifying Schedule Slots

LWT-WC classifies slots into two types: "sync slot" and "rand slot." If the current scheduled node transmits, it is identified as a sync slot. Otherwise, this slot becomes a rand slot, and all nodes can contend for this slot using DCF. The identification of sync slot and rand slot is automatically carried out with CS/CCA and backoff mechanism. In LWT-WC, when nodes are not scheduled in the current schedule slot, they pick a random backoff number ≥ 1. Since the current scheduled node sets backoff number to zero, nodes identify the start of a transmission within the first backoff slot if the current scheduled node transmits. Otherwise, if there is no transmission starting within the first backoff slot, nodes continue to backoff and this slot becomes a rand slot.

3.2.3 Experimental Validation of Work-Conserving

We set up WiFi communications using WARP [12] to validate the core insight of LWT-WC. In our experiment, one node acts as Tx and transmits unicast packets to the other node, which acts as Rx and tries to figure out which slot the Tx uses. We evaluate slot identification rate when the Tx always transmits in sync slots and when the Tx always transmits in rand slots. When transmitting in rand slots, the Tx sets backoff number to one, which creates the smallest time difference between rand slots and sync slots. Fig. 6 shows the correct identification rate of both scenarios under different receive power. The experiment result shows that it is reliable (over 98% correctness) to use CS/CCA to identify a sync slot and a rand slot. In Fig. 6, the identification rate when Tx transmits in sync slot goes down to 98% when the receive power is -64dBm. It is because that the probability of CS/CCA correctly indicating channel busy becomes lower in low receive power. If CS/CCA fails to identify the transmission in the first backoff slot, the sync slot is wrongly identified as a rand slot.

3.2.4 Algorithm

Algorithm 2 illustrates the Pos synchronization and self-triggering mechanism of LWT-WC. Nodes record a parameter w_t, which is the waiting time between the current transmission and the previous transmission, and a parameter w_{t_thd}. If the previous transmission is Col, w_{t_thd} is set to (EIFS $+SlotTime/2$), where EIFS is the interframe space after an erroneous transmission [1] and $SlotTime$ is the time duration of a backoff slot; otherwise, w_{t_thd} is set to (DIFS $+SlotTime/2$). Transmissions in rand slots won't trigger reset of ST even if it is a collision Col. That is, if there is a collision in a rand slot, ST will remain the same value, and if *ST* is *SYNC*, nodes simply increase Pos by one (line 9). When nodes are not scheduled in the current schedule slot, they pick a random backoff number $b \geq 1$.

3.2.5 Efficiency Estimation

Implementing work-conserving mechanism can increase the overhead of LWT since the rand slots can have larger backoff slots and collisions. As estimated in Section 3.1.2, the overhead caused by packet errors and collisions in DCF is $O_{err\_col} = (1 - P_{col})P_{err} + P_{col}$; this overhead in LWT-Baseline is $\leq P_{err} \times \frac{1}{1-P_{col}}$. The overhead of LWT-WC is the linear combination of the overhead of DCF and LWT-Baseline, weighted by the time portion of the rand slots and sync slots. When the traffic load is high, the channel efficiency of LWT-WC approaches that of LWT-Baseline, which is near-optimal. When the traffic load is low, the channel efficiency of LWT-WC approaches that of DCF, which is efficient enough in low traffic load.

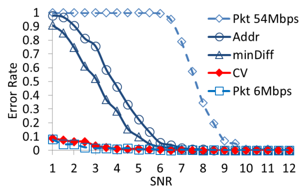
3.3 Decodability vs. Detectability

In this section, we consider scenarios in which overheard transmissions cannot be decoded. As mentioned in Sec. 2, nodes sometimes can not get information from overheard transmissions. However, using extra signals/control frames introduces overheads. Although nodes in LWT only need the Tx and Rx addresses of the first successful transmission for Pos synchronization, re-synchronization can be triggered by change of schedule, sleeping nodes, new nodes, transmission errors, and collisions. Identifying the addresses of transmissions in sync slots is important under those re-synchronization situations. Thus, we introduce a conditional Viterbi algorithm in LWT (LWT-CV) to help nodes identify addresses of a transmission when it cannot be decoded.

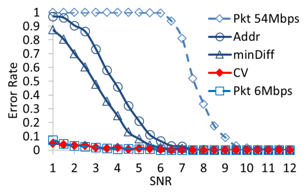
3.3.1 Theoretical Error Rate of Identifying Addresses

The core insight of conditional Viterbi algorithm is: *instead of consider all decode paths, consider only a subset of decode paths when decoding*[4]. Since the target schedule S contains addresses of all nodes in the network, we

[4]Please find the definition of decode path in Chapter 10.5 of [13]



(a) $2m$=50 possible addresses



(b) $2m$=20 possible addresses

Figure 7: Address identification error rate

can consider only the n addresses in S rather than all MAC addresses, which reduces the number of decode paths from 2.8×10^{14} to n (the number of nodes), which practically is less than 50 in a single collision domain. The large reduction of number of decode paths greatly increases the distance between paths and also restricts the possibility of errors to only certain bits, thus the error rate decreases significantly.

3.3.2 *Conditional Viterbi Algorithm*

Convolutional Code

WiFi uses convolutional code for channel coding. Convolutional code [13] is an error-correcting code . Small bit errors can be recovered by observing the whole sequence. The encoder can be viewed as a finite-state machine.

