Order Matters: Transmission Reordering in Wireless Networks

Justin Manweiler
Duke University
Durham, NC, USA
jgm@cs.duke.edu

Naveen Santhapuri
Univ. of South Carolina
Columbia, SC, USA
santhapu@cec.sc.edu

Souvik Sen
Duke University
Durham, NC, USA
ssen@cs.duke.edu

Romit Roy Choudhury
Duke University
Durham, NC, USA
romit@ee.duke.edu

Srihari Nelakuditi
Univ. of South Carolina
Columbia, SC, USA
srihari@cse.sc.edu

Kamesh Munagala
Duke University
Durham, NC, USA
kamesh@cs.duke.edu

ABSTRACT

Modern wireless interfaces support a physical layer capability called *Message in Message* (MIM). Briefly, MIM allows a receiver to disengage from an ongoing reception, and engage onto a stronger incoming signal. Links that otherwise conflict with each other, can be made concurrent with MIM. However, the concurrency is not immediate, and can be achieved only if conflicting links begin transmission in a specific order. The importance of link order is new in wireless research, motivating MIM-aware revisions to link scheduling protocols. This paper identifies the opportunity in MIM-aware reordering, characterizes the optimal improvement in throughput, and designs a link layer protocol to achieve it. Testbed results confirm the performance gains of the proposed system.

Categories and Subject Descriptors

C.2.1 [**Network Architecture and Design**]: Centralized Networks, Wireless communication; C.2.5 [**Local and Wide-Area Networks**]: Access schemes

General Terms

Algorithms, Design, Experimentation, Performance

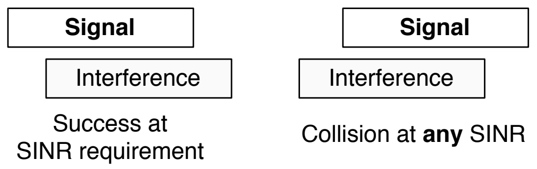
Keywords

EWLANs, MIM, Interference-awareness, Scheduling

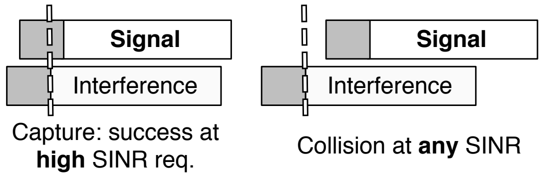
1. INTRODUCTION

Physical layer research continues to develop new capabilities to better cope with wireless interference. One development in the recent past is termed *Message in Message* (MIM). Briefly, MIM allows a receiver to disengage from an ongoing signal reception, and engage onto a new, *stronger* signal. If the ongoing signal was not intended for the receiver (i.e., interference), and if the new signal is the actual signal of interest (SoI), then re-engaging onto the new signal is beneficial. What would have been a collision at a conventional receiver may result in a successful communication with MIM-capable hardware. For a better understanding of MIM, we contrast it with the traditional definition of collision. More importantly, we differentiate MIM from the existing notion of *physical layer capture*.

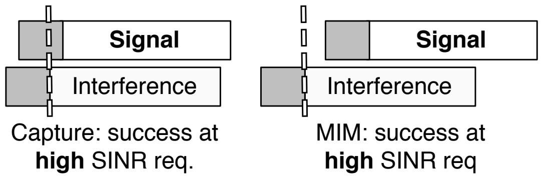
Collision was widely interpreted as follows: A signal of interest (SoI), however strong, cannot be successfully received if the receiver is already engaged in receiving a different (interfering) signal. Most simulators adopt this approach, pronouncing both frames corrupt. If, on the other hand, the SoI arrives before the interference, and satisfies the required SINR, the signal can be successfully decoded. The figure below shows the two cases.



Physical Layer Capture was later understood through the systematic work in [1, 2]. Authors showed that capture allows a receiver to decode a later-arriving SoI, provided the start of both the SoI and the interference are within a preamble time window. The figure below illustrates this. While valuable in principle, the gain from capture is limited because the 802.11 preamble persists for a short time window (20 μs in 802.11a/g/n). If the SoI arrived later than 20 μs, both frames will be corrupt.


Signal
Interference
Capture: success at **high** SINR req.
Signal
Interference
Collision at **any** SINR

Message in Message (MIM) is empowering because it enables a receiver to decode an SoI, even if the SoI arrives after the receiver has already locked on to the interference [3]. Of course, the required SINR is higher for re-locking onto the new signal. Conversely, if the SoI arrives earlier than the interference, MIM is same as traditional reception. The following figure illustrates the MIM advantage.


Signal
Interference
Capture: success at **high** SINR req.
Signal
Interference
MIM: success at **high** SINR req

To summarize, unlike traditional receivers, an MIM-capable receiver can decode a strong signal of interest even if it arrives later than the interference. Of course, the required SINR to decode the later packet is relatively higher (≈10 dB) compared to when the packet arrives earlier (≈4 dB).

What makes MIM feasible? An MIM receiver, even while locked onto the interference, "simultaneously searches" for a new (stronger) preamble. If a stronger preamble is detected (based on a high correlation of the incoming signal with the known preamble), the receiver unlocks from the ongoing reception, and re-locks on to this new one. The original signal is now treated as interference, and the new signal is decoded. The ability to extract a new signal, even if at a higher SINR, can be exploited to derive performance gains. We motivate the opportunity with an example.

Link Layer Opportunity

Consider the example in Figure 1. For R1, node AP1 is the transmitter, while AP2 is the interferer (and the vice versa for R2). When using MIM receivers, observe that the two links can be made concurrent only if AP1→R1 starts before AP2→R2. Briefly, since AP2→R2 supports a higher SINR of 11 dB, it can afford to start later. If that is the case, R2 will begin receiving AP1's transmission first, and later re-lock onto AP2's new signal which is more than 10 dB stronger than AP1's. However, in the reverse order, R1 will lock onto AP2's signal first, but will not be able to re-lock onto AP1, because AP1's signal is not 10 dB stronger than AP2's (it is only 5 dB stronger). Therefore, R1 will experience a collision. As a generalization of this example, MIM-aware scheduling protocols need to *initiate weaker links first, and stronger links later*. Appropriate ordering of the links can improve spatial reuse.

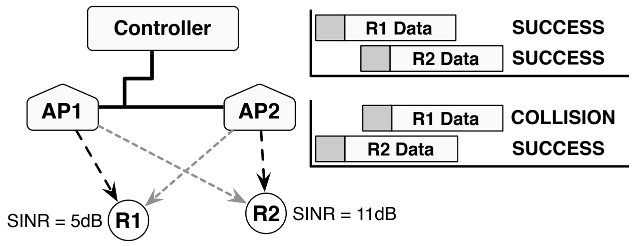


Figure 1: AP1→R1 must start before AP2→R2 to ensure concurrency. If AP2 starts first, R1 locks onto AP2 and cannot re-lock onto AP1 later.

In a larger network, choosing the appropriate set of links from within a collision domain, and determining the order of optimal transmission, is a non-trivial research problem. IEEE 802.11 is unable to ensure such orderings, failing to fully exploit MIM-capable receivers. Perhaps more importantly, *graph coloring based scheduling* approaches are also inapplicable. This is because graph coloring assumes symmetric conflicts between links. MIM link conflicts are *asymmetric* (i.e., depend on relative order), and may not be easily expressed through simple abstractions. In response to this research problem, this paper proposes *Shuffle*, an MIM-aware link layer solution that reorders transmissions to extract performance gains. Our main contributions are:

(1) Identifying the opportunities with MIM. We use MIM-enabled Atheros 5213 chipsets, running the MadWiFi driver to verify that transmission order matters.

(2) Analysis of optimal performance with MIM. We show that MIM-aware scheduling is NP-hard, and derive upper bounds on performance using integer programming. CPLEX results show that the optimal gain from MIM is substantial, hence, worth investigating.