Decoding with an Address Tree

Viterbi algorithm is an optimal algorithm for decoding convolutional codes. It utilizes dynamic programming to find the decode path with minimum distance. Conditional Viterbi algorithm is based on Viterbi algorithm, but only considers certain decode paths. An address tree is built according to addresses in S, and only the decode paths in the address tree are considered.

3.3.3 *Algorithm*

Algorithm 3 illustrates the conditional Viterbi Algorithm, where $tree$ is the address tree built according to addresses in S. Each time the received/overheard transmission can not be decoded, conditional Viterbi algorithm is used to identify the Rx and Tx addresses. A parameter, $tree\_state$, is used to trace the current location in the address tree. $tree\_state$ is set to the root of address tree at the first bit of each address (line 12). Then, only the input that has a path in the tree will be considered when decoding (line 15).

3.3.4 *Evaluation of Conditional Viterbi Algorithm*

We use Matlab simulation to evaluate conditional Viterbi Algorithm. In the simulation, m MAC addresses are randomly generated. For each of the m addresses, another address that has different last 4 bits to it is generated. (Note that for each address, there is another address having the same first 44 bits.) From the $2m$ addresses, two addresses are randomly selected. Transmissions containing the two addresses and random payload are received though an AWGN channel with different SNR. Fig. 7(a) and 7(b) shows the address identification error rate of conditional Viterbi algorithm when the transmission rate is 54Mbps (64-QAM with 3/4 coding rate) with $2m = 50$ and $2m = 20$. Only the least robust rate is shown to keep the graph clear. The identification error rate of other rates are smaller than that of 54Mbps. The packet error rate of 6Mbps (Pkt 6Mbps) and 54Mbps (Pkt 54Mbps) are also shown as references. The error rate of choosing the addresses in S that has minimum distance to the decoded addresses of Viterbi algorithm, which is the "minDiff" in figures, is also presented. As indicated in figures, conditional Viterbi algorithm greatly decreases the identification error rate of addresses.

Algorithm 3 Conditional Viterbi Algorithm

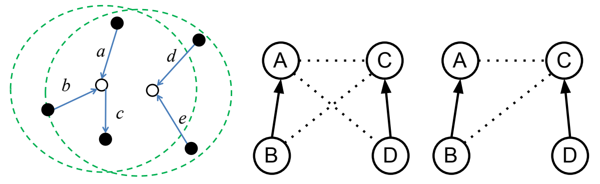
```
1: function CONDVIT(recv_bits, trellis, end_of_addr, init_of_addr, tree)
2:     NumStates= trellis.numStates
3:     n_addr_bit = 48
4:     metrics = zeros(NumStates, end_of_addr+1)+inf
5:     path = zeros(NumStates, end_of_addr+1)-1
6:     tree_state = zeros(NumStates, end_of_addr+1)-1
7:     metrics(1,1)=0
8:     for i = 1 to end_of_addr do
9:         for state = 1 to NumStates do
10:            if metrics(state,i)!=inf then
11:                if (i==init_of_addr)or(i== init_of_addr + n_addr_bit) then
12:                    tree_state(state,i)=root
13:                end if
14:                for input = 0 to 1 do
15:                    if (i<init_of_addr)or(tree(tree_state(state,i),input)!=NULL) then
16:                        next_state=trellis.nextStates(state,input)
17:                        output= trellis.outputsstate,input
18:                        mt=sum_diff(output,recv_bits(i))+metrics(state,i)
19:                        if mt < metrics(next_state,i+1) then
20:                            metrics(next_state,i+1)= mt
21:                            path(next_state,i+1)= state
22:                            if i ≥ init_bit_addr then
23:                                next_tree_state=tree(tree_state(state,i),input)
24:                                tree_state(next_state,i+1)=next_tree_state
25:                            end if
26:                        end if
27:                    end if
28:                end for
29:            end if
30:        end for
31:    end for
32:    % trace back the path with minimum metrics
33:    decode = TraceBackPath(path, min(metrics), trellis)
34:    decoded_addr=decode(init_of_addr:end_of_addr)
35: end function
```

3.4 Partial Connectivity

In this section, we consider scenarios in which some transmissions cannot be overheard, causing hidden terminals problems. RTS/CTS is a famous solution for hidden terminal problems. However, it introduces considerable overheads and has unfairness problems in certain scenarios. Thus, we propose a transparent transmissions mechanism in LWT (LWT-TT) to deal with hidden terminal problems in a single collision domain.

3.4.1 *Hidden Terminal Problem in a Single Collision Domain*

We define a single collision domain as "a set of links in



(a) Single Collision Domain (b) HtRx (c) HtNRx

Figure 8: Examples of hidden terminals in a single collision domain

which transmissions through any two of them cause a collision." Fig. 8(a) shows a single collision domain. Hidden terminal problems happen when Tx of links cannot be overheard. We classify hidden terminal problems in a single collision domain into 2 types: i) HtRx: at least one node of each link can be overheard by all nodes in the single collision domain. ii)HtNRx: there exists a link where both Rx and Tx cannot be overheard by some nodes in the single collision domain. Fig. 8(b) and 8(c) show examples of both scenarios.