(3) Design of an MIM-aware system, *Shuffle*, for enterprise wireless LANs. Links within the same collision domain are suitably reordered to enable concurrency. A measurement-based protocol engine coordinates the overall operation, and copes with failures.

(4) Implementation and deployment within our university building. Testbed results demonstrate practicality and consistent performance improvements over 802.11 and order-unaware TDMA.

2. VERIFYING MIM

We validate the existence of MIM capabilities in commodity hardware using a testbed of Soekris embedded PCs, equipped with Atheros 5213 chipsets running the MADWiFi driver. The experiment consists of two transmitters with a single receiver placed at various points in-between (Figure 2). This subjects the receiver to varying SINRs. To ensure continuous packet transmissions from the transmitters, we modify the MADWiFi driver to disable carrier sensing, backoff, and the inter-frame spacings. To time-stamp transmissions, a collocated monitor is placed at each transmitter. Each monitor is expected to receive all packets from its collocated transmitter, while the in-between receiver is expected to experience some collisions. Merging time-stamped traces from the two monitors and the receiver, we were able to determine the relationship between *transmission order* and *collision*.

Figure 2 shows delivery ratios for different order of packet arrivals, at different positions of the receiver. For all these positions, the interference was strong, i.e., in the absence of the SoI, we verified that the interfering packets were received with high delivery ratio. Under these scenarios, observe that when the receiver is very close to the transmitter (positions 1, 2, and 3), it achieves a high delivery ratio independent of the order of reception. This is a result of achieving a large enough SINR such that both SoI-first (SF) and SoI-last (SL) cases are successful. However, when the receiver moves away

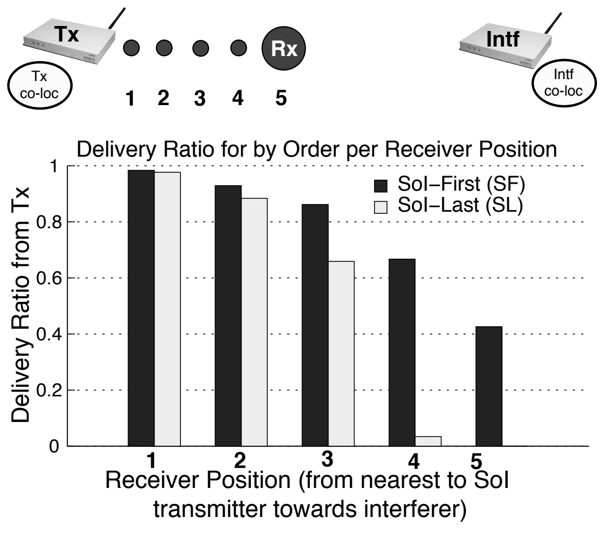


Figure 2: Testbed confirms MIM. Rx receives from Tx (at 5 positions) in presence of Interference (Intf).

from the transmitter (positions 4 and 5), the SINR is only sufficient for the SF case, but not the SL case. Hence, at position 4, only 4% of the late-arriving packets get received, as opposed to 68% of the early-arriving packets. This validates the existence of MIM on commercial hardware, and confirms that enforcing the correct order among nearby transmissions can be beneficial.

3. MIM: OPTIMALITY ANALYSIS

A natural question to ask is *how much gain is available from MIM?* Characterizing the optimal gain will not only guide our expectations, but is also likely to offer insights into MIM-aware protocol design. Towards this, we first prove that MIM-aware scheduling is NP-hard, and use integer programming methods to characterize the performance bounds for a large number of topologies. We compare the results with an MIM-incapable model.

THEOREM 1. *Optimal MIM Scheduling is NP-hard.*

PROOF. Consider the problem of *Optimal Link Scheduling* with *MIM-capable nodes*. An optimal schedule consists of a link selection and a corresponding MIM-aware ordering, that together maximizes the network throughput. Assume a polynomial time algorithm exists to provide the optimal MIM link scheduling from known network interference relationships. Conventional (no-MIM) link scheduling is a known NP-complete problem, reduced from *Maximum Independent Set* [4]. Therefore, if our assumption is true, then it would be possible to find the optimal MIM-incapable link schedule in polynomial time, just by restricting the SoI-Last SINR threshold to infinity in our algorithm (i.e., ensuring later-arriving signals are never decoded). This contradiction proves that optimal MIM link scheduling is NP-hard. □

3.1 Optimal Schedule with Integer Program

To quantify the performance gains from MIM, we model MIM-capable and MIM-incapable networks, and compare their optimal throughput over a variety of topologies. The networks consist of multiple access points (APs), each associated to a number of clients. Each transmission produces an interference footprint derived from a path loss index of 4. With MIM-capable receivers, the SoI-First (SF) SINR requirement is 4 dB, while the SoI-Last (SL) requirement is 10 dB [3]. With MIM-incapable receivers, the SINR requirement for reception is uniformly 4 dB, and later-arriving packets cannot be received. We construct linear (binary integer) programs to compute the maximum number of concurrent links meeting the required SINR thresholds. The linear program also produces the order. Fairness is not considered in this analysis. To make our model solvable within reasonable execution time, we make the following simplifying assumptions. (1) All clients are associated to the AP with the strongest signal strength. (2) A frame is always pending on any AP-to-client link. (3) Only a single data rate r is used throughout the network. In section 4.2 we consider MIM scheduling with rate control.

Let a and b be arbitrary nodes in a wireless network and N be the set of all such nodes. We define the boolean relation $\text{RANGE}(a \to b) = true \iff b$ is within the transmission range of a. Let $\mathcal{L}$ denote the set of wireless links l_{ab} such that $\text{RANGE}(a \to b) = true$.

Let $S_{l_{ba}}$ denote the SoI strength received on link l (by node a from an active transmission by node b), measured in units of power. Equivalently, let $I(m_{cd} \to l_{ab})$ denote interference received on link l (by node b) due to a concurrent transmission on link m (from node c). The following table summarizes the parameters and variables. Under an assumption of additive multiple interference and non-fading channels, the maximum link concurrency of a given wireless network under MIM-aware MAC can be found using the IP presented below.

Parameter	Meaning	
N	Set of all nodes.	
$\mathcal{L}$	Set of al links l_{ab} s.t. $\text{RANGE}(a \to b)$	
S_l	Signal strength on link l.	
$I(m \to l)$	Interference on link l from link m.	
τ^{SF}	*Sender First* capture threshold.	
τ^{SL}	*Sender Last* capture threshold.	
Variable	**Value**	**Meaning**
$x_{l_{ab}}$	1	Link l ($a \to b$) in use.
	0	Otherwise.
y_{lm}	1	Link l starts before link m.
	0	Otherwise.

Maximize: $\sum_{\forall l \in \mathcal{L}} x_l$

Subject To:

$\forall a \in N$
$$\sum_{\forall b \in N | l_{ab} \in \mathcal{L}} x_{l_{ab}} + \sum_{\forall b \in N | l_{ba} \in \mathcal{L}} x_{l_{ba}} \leq 1 \tag{1}$$

$\forall l, m \in \mathcal{L} | l \neq m$
$$x_l + x_m - 1 \leq y_{lm} + y_{ml} \leq \min(x_l, x_m) \tag{2}$$

$\forall l, m, n \in \mathcal{L} | l \neq m \wedge l \neq n \wedge m \neq n$
$$x_l + x_m + x_n - 2 \leq y_{lm} + y_{mn} + y_{nl} \leq 2 \tag{3}$$

$\forall l \in \mathcal{L}$
$$\sum_{\forall m \in \mathcal{L} | m \neq l} y_{ml} \cdot I(m \to l) \leq \frac{S_l}{10^{(\tau^{SLC}/10)}} \tag{4}$$

$$\sum_{\forall m \in \mathcal{L} | m \neq l} (y_{ml} + y_{lm}) \cdot I(m \to l) \leq \frac{S_l}{10^{(\tau^{SF}/10)}} \tag{5}$$

Aggregate network throughput may be computed as $r \cdot \sum_{\forall l \in \mathcal{L}} x_l$.

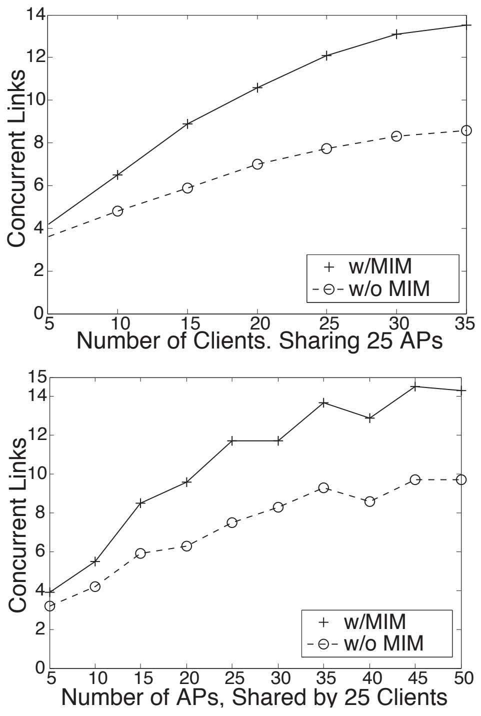


Figure 3: MIM can provide large concurrency gains.

THEOREM 2. *Any* $0/1$ *solution to the above IP satisfies the following:*

1. $x_l = 1$ *encode the active links.*
2. *The* $y_{lm} = 1$ *variables encode a total ordering on the active links where* m *is made active after* l*. (Constraints (2) and (3)).*
3. *The set of active links along with their ordering satisfies the interference constraints, and is hence feasible. (Constraints (4) and (5)).*

The optimal solution to the IP is therefore precisely the optimal solution of interest.

PROOF. Consider constraints (2) and (3). Suppose first that all the $x_l = 1$. Then, constraints (2) and (3) are equivalent to: $y_{lm} + y_{ml} = 1$, and $1 \leq y_{lm} + y_{mn} + y_{nl} \leq 2$. We interpret the variable y_{lm} as follows: $y_{lm} = 1$ if m follows l in the ordering, and 0 otherwise. Note that the constraints exactly encode the following information: In any ordering, for every l, m, either l appears after m or the other way around; and for every l, m, n, it cannot happen that l follows m, m follows n, and n follows l. It is shown in literature that these constraints are necessary and sufficient to encode a complete ordering.

Now suppose the x_l are not all 1. In that case, $y_{lm} = 1$ only if m follows l in the ordering *and* both l and m are active, so that $x_l = x_m = 1$. In this case, the constraints (2) and (3) are meaningful only if all the corresponding x variables are all 1, which means all the corresponding links are active. For these links, the constraints (2) and (3) encode a total ordering.

The constraints (4) and (5) encode the interference constraints. For any l, the only y_{ml} that contribute to constraint (4) are those with $y_{ml} = 1$, which are precisely the m that are active and precede l. Further, the LHS of the constraint is non-zero only if l itself is active. A similar reasoning shows the validity of constraint (5). $\square$

3.2 Results

We used CPLEX to solve many instantiations of the IP. In Figure 3, we present results for topologies of grid-aligned access points and randomly-placed clients. All clients associate to the the AP from which it receives the strongest signal. Each data point is sampled as the arithmetic mean of 15 trials. Results show that ideal concurrency gains can be large with MIM-capable receivers. This provides a sound motivation for designing and implementing MIM-aware protocols.

4. SHUFFLE: SYSTEM DESIGN

We propose Shuffle, an MIM-aware link layer solution that reorders transmissions to improve concurrency. Shuffle targets enterprise WLAN (EWLAN) environments, such as universities, airports, and corporate campuses [5, 6]. In EWLANs, multiple access points (APs) are connected to a central controller through a high speed wired backbone (Figure 1 and 7). The controller coordinates the operations of APs. The APs follow the controller's instructions for packet communication.

The reason to target EWLAN architectures is two-fold. (1) EWLANs are becoming popular in single administrator environments [6–10]. Developing this platform on sound physical and link layer technologies can further drive its proliferation. (2) MIM-aware scheduling is hard, and a systematic approach should perhaps start from a more tractable system. EWLANS present a semi-centralized platform, amenable to experimentation. Exploiting MIM on this architecture is itself a rich, unexplored, research area, that could lay the foundation for decentralized systems.

4.1 Protocol Design

We first sketch the three main operations of Shuffle. Figure 4 illustrates their interactions.

1. **Conflict Diagnosis:** Shuffle characterizes the interference relationships between links. In the steady state, links that have been concurrent in the past are scheduled concurrently, while those in conflict (across both orders) are serialized. With link failures, the interference relationships are appropriately revised. Over time, this continuously learned knowledge base becomes an *interference map* against which future transmissions may be scheduled.
2. **Packet Scheduling:** From the learned interference relationships, an MIM-aware scheduler (running at the controller) computes batches of concurrent links and their relative transmission order.
3. **Schedule Execution:** After scheduling batches of concurrent packets, the controller notifies relevant APs when transmissions should occur. APs maintain precise time synchronization with the controller so that expected transmission orderings may be accurately executed.

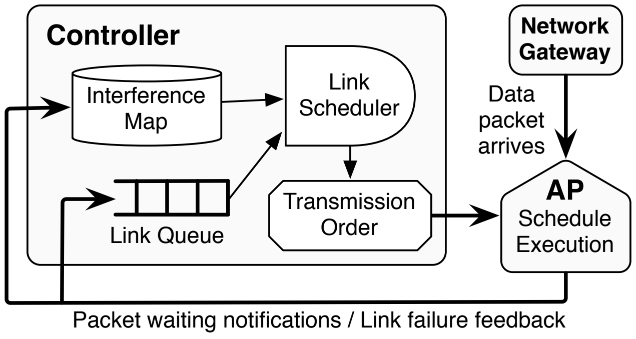


Figure 4: Flow of Shuffle operations. Data packets arrive from the network gateway and are enqueued at an AP. The AP notifies the controller of the waiting outbound packet. The controller inserts the corresponding AP-client pair into a network-wide link queue, and eventually schedules this link as part of a concurrent batch. The AP dequeues and transmits the packet according to the controller's prescribed schedule, and subsequently notifies the controller of all failures. The controller utilizes this feedback for loss recovery and conflict diagnosis.

(1) Conflict Diagnosis

Scheduling algorithms require knowledge of link conflicts. MIM increases the difficulty of inferring conflicts because it introduces a dependency on transmission order. Shuffle overcomes this difficulty by admitting some inaccuracy in the interference map. The main idea is to speculate that some permutations of links are concurrent, maintain their delivery ratios over time, and use these delivery ratios to infer conflict relationships. The learnt relationships can be used to speculate better, and schedule future transmissions. In the steady state, *learning aids scheduling which in turn aids learning*, thereby sustaining a reasonably updated interference map. Of course, failures happen when the interference map becomes inconsistent with the time-varying network conditions. Shuffle copes through retransmissions.

Speculating and Verifying Concurrency: While bootstrapping, the central controller assumes (optimistically) that any set of links formed by distinct APs may be scheduled concurrently. Upon link failure, detected by per-client acknowledgments, APs request the controller to reschedule the lost packet. The controller revisits the unsuccessful schedule to determine all the active APs in that schedule; it reduces the delivery ratio of the failed link against each of these APs. When no rescheduling request is received, the controller assumes successful delivery, and appropriately updates the delivery ratio for each link, against all interfering APs. For example, when link l_i, belonging to a schedule S, is successful, the controller gains confidence that l_i is truly concurrent with all other APs in S. More precisely, it gains confidence that l_i can sustain *earlier-arriving* interferences from APs that started before l_i, and *later-arriving* interference from APs that started after l_i. The delivery ratios for each link-AP pair are maintained for both orders, and are updated appropriately. Figure 5 shows the data structures maintained at the controller. When the delivery ratio between l_i and an interfering AP falls below a threshold, the controller will either enforce schedules where l_i starts first, or, (depending on the severity) will pronounce l_i and the interfering AP to be in complete conflict. Link-AP pairs in complete conflict are scheduled in separate batches thereafter.