A well-known solution for hidden terminal is exchanging RTS/CTS before a data transmission. However, RTS/CTS introduces extra 116 μs to each transmission. Consider a packet of 1500bytes transmitting in 54Mbps, the overhead caused by RTS/CTS is 26%. Also, RTS/CTS can not solve unfairness problems in HtNRx scenarios. In Fig. 8(c), it is very hard for node D to win a contention. Since the transmission of node B interferes the reception of node C, but the transmission of node D does not interfere reception of node A or B, RTS transmitted from node D to node C can be corrupted by node A or B easily. Also, since node D can not hear the RTS/CTS of node A and B, it can keep retransmitting RTS while any of the 2 nodes are transmitting. This makes node D use unnecessary large backoff numbers.

In order for LWT to gain *Pos* synchronization, the start and end time of a transmission need to be learned by all nodes in the single collision domain. We first declare that "if every node that hears a transmission, broadcasts the information of this transmission, all nodes in a single collision domain can get the information of this transmission." The proof of this statement is simple. If there is a node that can not get the information of a transmission, this means that all of its Rx cannot hear this transmission, thus this node actually can transmit without interfering with the on-going transmission, which contradicts the definition of a single collision domain. Below, we use this statement and introduce the transparent transmissions mechanism.

3.4.2 Transparent Transmissions

In order to pass the transmission information to all nodes in the single collision domain, we propose a mechanism named *transparent transmissions*. When a node receives or overhear a transmission, it transmits a special signal, tt_Start, to indicate the start of a transmission. After the transmission is finished, the node transmits during SIFS another special signal, tt_End, indicating the end of this transmission. Fig. 9 illustrates the time line of a transparent transmission. Nodes receive tt_Start and tt_End can use the timing of these signals to estimate the start and end time of the current transmission. Since all nodes that hear the current transmission transmit those signals, all nodes in the single collision domain learn the start and end time of the current transmission. Transparent transmissions needs to satisfy the following conditions: i) when overlapping with data packets, the special signal can be easily identified under low SNR without extra hardware requirements, ii) the transmission of the special signal can not disturb the reception of data packets, and iii) the reception of special signal can not disturb the reception of data packets.

Flash Signal

We use flash signals introduced by Flashback [10] as the special signal in transparent transmissions [5]. Flash signals are sinusoids with frequency equal to a certain subcarrier of the current Wi-Fi channel, and with duration equal to a OFDM symbol (4μs). Since flash signals can use any of the 36 subcarriers in a WiFi channel, there are at least 36 different flash signals, and transparent transmissions only use 2 of them for tt_Start and tt_End. Since the Fourier transform of a sinusoidal wave is delta functions, as evaluated in the experiments of Flashback [10], flash signals can be detected easily by simple peak detection algorithm without extra hardware requirements (the transmit/detect of flash signals can be achieved through software/firmware changes by reusing OFDM DSP blocks.). The ease of detection guarantees the robustness of transparent transmissions. Still, packet receptions won't be disturbed by flash signals as long as the flash rate is less than 50,000 flashes per second.

Switch to Transmit while Receiving

Due to the half-duplex property of WiFi radios, a node can not transmit a flash signal while receiving. Thus, we introduce a "switch to transmit while receiving" mechanism for nodes to transmit flash signals while receiving. A tt_padding, which contains only zeros, is inserted after the PHY header of each packet. After receiving the PHY header of a packet, a node switches to transmit tt_Start, and then switch back to continue receiving. Since the received portion that missed due to transmitting tt_Start is tt_padding, the packet can be received successfully. tt_padding also ensures that the reception of special signal does not disturb the reception of data packets. The length of tt_padding considers both switch time and signal transmission time. As will be shown in the experimental results in Section 3.4.3, we select the length of tt_padding based on the time synchronization requirement of IEEE 802.11 [1].

[5]Transparent transmissions can work with other special signals such as Gold codes [11] used by DOMINO [9] or the correlatable symbol sequences in 802.11ec [14], as long as the signals can be easily identified. Flash signals is selected due to its simplicity.

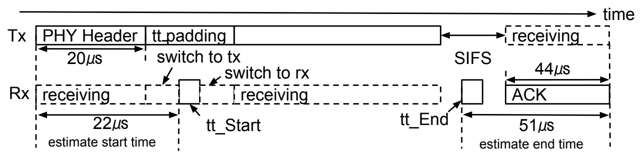


Figure 9: Time line of transparent transmissions

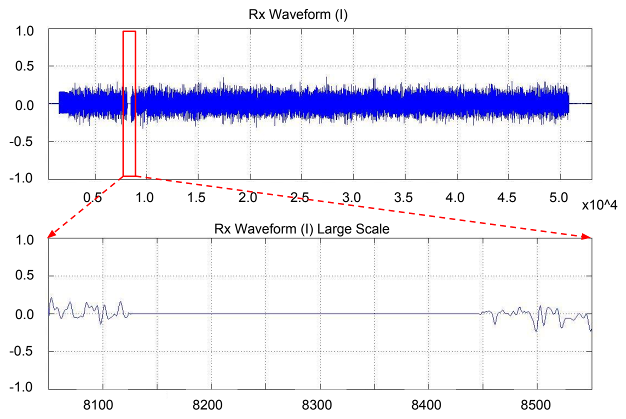


Figure 10: Rx wave of transparent transmissions

Transmit during SIFS

In order to inform other nodes the end of the current transmission, another flash signal, tt_End, is transmitted after the DATA transmission is finished. Nodes utilize the SIFS duration between DATA and ACK to transmit tt_End. Since SIFS is 10μs, the duration of tt_End is 4μs, and the receive to transmit turnaround time is smaller than 2μs, the time duration of SIFS is quite sufficient for transmitting tt_End.

3.4.3 Experimental Validation of Transparent Transmissions

Experiments using WARP [12] are carried out to validate transparent transmissions. In the experiment, a Tx generates a preamble following random data bits using 300 OFDM symbols. After receiving the preamble from the Tx, a Rx switches to transmit mode and transmits a sine wave (represents tt_Start) for 4μs. After this transmission, the Rx switches back to receive mode and keep receiving. Fig. 10 shows a received signal wave with 2 different scales; the unit of x-axis is 2.5e-8 seconds. There is a small duration (around 8μs) of "no received signal" when the Rx switches to transmit. Table 1 shows the summary of the results. The duration of "switch to transmit" measured by WARP is : (avg: 8.7μs, max: 9μs, min: 8μs). Since the duration of one OFDM symbol is 4μs, "switch to transmit" usually damages reception of 3 OFDM symbols. The total number of error symbols in the 300 symbols is: (avg: 3.2, max: 5, min: 2). Since the symbol error can be caused by channel noise, it is more accurate to estimate the damage using time duration. According to the experiment results, we use 3 OFDM symbols (12μs) for tt_padding.

3.4.4 Algorithm

Measurement	Avg	Max	Min
Duration of switching and transmitting	8.7μs	9μs	8μs
Total Number of Error Symbols	3.2/300	5/300	2/300

Table 1: Evaluation of transparent transmissions

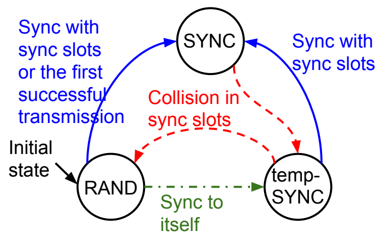


Figure 11: State Diagram of *ST*

The *Pos* synchronization and self-triggering of LWT-TT is similar to that of LWT-WC, only with small differences.

For self-triggering, every node maintains a record of "overheard nodes." Initially, a node assumes that it can not overhear any node and update this record as it starts to overhear transmissions. If the Tx of current schedule slot is recorded as "can not overhear," the node waits extra 24μs after DIFS for possible flash signals. Otherwise, the self-triggering is the same as that of LWT-WC.

For ease of *Pos* synchronization, since transparent transmissions only carry information of the start and end time of a transmission, there are 2 unknown information: i) if ACK is transmitted, and ii) the Rx and Tx addresses of the transmission. Again, nodes utilize the "overheard nodes" record. If the Rx of the transmission cannot be heard, nodes assume that ACK is transmitted and will not reset its *ST* to *RAND*. If the Tx of the transmission cannot be heard, nodes stay in *RAND* when *ST* is *RAND*, and increase *Pos* by 1 if *ST* is *SYNC*. (Note that nodes still can use timing to identify sync slots and rand slots.)

A new state of *ST*, *tempSYNC*, is introduced for ease of *Pos* synchronization. If nodes hear sync slots while its *ST* is *RAND*, it tries to synchronize using those sync slots. However, if the number of overheard sync slots is not enough for a node to figure out the position of *Pos*, a node can synchronize to itself and switches its *ST* from *RAND* to *tempSYNC* when it has the first successful transmission in a rand slot. When *ST* is *tempSYNC*, nodes setup its *Pos* (match to its own successful transmission) and transmit in its scheduled slots indicated by its *Pos*. If the transmissions (transmit using sync slots) in the scheduled slots succeed, nodes switch its *ST* from *tempSYNC* to *SYNC* after a schedule duration. Otherwise, if any collision happens in the sync slots, *ST* is switched from *tempSYNC* to *RAND*.

On the other hand, if a collision happens in a sync slot, a node will not directly switch from *SYNC* to *RAND*. It first switches from *SYNC* to *tempSYNC* for a schedule duration. If a collision happens in a sync slot again when *ST* is *tempSYNC*, it switches from *tempSYNC* to *RAND*. Fig. 11 shows the state diagram of *ST*. The rest of the algorithm is same as that of LWT-WC.

3.4.5 *Efficiency Estimation*

The protocol overhead of transparent transmissions is the extra time duration of tt_padding in a packet. Assuming a 1500byte packet transmitting in 54Mbps, the overhead caused by tt_padding (12μs) is only 3%, and this overhead decreases as the transmission rate becomes lower. Comparing to the 26% overhead caused by RTS/CTS, transparent transmissions is much more efficient.

3.5 Other challenges and considerations

3.5.1 *Supporting backward compatibility*

LWT is backward compatible, and can operate with legacy nodes without separating them to a different time duration. Legacy nodes can transmit in rand slots, in which all nodes operate DCF. If the traffic load from legacy nodes is high, the central controller can schedule special legacy-only slots that LWT nodes won't contend for. Although the probability of legacy nodes disturbing sync slots is non-zero, this probability decrease exponentially due to the exponential grow of the contention window (cw) in DCF. For example, the probability of a legacy node disturbing a sync slot is 6% (the probability of a legacy node selecting zero as its backoff number with cw=15) in the first time, 3% (cw=31) for the second time, and becomes less than 1.5% (cw$\geq$63) after the third time. The *tempSYNC* state also helps LWT nodes to be more robust against disturbance.