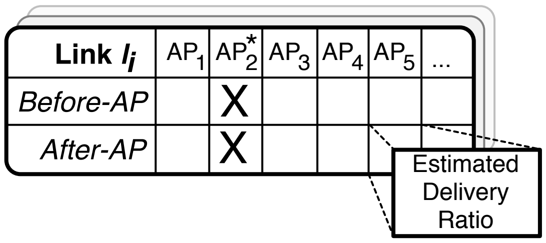


Figure 5: Per-link data structure at controller. AP$_2$ is the transmitter for link l_i.

Reviving Concurrency. Link-AP pairs in complete conflict may become concurrent in the future. Unless such concurrency is revived, the network may degenerate to a very conservative (serialized) schedule. To this end, Shuffle uses a "forgetting" mechanism. Over time, the controller assumes that conflicting link sets and orderings may have become concurrent and artificially improves recorded delivery ratios. When delivery ratios rise above the threshold requirements, previously conflicting link sets and orderings are attempted anew.

Opportunistic Learning. To update the interference map more frequently, Shuffle takes advantage of opportunistic overhearing. For instance, a client C3 that *overhears* a packet from AP5 at time t_i, can piggyback this information in an ACK packet that it sends in the near future. The controller has a record of which other APs were transmitting at t_i. Assuming AP7 was, the controller can immediately deduce that link (AP5 $\rightarrow$ C3) can be concurrent with a transmission from AP7. The exact order for this concurrency can also be derived since the controller also remembers the relative transmission order between AP5 and AP7, from past time t_i. Continuous overhearing of packets and piggybacking in ACKs can considerably increase the refresh rate of the interference map. Convergence time can reduce, facilitating better scheduling.

(2) Packet Scheduling

Given the interference map of the network, the MIM-aware scheduler selects an appropriate batch of packets from the queue and prescribes their order of transmission. To maximize throughput, it should schedule the largest batch of packets that can be delivered concurrently without starving any client. As noted earlier, optimal MIM-aware scheduling is NP-hard and graph coloring approaches are not applicable because MIM conflicts are asymmetric in nature. Thus, new algorithms are required for effective MIM-aware scheduling. In this section, we consider packet scheduling with fixed rate. In section 4.2, we discuss how the Shuffle controller incorporates active rate control into its scheduling decisions.

Feasibility of a Schedule. An MIM-aware concurrent link schedule S consisting of *ordered* links l_1 through l_n may be considered *feasible* if and only if for all $l_i \in S$, all AP(l_i) are unique; all Client(l_i) are unique; l_i can sustain *earlier-arriving* interference from all APs starting before l_i; and l_i can sustain *later-arriving* interferences from from APs starting after l_i. Given Q, a network-wide queue of all packets waiting for download transmissions, one possible brute-force scheduling

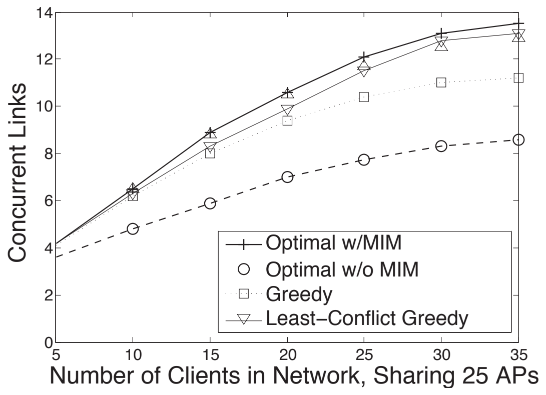


Figure 6: Heuristics for MIM-aware scheduling.

technique would be to generate all $B \subset Q$ such that B is feasible. Assuming a fixed data rate, the highest throughput schedule would be the feasible subset of Q with maximum cardinality. While this approach may be plausible with small queue sizes, it is imperative that we develop more computationally efficient heuristics for wider applicability. To this end, we present two sub-optimal but practical (greedy) heuristics.

Greedy Algorithm. A simple greedy algorithm would be to consider the packets in the FIFO order for inclusion in the batch. Initially, the batch B is set to empty. Then, each packet is attempted in turn to check if it is feasible to add it to the batch. Since the ordering of packets in the batch may affect its feasibility, a packet may need to be tried at different positions. Once a feasible ordering is found, the packet is inserted in that position in the batch. While this greedy scheme may not achieve optimal concurrency, it protects the clients against starvation. In every round, the first packet in the queue is guaranteed to get scheduled, and hence, every conflicting packet will progress by at least one position in the queue. As a result, it is guaranteed to get transmitted within a bounded number of batches. A reasonable fairness is also achieved through this simple scheme. The worst-case time complexity of the basic greedy algorithm is $O(n^2)$, where n is the number of packets in the queue. Pseudocode is presented in Algorithm 1.

Algorithm 1 Greedy.

```
1: Let $Q$ be the first $L$ packets in the queue.
2: $B := \emptyset$
3: for all Packets $p \in Q$ do
4:     for $j = 0$ to $|B|$ do
5:         if $noConflict(p, j, B)$ then
6:             Add $p$ to $B$ in position $j$
7: Return $B$
```

Least-Conflict Greedy. A conflict-oriented greedy metric may offer higher concurrency. The basic idea, with the pseudocode given in Algorithm 2, is as follows. Each of the packets in the queue can be checked to see the type of pair-wise conflict it has with all other packets in the queue (line 3). Each packet can be assigned a *score* based on such conflicts. For example, while computing the conflict of packet P_i, it is compared with every other packet P_j in the queue. If P_i and P_j are found to be concurrent, irrespective of their temporal order, then P_i's conflict score remains unchanged. However, if P_i must begin earlier than P_j, then the conflict score for P_i is incremented by one (line 5). If P_i can begin later, then again, P_i's conflict score remains unchanged. The controller computes the conflict score for each packet, and sorts the packets in increasing order of this score (line 6). Then, the controller performs the basic greedy algorithm on this order of packets (line 7). The intuition is that packets with fewer conflicts will be inserted early in the batch, potentially accommodating more concurrent links. The time complexity of Least-Conflict Greedy is also $O(n^2)$.

Without the hard scheduling guarantee of first packet inclusion in every batch, clients may encounter unfairness, and even starve. To cope with this, an aging factor can be introduced along with the conflict scores (line 2). Packets that experience prolonged queuing delay receive a proportional score reduction. Over time, the packet is likely to have a low score, and hence will be certainly scheduled by the controller. This is likely to achieve better concurrency compared to the simple greedy scheme at the expense of higher unfairness.

Algorithm 2 Least-Conflict Greedy(Q).

```
1: for all Packets $p \in Q$ do
2:     $score[p] := -age(p)$
3: for all Packets $p, q \in Q$ do
4:     if $source(p) \neq source(q) \wedge dest(p) \neq dest(q) \wedge$
       $p.mustStartBefore(q)$ then
5:         $score[p] := score[p] + 1$
6: Sort $Q$ by increasing score
7: Return $Greedy(Q)$
```

Comparison with Optimal. Figure 6 compares the concurrency of proposed heuristics to that of the optimal with and without MIM (derived from the integer program). Least-Conflict Greedy scheduling achieves near-optimal concurrency and provides substantial improvement over the naive heuristic.

(3) Schedule Execution

The controller repeatedly runs the scheduling heuristic on the queue of wireless-bound packets, and selects batches of ordered packets. Packets are actually queued at the respective APs, while the controller is only aware of *<AP, packet-destination>* link identifier tuples. For each link in the batch, the controller notifies the corresponding AP of the precise *duration of stagger*. By maintaining tight time synchronization with the controller (discussed in Section 5), APs are able to execute the staggered transmission schedule, illustrated in Figure 7. In this example, transmissions are staggered in the order AP1→C13 before AP3→C32 before AP2→C21. Backoff durations and RTS/CTS handshakes are not necessary because the scheduler accounts for link conflicts based on the interference map. Of course, transmission losses will still occur due to a variety of unpredictable reasons. Loss detection and recovery are certainly necessary.

Loss Detection and Recovery. Shuffle requires client acknowledgments for loss recovery and delivery ratio estimation. Shuffle schedules periodic upload time windows (UTW), reserved for ACKs and other client upload traffic. At the expiration of a UTW, the AP can deduce reception failures for the packets transmitted in the preceding download period. For each failed

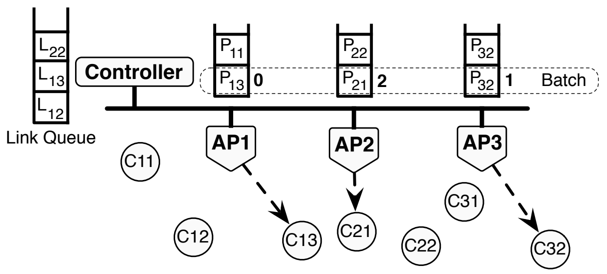


Figure 7: Packet batch with staggered transmission times. AP1 starts first, followed by AP3, then AP2.

reception, APs place the packet on a high-priority retransmission queue, and send a negative acknowledgment (NACK) to notify the controller of the loss. The controller adjusts the corresponding delivery ratios and schedules a retransmission. The AP retransmits the failed packet prior to any new packet to the same client, reducing out-of-order packet delivery.

4.2 Design Details

Practical considerations arise while translating the Shuffle protocol into a functional system.

Rate Control. In the preceding discussion on packet scheduling, we assume for simplicity that all network links operate at a fixed rate. A link's tolerance to an interference source is dependent on its operating data rate and channel quality. A practical approach must jointly consider link concurrency decisions with rate control. In view of implementation complexity, Shuffle adopts a simple strategy based on recent delivery ratios of a link at different data rates. The approach is adapted from the popular SampleRate protocol [11], as follows.

Shuffle maintains independent rate control state for each link-interferer pair. With knowledge of delivery ratios at each link, the controller runs a rate control algorithm RateControl(l, i) to select the best rate for link l in presence of interfering AP i. Observe that this rate is the *best known rate* at which link l and AP i have been successful in the recent past. Not all rates may have been attempted recently, hence, this is only a heuristic. In the presence of more APs in the concurrent batch, the rate assigned to the link is conservatively chosen as the minimum among the best known rates for each AP. Once a batch of links have been formed, the controller sorts the rates in increasing order. Lower rates imply weaker links, suggesting that it is beneficial to start them earlier. Shuffle staggers each of the links to match the sorted order of the rates. Where two links share the same rate, they are staggered in order of increasing delivery ratio (offering the weaker link a greater chance of success). As time progresses, the controller gradually increases the rates of links that attain high delivery ratios. When delivery ratios go down, the rates are reduced [11].

Upload Traffic. The controller must account for client-to-AP transmissions in its schedules. For loss detection and upload traffic, the controller frequently reserves a network-wide upload time window (UTW). During each UTW, clients contend for channel access using CSMA (for simplicity). APs notify their clients about the periodicity and duration of UTWs through beacons. Thus, the controller may dynamically adapt the UTW schedule once per beacon interval, in response to changes in the relative bidirectional traffic load. Given that UTWs may be scheduled frequently, each in the order of a few packets, division of time into upload and download windows is not expected to substantially impact latency. Since Shuffle achieves time synchronization on the order of 20 μs (Section 5), such division is practical.

Controller Placement. A number of placement options exist for installing the controller into the network. The controller may be collocated with the network gateway, allowing it to create MIM-aware schedules as the packets pass through the gateway. In reality, proprietary router software and administrative restrictions may impose practical constraints on collocation. To circumvent this, we propose to run our controller module on any one of the APs. Because of our lightweight scheduling heuristics, we find relatively low CPU utilization for the controller process ($\approx$20%). When an AP receives a packet destined for its client, it sends a notification to the controller over the wired ethernet segment. The AP queues the packet until it receives a reply from the controller containing the corresponding transmission schedule.

5. SHUFFLE: IMPLEMENTATION

Testbed Platform

We evaluated our fully-functional Shuffle implementation on a testbed consisting of laptops, running Linux kernel 2.6.27 and equipped with Atheros chipset D-Link DWA-643 ExpressCard interfaces, Soekris embedded PCs running Metrix Pyramid Linux with Atheros 5213 chipset Mini PCI interfaces, and a high performance Lenovo server. The laptops served as APs, clients, and the controller. Soekris devices were used as additional clients. The server functioned as a high-volume data source, representing the network gateway.

Shuffle's functional logic (including conflict diagnosis, MIM scheduling, and loss recovery) are implemented through element extensions to the *Click Modular Router*. Our tests assume the wireless link to be the bottleneck for all flows. Thus, in the steady state, our gateway module injects CBR UDP traffic (in 1500B datagrams) to each AP at a rate just exceeding the maximum theoretical wireless bandwidth. This ensures that APs are always backlogged with wireless-bound traffic.

To implement Shuffle and TDMA schedule execution, we customized the MadWiFi 802.11 (madwifi-hal-testing, revision 3879) driver. By modifying the MadWiFi `txcont` configuration command, a driver `ioctl` call, we can selectively disable hardware carrier-sense, virtual carrier sense, backoff, and DIFS/EIFS/SIFS intervals on the wireless interfaces. This allows Shuffle to schedule its own stream of packets without 802.11-specific timing constraints. To allow precise transmission timing, we provide a mechanism inside the MadWiFi driver that transmits a packet on the basis of the 802.11 Timing Synchronization Function (TSF) clock. For synchronization between APs and controllers, we modified the Sky2 (v1.22) ethernet driver to include the 802.11 TSF timestamp in ethernet packets. With extensive optimization, we have been able to achieve synchronization in the order of 20 μs. We report the relevant details next.

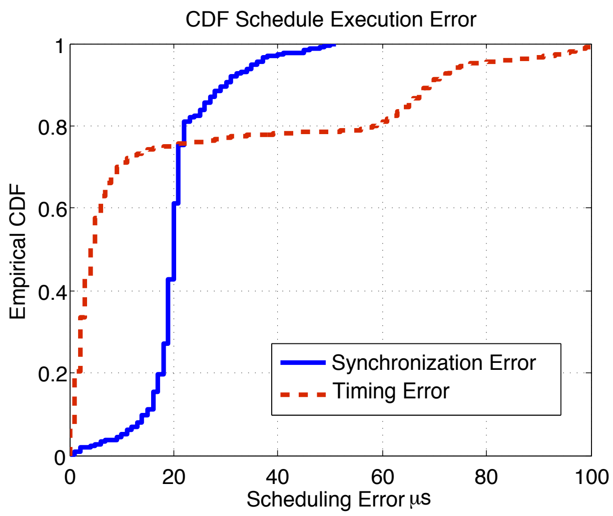


Figure 8: AP-to-controller clock synchronization error and transmission deviance from assigned schedule, relative to local clock. AP and controller were separated by approximately 20 m of CAT-5 cable, 1 switch, and 1 hub. Measurements in μs. Margin of error $\leq$ 5 μs, attributable to 802.11 TSF inaccuracy.