If there are legacy nodes that can not overhear the current transmission, the Tx will use a tt_padding of 13 OFDM symbols and the Rs transmits CTS (44 μs) with tt_Start during the tt_padding. The disturbing probability, which also decreases exponentially, is 18% in this situation (the probability of a legacy node selecting a backoff number $\leq$2, which leads to a backoff time $\leq 22\mu$s).

LWT can also use PIFS, an IFS shorter than DIFS, in sync slot to decrease the disturbing probability. PIFS is used by APs and non-AP nodes under PCF, in which the behavior of non-AP nodes is under control of the AP. Since the behavior of nodes in sync slots is also under control of the central controller, it is reasonable to allow nodes to use PIFS in sync slots.

In LWT, nodes need information of all MAC addresses in the network for LWT-CV. Since all legacy nodes need to associate with an AP, the MAC addresses of legacy nodes can be collected by APs and give to all nodes the same way as the target schedule S. Nodes can also collect legacy MAC addresses by learning from overhearing.

3.5.2 *Sleeping nodes*

LWT nodes lose *Pos* synchronization after sleeping since they cannot overhear transmissions during sleeping. As mentioned in Section 2, using fixed duration for each transmission can make sleeping nodes keep *Pos* synchronization. However, fixed transmission time requires packet aggregation and fragmentation, which introduces extra delays. Thus, instead of using fixed duration for each transmission, sleeping nodes achieve *Pos* synchronization by overhearing sync slots or synchronizing to itself through *tempSYNC* state (Fig. 11). Once a node wakes up from sleeping, it operates DCF until it hears enough sync slots to identify the position of *Pos*. However, if the node didn't hear any sync slots for a schedule duration, it switches to *tempSYNC* state when it successfully finishes a transmission in a rand slot (synchronizes to itself). If the later transmissions in sync slots succeed, nodes switches to *SYNC* state after a schedule duration. Otherwise, if any collision happens in sync slots, *ST* is switched from *tempSYNC* to *RAND*.

3.5.3 *Schedule updates and membership changes*

The target schedule is determined by the central controller using scheduling algorithms such as [15]. The target schedule is broadcasted by APs periodically. Although broadcasting is efficient, it is not reliable. Transmitting following different schedules can cause collisions in sync slots. Thus, LWT requires a mechanism that quickly indicates the update of S. The update of S is indicated by another flash signal, S_update, transmitted by APs along with DATA or ACK. Since flash signals can be transmitted frequently (50,000 flashes per second) without disturbing receptions, and can be identified even when colliding with other flash signals, nodes learn the update of S quickly and reliably. Once receive S_update, nodes operate DCF until they overhear the new S. Thus, although there might be some delay of getting the new S, nodes won't disturb sync slots once they receive S_update.

When a node wants to join the network, it operates DCF until overhearing S from APs. New nodes get Pos synchronization the same way as sleeping nodes. Each AP maintains a client list and periodically sent it to the central controller for updating S.

3.5.4 *Multiple collision domains*

So far, we propose algorithms for WiFi networks in a single collision domain. It is possible to extend them for multiple collision domains, but it is beyond the scope of this paper. The main idea is using different flash signals to delivery transmission information of different collision domains. There are 36 subcarriers that can be used by flash signals and LWT only needs 2 for each collision domain. In multiple collision domains, multiple transmissions can be scheduled in the current schedule slot. Nodes increase Pos by one only after confirming the end of transmissions of all collision domains in the current schedule slot. This can be learned by either overhearing or receiving flash signals. We leave the in-depth exploration of multiple collision domains for future work.

4. EVALUATION

In this section, we evaluate LWT using real-time experiments and ns-3 simulations [16].

4.1 Experimental Evaluation

We implement LWT in a software defined radio platform: Wireless open-Access Research Platform (WARP) v3. WARP

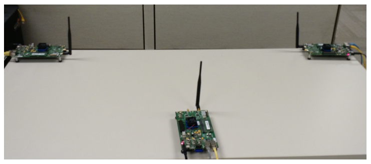


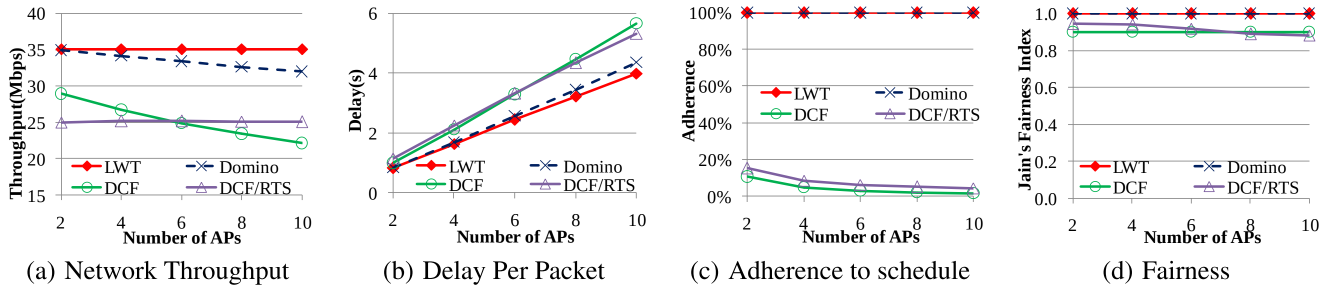
Figure 12: WARP test-bed

Link Number	LWT(Mbps)	DCF(Mbps)
1	9.00	6.62
2	8.03	8.98
3	9.00	6.62
4	8.03	9.00
Total	34.16	31.22