Time Synchronization and Stagger

To enforce transmission staggers on the order of preamble durations (tens of μs), we need equally precise time synchronization between AP and controller. The 802.11 TSF clock is used for synchronizing all stations in a BSS. To synchronize APs with the Shuffle controller, we insert 802.11 TSF timestamps into ethernet packets by modifying the Sky2 ethernet driver. These time-stamped control packets are exchanged bidirectionally between the controller and APs. When a controller receives a TSF timestamped packet from the AP, it computes the offset between the timestamp and its local TSF clock. This offset includes wire propagation delays, ethernet switching latencies, processing time, and the clock difference between the controller and the AP. The same offset is also computed at the AP, and exchanged between the two parties. The AP averages the two offsets and deduces an estimate of the actual instantaneous difference between the controller's clock and its own TSF. Propagation delays and processing latencies in the ethernet driver are reasonably symmetric, hence, the clock difference estimation can be accurate on the order of microseconds. The clock difference is cached and exposed to Click through a `sysctl` interface.

Figure 8 presents the empirical CDF of the AP/controller synchronization error achieved by our implementation. We estimate this error by spatially collocating the AP and the controller, which then get synchronized by the same TSF clock on their wireless interfaces. The TSF clock now acts as the reference clock, allowing us to quantify our synchronization precision over the AP/controller wired connection. We consistently achieved a median error of 20 μs. Since the Atheros TSF implementation is accurate to $\pm$ 5 μs (verified in a separate experiment by comparing packet reception times at multiple receivers from a single transmitter), we believe that our total margin of error is within 25 μs.

Upon receiving a packet from the controller, APs busy-wait on their TSF clocks to transmit the packet at the scheduled instant. We measure the inconsistency between the scheduled time of transmission, t_s, and the actual time it was transmitted, t_a. This measurement was performed by assigning an AP to transmit packets with a precise spacing of 20 ms. At a collocated receiver, we measure error as 20 ms minus the observed inter-packet arrival times. The dashed line in Figure 8 plots the CDF of this deviation. Mean timing error is 4 μs.

Coordination and Dispatching

For every wireless-bound packet, the AP places the packet on a per-client queue and sends a notification packet to the controller. Once MIM-scheduling selects the packets to schedule, the controller broadcasts the schedule in the form of <*AP, client, start time*> tuples. Observe that the controller does not specify which exact packets must be transmitted – it only specifies the links that must be activated. Upon receiving a schedule, the AP dequeues a packet to the specified client and passes it to MadWiFi, along with the exact local TSF clock at which transmission must start. The MadWiFi driver busy waits on the TSF clock, and hence, can transmit the packet at the precise time. Transmissions continue until the controller schedules an upload window, at which point the clients respond with batch acknowledgments. The batch ACK contains a bit vector that marks the failed transmissions in the preceding upload window. The AP places the failed packets on a highest-priority retransmission queue, and informs the controller. The highest-priority queue ensures that the AP will not transmit any new packet to the same client before all retransmissions have been satisfied. In the subsequent download window, the controller accounts for the failed packets, and generates new ordered schedules. The process continues.

6. EVALUATION

Our testbed evaluation aims to demonstrate the feasibility of Shuffle on commodity hardware, and to characterize the performance improvements with MIM-aware scheduling. Comparison with IEEE 802.11 MAC confirms gains from centralized scheduling. To highlight the gains attributable to order-awareness, independent of centralized scheduling, our evaluation focuses on comparison with a capture-aware TDMA (running on MIM hardware, but without imposed ordering). We begin our analysis with results using a fixed data rate, showing how correct ordering improves the delivery probability on weaker links. Next, we compare Shuffle and TDMA operating with full 802.11g rate control. We summarize our main findings below.

- We begin with 4 simple 2-AP topologies. Shuffle outperforms 802.11 by 40% and TDMA by 20% (Fig. 9). Shuffle's Jain Fairness Index is close to 1, while 802.11 and TDMA are around 0.95 and 0.93, respectively.

- Incorrect order of transmissions considerably degrades performance. In 2-AP topologies, the difference between the correct and wrong order is almost 30% (Fig. 9).

- The importance of order is more pronounced in 3 AP topologies (Fig. 10 and 11). More concurrency opportunities offer higher gains with Shuffle – up to 100% over 802.11, and 20% over TDMA. Fairness improves too. Results from 10-link topologies also attain around 70% gain over 802.11, and 20% over TDMA (Fig. 12).

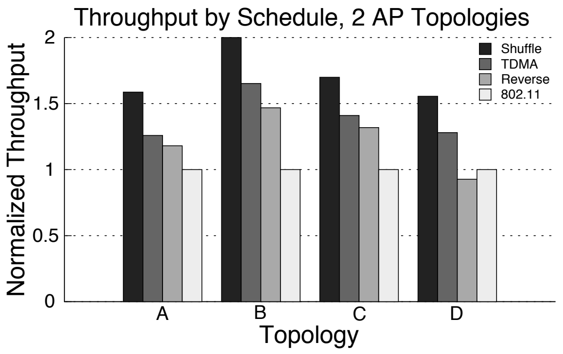


Figure 9: Concurrency gains with only two links.

- Fully-functional Shuffle including 802.11g rate control show 29% gains over TDMA in a favorable topology (Fig. 13) and yields 17% when different client positions are systematically tested (Fig. 14, 15).

Throughput with 2 Access Points

To understand the primitives of MIM-aware scheduling, we begin with topologies of 2 APs, each associated with a single client. We characterized the interference relationships, as coordinated by the Shuffle controller, finding the proper stagger order for maximal concurrency. To understand the ramifications of incorrect ordering, we forced the controller to schedule transmissions in correct and incorrect orders. For fairness towards 802.11, we disabled RTS/CTS protection and ensured that the topologies under test did not include hidden terminals. Figure 9 presents our results.

Evident from the graphs, MIM-aware transmission reordering consistently yields higher throughput than both 802.11 and order-unaware TDMA scheduling. When ordered correctly, strong links allow weaker links to start first, and then extract their own signal of interest from the channel (recall the notion of re-locking). In the absence of explicit ordering in TDMA, concurrent packets may naturally achieve "good" and "bad" link orderings due to clock synchronization error. For some packets, the "right" AP will transmit first, and for others, it will start too late and fail. Thus, for any pair of links, we expect a TDMA schedule to result in the correct order (and thereby gain) approximately half of the time. Our results support this intuition. Fairness, computed as Jain's Fairness Index, also improves. We discuss this more later.

Throughput with 3 Access Points

The notion of ordering becomes more complex with 3 clients, each associated with a distinct AP. Figures 10 and 11 show the throughput and fairness comparisons. Since more concurrent links are feasible, Shuffle outperforms 802.11 and TDMA by larger margins. However, of greater interest is the sensitivity of performance to the different transmission orders. The variation in throughput between different orders is evidently large, indicating that gains from MIM reordering may not be extracted blindly. Use of the incorrect order lowers throughput below that of a TDMA schedule. Interestingly, even suboptimal orderings provide gains over 802.11. This is an attribute of overly conservative carrier sense mechanisms in 802.11, leading to exposed terminal problems [12]. Shuffle overcomes these problems through centralized scheduling and overlapping transmissions.

Fairness

Centralized MIM-aware scheduling does not degrade fairness among clients (recall that our scheduling algorithms account for fairness and starvation). Shuffle improves fairness over 802.11 and TDMA. In Figure 11, we characterize these gains using Jain's Fairness Index. The 802.11 backoff mechanism preferentially treats links which experience fewer losses. Thus, 802.11 exacerbates the already disproportionate bandwidth allocation to stronger links. The Shuffle controller attempts to ensure that sufficient transmission opportunities are extended to all links, reducing this effect.

Performance on Larger Topologies

We tested Shuffle on larger topologies with 3 APs each connected to up to 4 clients. One of the link topologies with 10 links is illustrated in Figure 12(a). For this experiment, equal traffic is generated for each client. Based on their interference relationships, not all scheduled batches can consist of 3 concurrent links. As a result, Shuffle sometimes schedules batches of 2 concurrent links (especially in view of fairness). Figure 12(b) compares the throughput between Shuffle, TDMA, and 802.11. Performance improvements in this, and other topologies are reasonable and consistent. Fairness among the links proves to be high, as illustrated in Figure 12.

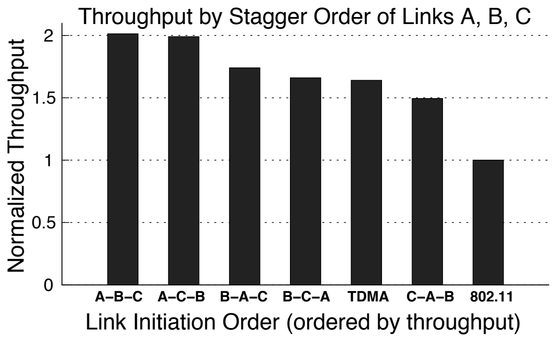


Figure 10: Multiple Shuffle orders provide higher throughput than TDMA and 802.11.

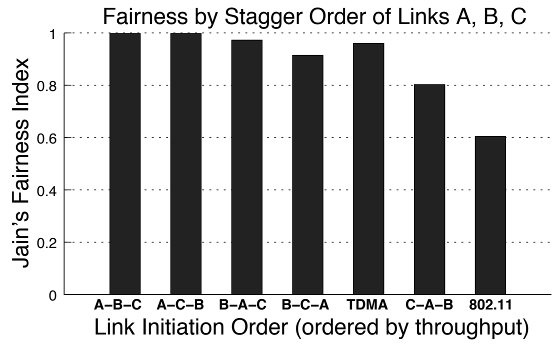


Figure 11: Shuffle scheduling improves fairness.

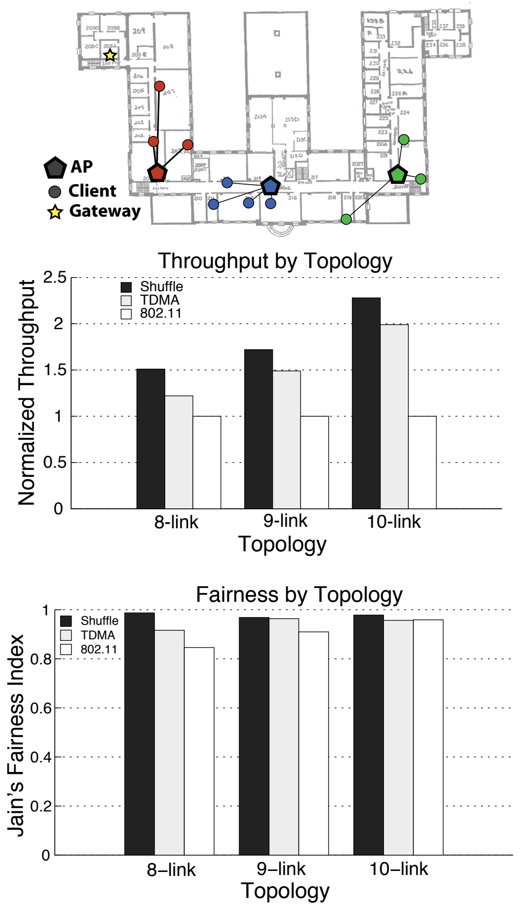


Figure 12: (a) Example 10 link topology in our building (b) Throughput and fairness on entire Shuffle testbed.

Complete Shuffle with Rate Control

We evaluate the benefits of MIM-aware ordering with rate control enabled. Our rate control mechanism is similar in principle to SampleRate and utilizes all 802.11g rates. Rate control and MIM-aware ordering decisions are made holistically by the controller, as part of the scheduling heuristic.

With aggressive rate control mechanisms, channel fluctuations can cause dramatic changes in throughput between tests as the ideal rate changes with time. To ensure that Shuffle gains relative to TDMA are attributable to ordering and not testing artifacts, we measure throughput for Shuffle and TDMA *simultaneously*. Our controller alternates between scheduling batches of concurrent packets with and without ordering to effectively time-share the channel. In this way, our comparison is unbiased if channel behavior is coherent for longer than a single packet duration. By maintaining separate interference maps and rate control state for both Shuffle and TDMA, Shuffle's order-aware conflict learning is not impacted by failures due to TDMA naiveté. To compare throughput, the controller records the number of packets scheduled, the number of failed transmissions, and the amount of time scheduled on the channel, independently for Shuffle and TDMA. Given that Shuffle and TDMA run identical algorithms for centralized scheduling and rate control, with the one exception of imposed ordering through stagger, we believe this to be a highly fair comparison. Since 802.11 is not compatible with centralized scheduling, and thus with this testing methodology, results for 802.11 are not reported in this section.

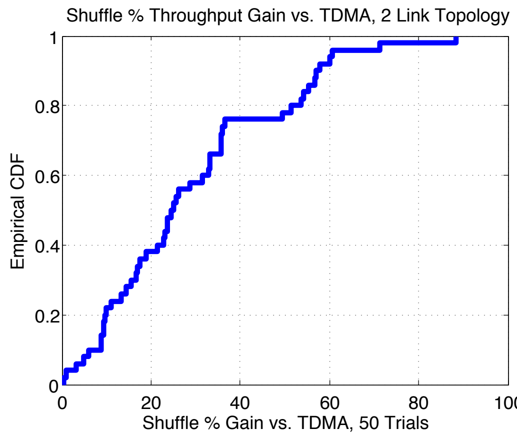


Figure 13: Throughput for Shuffle versus TDMA using 802.11g with 6-54 Mbps rate control enabled.

In Figure 13, we present results from one topology consisting of two mutually-interfering links, similar to that presented in Figure 1. One link strong and relatively unaffected by the interferer. The other link is far more susceptible. With Shuffle, the weaker link successfully maintains a higher data rate than it can under TDMA. We plot a CDF of our results over 50 trials (the system starting from a ground state for each trial) to show that the Shuffle conflict interference mechanism can reliably deduce the proper ordering. Mean gain is 29%.

Although the potential for gains with MIM ordering is topology dependent, it is not highly sensitive. As depicted in Figure 14, we deployed a two AP topology. One AP was positioned to serve a classroom and another just outside, a strong interference source. We collocated a receiver with the outside AP, creating a strong link. By systematically moving the other client between each of the 54 seating positions, we create a diverse set of channel conditions. Figure 15 presents a CDF of our results. Mean throughput is 31.6 and 27.0 Mbps for Shuffle and TDMA, respectively (a Shuffle gain of 17.1%).

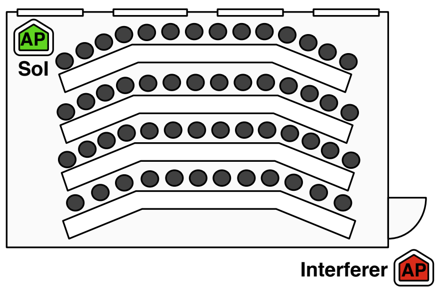


Figure 14: A classroom environment with 54 seats. Leaving the AP and one client fixed, we tested with a client placed on the desk in front of each chair.

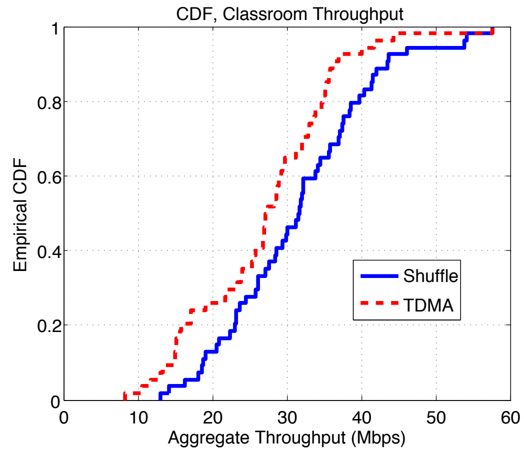


Figure 15: CDF of throughput for classroom test.