Table 2: Experiment results of LWT vs. DCF

Parameter	Value
Frame size	1500byte
Basic transmission rate	6Mbps
Data transmission rate	54Mbps
Slot time	9μs
SIFS	10μs
DIFS	28μs
AIFS	37μs
RX/TX switching delay	<2μs

Table 3: ns-3 parameter



(a) Network Throughput (b) Delay Per Packet (c) Adherence to schedule (d) Fairness

Figure 13: Saturated traffic with fully connected topology

supports modification and monitoring of parameters and functions in both the MAC and PHY layer, and can operate fast enough to perform WiFi communications with off-the-shelf WiFi devices.

As shown in Fig. 12, we set up 3 WARP nodes in a fully connected topology. One acts as AP and the other 2 act as STAs. The experiment is set up in an indoor environment using 802.11a in 5.18 GHz channel with 54Mbps data transmission rate. We use iperf to generate traffic on the 2 uplinks and 2 downlinks. Each link is scheduled once in S.

Table 2 shows the throughput of LWT and DCF from experimental evaluation. Comparing to DCF, LWT gives better throughput and fairness. Due to the low contention (only 3 nodes), the difference in throughput is not large. The two uplinks (link 2 and 4) have slightly better throughput in DCF. It is because DCF roughly gives each node the same access rate, which gives each node approximately $\frac{1}{3}$ access rate. LWT gives each link the same access rate, which gives each link approximately $\frac{1}{4}$ access rate. Since $\frac{1}{4} < \frac{1}{3}$, links 2 and 4 get slightly higher throughput in DCF.

Although there are only 3 nodes in the WARP-based evaluation, it proves that: (i) the schedule tracking, slot start/end time tracking, and CS/CCA mechanisms all work harmonically under LWT design to achieve better system throughput and fairness. (ii) This implementation is done without special time synchronization mechanism or hardware change support, which proves that LWT requires only the time synchronization level of 802.11 standard and can be implemented with only software/firmware changes. We would also like to point out that other features of LWT, such as slot identifications and Transparent transmissions, though not be evaluated under the three-node experiment setting, have been evaluated through other WARP experiments in previous sections.

We take these experiment results as a proof of concept of implementing LWT and evaluate the performance of LWT in more complicated situations using ns-3 simulations.

4.2 Simulation Based Evaluation

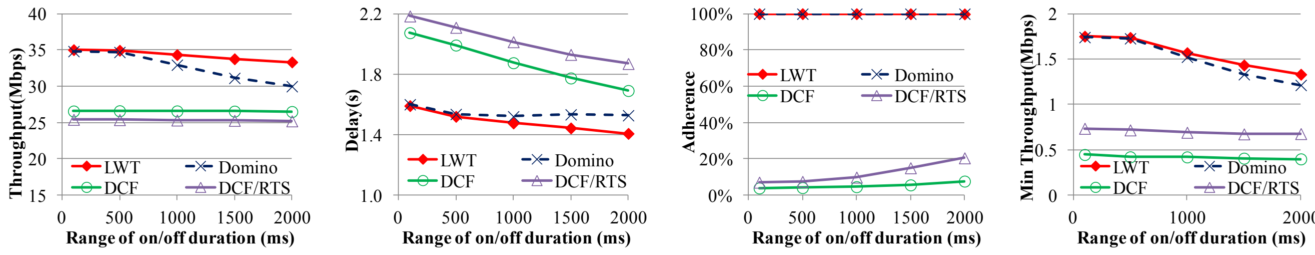
We present the performance evaluation of LWT, DOMINO, DCF, and DCF with RTS/CTS (DCF/RTS) using ns-3 simulation. Table 3 shows the simulation parameters, which follows 802.11g. In all scenarios, traffic is generated on all uplinks and downlinks, and each link is scheduled once in S.

We first evaluate the performance of each mechanism under fully connected topology (no exposed terminals and no hidden terminals). Scenarios with and without non-backlogged nodes are both considered. Then, the performance of each mechanism is evaluated under the 2 types of hidden terminal scenarios: HtRx and HtNRx.

4.2.1 *Saturated traffic in fully connected topologies*

We set up fully connected topologies with saturated traffic using different number of APs. Each AP has 2 clients. We consider different number of nodes since there can be a large diversity in node density in practice.

Fig. 13(a) measures the throughput of LWT, DOMINO, DCF, and DCF/RTS. Due to transmitting according to a schedule and thus avoiding collisions, DOMINO and LWT achieves much higher throughput than DCF and DCF/RTS. DCF performs worse as the number of APs increases because of increasing contention. DCF/RTS performs worse than DCF in low contention scenarios since RTS/CTS generates large overhead, and performs better than DCF in high contention scenarios since RTS/CTS successfully decreases the overhead of collisions. Since DOMINO requires each AP to poll its clients for queue information periodically, the throughput of DOMINO slightly decreases as the number of AP increases.



(a) Network Throughput (b) Delay Per Packet (c) Adherence to schedule (d) Minimum throughput

Figure 14: Dynamic traffic in fully connected topology

Fig. 13(b) shows the average packet delay, which is the time between a packet's arrival at the queue and the time it is successfully transmitted. DCF and DCF/RTS have higher delay due to larger contention time and more collisions. As the number of APs increases, delay increases due to longer schedule in LWT and DOMINO.

Fig. 13(c) presents the adherence of each mechanisms. Since DCF and DCF/RTS does random backoff, their adherence is very low. LWT and DOMINO follow the schedule almost perfectly. Fig. 13(d) measures fairness, and all 4 mechanisms can provide good fairness in fully connected topologies.

4.2.2 *Non-backlogged nodes in fully connected topologies*

In this section, we consider scenarios with non-backlogged nodes. We use dynamic on/off traffic in a fully connected topology with 2 APs, each with 5 STAs. We consider different traffic dynamic since there can be a large diversity in traffic patterns/loads in practice. The on/off duration of traffic is randomly determined by each link during each on/off switch, and the range of randomness is changed from 100ms to 2000ms.

Fig. 14(a) shows the throughput of each mechanism under different range of traffic on/off duration. The randomness of traffic doesn't affect DCF and DCF/RTS. As the range of traffic on/off duration increases, the average number of non-backlogged nodes increases. Throughput of LWT slightly decreases by cause of increase in rand slots, which have larger backoff time and collisions. Throughput of DOMINO decreases due to missing schedules; that is, nodes cannot get scheduled quickly when it starts to get packets in queue, and also can be scheduled while its traffic already enter the off period. This situation becomes more severe as the delay between the central controller and APs increases (the simulation setting of this delay is only 250 μs).

The packet delay of each mechanism is presented in Fig. 14(b). The delay decreases as the average number of non-backlogged nodes increases. Again, LWT and DOMINO performs better than DCF and DCF/RTS. Fig. 14(c) measures the adherence of different mechanisms. Note that when calculating adherence, rand slots are always considered as slots that successfully follow the schedule if the current scheduled link does not have packet to transmit. Again, DCF and DCF/RTS have very low adherence whereas LWT and DOMINO follow the schedule strictly.

Instead of showing fairness, which is not very meaningful in dynamic traffic (each link has different traffic load), the minimum throughput among all links is presented in Figure 14(d). LWT gives the highest minimum throughput, and there is no starvation in any of the mechanisms.

4.2.3 *Hidden terminals in a single collision domain*

Although the probability of occurrence is not high, it is important to deal with hidden terminals. In CENTAUR [7], the observed ratio of hidden terminals is around 33% to 36%. More importantly, once hidden terminal problems happen, the performance degradation brought by it can be tremendous. In Jigsaw [17], it was proposed that "co-channel interference from hidden terminals is likely the dominate cause of interference." Thus, we analyze the performance of different mechanisms in the presence of hidden terminals (HT). We set up two types of scenarios, HtRx (Fig. 8(b)) and HtNRx (Fig. 8(c)). All the topologies contains 2 APs, each has 5 STAs.

Hiddern Terminal (HtRx)

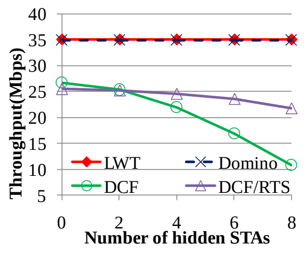
Fig. 15(a), 15(b), 15(c), and 15(d) present the throughput, packet delay, adherence, and fairness of each mechanism in HtRx scenarios. Both LWT and DOMINO are not affected by hidden terminals. As the number of hidden terminals increases, throughput of DCF dramatically decreases due to collisions, which also increases the packet delay. Throughput of DCF/RTS performs better than DCF, but it still slightly decreases when the number of hidden terminals increases. With large number of hidden terminals, DCF and DCF/RTS cannot achieve good fairness.

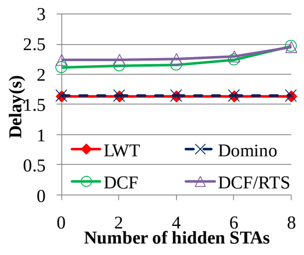
Hiddern Terminal (HtNRx)

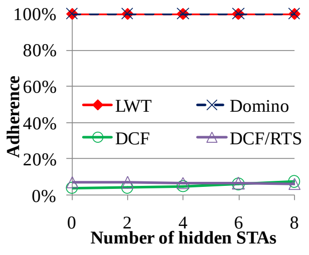
Fig. 16(a), 16(b), 16(c), and 16(d) present the throughput, maximum packet delay, adherence, and minimum throughput among all links of each mechanism in HtNRx scenarios. LWT is not affected by hidden terminals due to implementation of transparent transmissions.

As indicated in Fig. 16(d), DOMINO, DCF, and DCF/RTS can have starvation under HtNRx scenarios.

In DOMINO, the transmission of a link is triggered by the previous scheduled links. A link can only be triggered if it can hear the Rx or Tx of the triggering link. If a scheduled


(a) Network Throughput


(b) Delay Per Packet


(c) Adherence to schedule

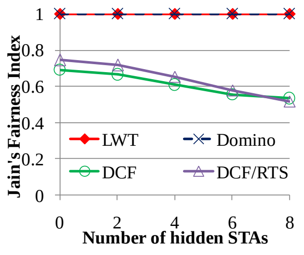
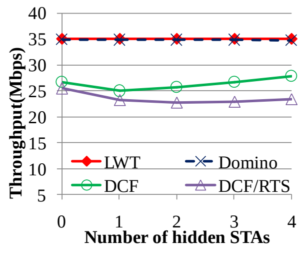
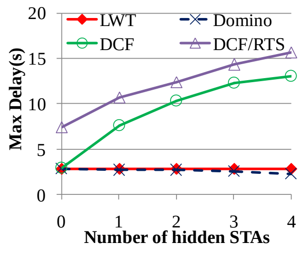
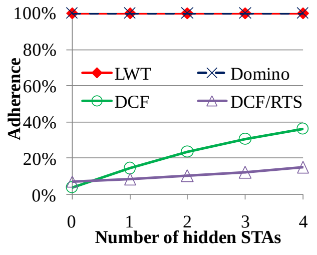

(d) Fairness

Figure 15: Hidden Terminal (HtRx)


(a) Network Throughput


(b) Max Delay Per Packet


(c) Adherence to schedule

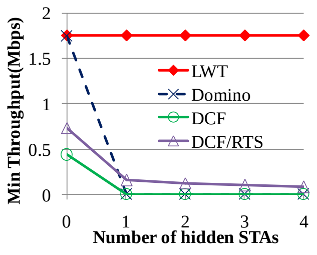

(d) Minimum throughput

Figure 16: Hiddern Terminal (HtNRx)

link cannot be triggered by any of the previous scheduled links, it will be removed from the schedule and wait to be rescheduled by the central controller. In HtNRx, there are nodes that cannot hear the Rx and Tx of certain links. These nodes have less chances to be triggered by previous scheduled links, and thus get less chance to transmit. For example, in Fig. 8(c), link (D,C) can only be triggered by link (C,D). If the link scheduled before link (D,C) is link (A,B) or link (B,A), link (D,C) will not be scheduled. This causes starvation of link (D,C).

As illustrated in Sec. 3.4.1, DCF and DCF/RTS have unfairness problems under HtNRx scenarios since the transmission of certain nodes can be easily corrupted by some other nodes. For example, in Fig. 8(c), transmission of node D can be corrupted by transmission of node A or B.

5. CONCLUSION

In this paper, we present LWT for realizing centralized WiFi scheduling using purely distributed operations. Mechanisms tackle practical challenges of non-backlogged nodes, decodability vs. detectability, hidden terminals, backward compatibility, sleeping nodes, and change to schedule are proposed. Using a WARP-based test-bed and ns-3, both experimental evaluations and simulation based analysis are carried out to evaluate LWT. Evaluation results show that LWT achieves better efficiency, delay, adherence, and fairness comparing to related works.

6. REFERENCES

[1] "IEEE 802.11: Wireless LAN MAC and Physical Layer Specifications," 2012.

[2] "Openflow: Enabling innovation in your network." [Online]. Available: http://www.openflow.org/

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

[4] "iPass WiFi Growth Map." [Online]. Available: http://www.ipass.com/wifi-growth-map/

[5] B. O'Hara and P. J. K. Calhoun, "Configuration and Provisioning for Wireless Access Points Problem Statement," 2005.

[6] P. Calhoun *et al.*, "Light Weight Access Point Protocol," 2007.

[7] V. Shrivastava *et al.*, "CENTAUR: Realizing the Full Potential of Centralized WLANs Through a Hybrid Data Path," in *ACM MobiCom*, 2009.

[8] Z. Zeng, Y. Gao, K. Tan, and P. R. Kumar, "CHAIN: Introducing Minimum Controlled Coordination into Random Access MAC," *IEEE INFOCOM*, 2011.

[9] W. Zhou *et al.*, "DOMINO: Relative Scheduling in Enterprise Wireless LANs," *ACM CoNEXT*, 2013.

[10] A. Cidon, K. Nagaraj, S. Katti, and P. Viswanath, "Flashback: Decoupled Lightweight Wireless Control," *SIGCOMM*, 2012.

[11] R. Gold, "Optimal binary sequences for spread spectrum multiplexing," *IEEE Transactions on Information Theory*, 1967.

[12] "WARP." [Online]. Available: http://warpproject.org

[13] S. Haykin, *Communication Systems*. WILEY, 2001.

[14] E. Magistretti, O. Gurewitz, and E. W. Knightly, "802.11ec: Collision avoidance without control messages," *MobiCom*, 2012.

[15] H. Fattah and C. Leung, "An Overview of Scheduling Algorithms in Wireless Multimedia Networks," *IEEE Wireless Communications*, 2002.

[16] "ns-3." [Online]. Available: https://www.nsnam.org/

[17] Y.-C. Cheng *et al.*, "Jigsaw: Solving the Puzzle of Enterprise 802.11 Analysis," *ACM SIGCOMM*, 2006.