Simulation

We performed QualNet simulations to evaluate performance in larger topologies – we report them briefly in the interest of space. We recorded actual AP placements across multiple buildings, and used them for simulation. Each AP was associated with 6 clients approximately placed in the rooms of the building. Figure 16(a) shows 4 topologies – Shuffle consistently outperforms *NoMIM* (i.e., TDMA) under all fading models. Figure 16(b) shows the results of synthetically increasing the AP density. The performance difference is consistent.

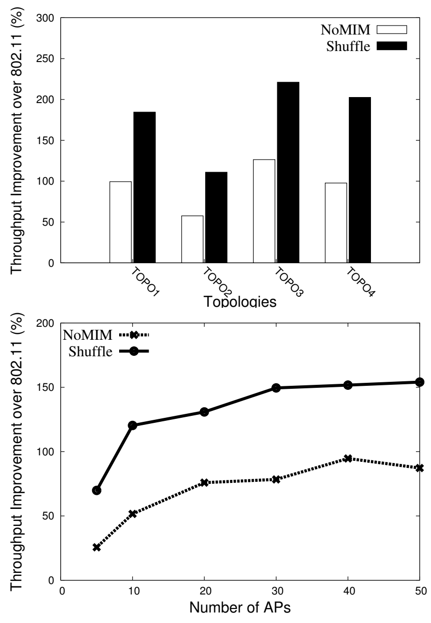


Figure 16: (a) Throughput for real-life AP placements. (b) Higher AP density increases concurrency.

7. LIMITATIONS AND DISCUSSION

We discuss some limitations with Shuffle, and identify avenues of further work.

External Network Interference. We assume that all WiFi devices are associated to the same enterprise network. Put differently, no *other* WiFi transmission occurs that is not accounted for by the central controller. In reality, electronic devices such as microwaves may interfere in the 2.4 GHz band. Wireless devices from "neighboring" networks may interfere at the periphery of a Shuffle deployment. Shuffle's loss recovery mechanism will be able to cope with sporadic interference. However, if the losses are frequent, carrier sensing may need to be selectively enabled for the peripheral APs, limiting the controller's ability to schedule for those clients.

Latency. Shuffle introduces some end-to-end delivery latency. When a packet is received at an AP, it cannot be forwarded to the intended client until the AP notifies the controller and receives a scheduled slot for the packet. Assuming no queuing at the AP or client, the added latency is only due to propagation, switching, and processing of two control messages. As a design alternative, this latency may be eliminated if the controller is collocated with the network gateway, so that schedules may be forwarded to APs in tandem with the outbound packet. While this provides no improvement for retransmissions, recall that retransmissions assume higher priority than new packets to the same client. This may make the retransmission delay tolerable.

Client Mobility. As a client moves, interference relationships may change dramatically. While Shuffle's concurrent link selection, rate control, and transmission ordering mechanism do adapt to changes in channel conditions, we have not yet characterized convergence time for continuous-mobility scenarios.

Transport Layer Interactions. We have not yet characterized TCP interaction behavior. A potential point of concern is division of time into upload and download periods, possibly impacting TCP round-trip time estimation and ACK timeouts. However, we believe that upload periods may be scheduled frequently enough (every few download packets) to limit this effect.

Compatibility. Shuffle is not immediately compatible with existing deployments. Clients must be protocol compliant so as to remain silent during download periods and provide ACKs during upload windows.

Small-scale Testbed. We tested our Shuffle implementation on topologies consisting of up to three concurrent links. Larger topologies is a part of our future work.

8. RELATED WORK

Capture and MIM. Theoretical models have been proposed to explain physical layer capture [13]. The first empirical evidence of capture was presented in [1]. The recent study in [3] quantifies SINR threshold requirements for 802.11a networks under different packet arrival timings. Capture awareness has been used for collision resolution in [14]. BER models for capture were proposed in [15].

Spatial Reuse. Schemes like [16, 17] make use of power control and carrier sense tuning to achieve improved spatial reuse. Prior work has considered RTS/CTS variants to schedule non-conflicting links [18]. However, most existing deployments do not use RTS and CTS [12], even with RTS/CTS do not exploit concurrency well. In CMAP [12], the authors propose a distributed scheme which makes use of partial packet decoding to determine if a concurrent transmission is possible. This distributed approach makes use of the delivery ratios of concurrent transmissions to determine whether they can be successful. CMAP can benefit from MIM-capable hardware, but is not MIM-aware. In contrast, our work explicitly orders transmissions to take advantage of MIM. In [19], the authors propose MIM-aware transmission reordering.

Enterprise Wireless LANs and Scheduling. Enterprise wireless LAN architecture is increasingly becoming popular to improve throughput, monitoring and management. SMARTA [9] utilizes a centralized server to build a conflict graph and fine tune the AP's transmit power and channel selection mechanisms. Several scheduling mechanisms for single and multihop radio networks like [4] were proposed and in the context of EWLANs. Our controller-AP interaction is similar to the one proposed in a recent work [20]. The speculative scheduling solution in [6] proposed a conflict graph based centralized scheduling mechanism to improve spatial reuse.

Characterizing and Measuring Interference. In [21, 22] the authors analyze the effects of combined interference. A capture-aware linear order algorithm for estimating link state interference in multihop wireless networks was presented in [23]. In [24] the authors show how signal strength conditions vary transiently in real networks and they quantify the affects of received signal strength on delivery probability. In this work we use a decision making scheme based on concurrent delivery ratios similar to [12].

9. CONCLUSION

Message in Message (MIM) in modern wireless cards allows a receiver to disengage from an ongoing reception, and engage onto a new stronger signal. The rewards from this physical layer capability cannot be fully realized unless link layer protocols are explicitly designed with MIM-awareness. Specifically, links that conventionally conflict with each other may be made concurrent if they are initiated in a specific order. This paper presents Shuffle, a system that reorders transmissions to improve spatial reuse. Theoretical analysis shows that the optimal improvements with MIM can be significant. A functional testbed validates that MIM-awareness is practical, while results of experimental evaluation confirm consistent performance improvements.

10. ACKNOWLEDGEMENTS

We sincerely thank our anonymous reviewers for their invaluable feedback. We also thank Victor J. Orlikowski (Duke OIT) for help during the early stages of the testbed installation. Finally, we are grateful to our funding sources NSF (grants CNS-0747206, CNS-0448272, CNS-0551650) and Cisco.

11. REFERENCES

[1] A. Kochut et al., "Sniffing out the correct physical layer capture model in 802.11b," in *ICNP*, 2004.

[2] T. Nadeem and L. Ji, "Location-aware ieee 802.11 for spatial reuse enhancement," *IEEE Trans. Mobile Computing*, vol. 6, no. 10, Oct. 2007.

[3] J. Lee et al., "An experimental study on the capture effect in 802.11a networks," in *WinTECH*, 2007.

[4] S. Ramanathan and E. L. Lloyd, "Scheduling algorithms for multihop radio networks," *IEEE/ACM Trans. on Networking.*, vol. 1, no. 2, 1993.

[5] P. Bahl et al., "DAIR: A framework for managing enterprise wireless networks using desktop infrastructure," in *HotNets IV*, 2005.

[6] N. Ahmed et al., "Interference mitigation in enterprise WLANs through speculative scheduling," in *Mobicom*, 2007.

[7] Meru Networks, "Revolutionizing wireless lan deployment economics with the meru networks radio switch," in *White Paper*, 2005.

[8] R. Murty et al., "Designing High Performance Enterprise Wi-Fi Networks," *NSDI*, 2008.

[9] N. Ahmed and S. Keshav, "Smarta: A self-managing architecture for thin access points," in *CoNEXT*, December 2006.

[10] Y.C. Cheng et al., "Jigsaw: solving the puzzle of enterprise 802.11 analysis," in *ACM SIGCOMM*, 2006.

[11] J.C. Bicket, "Bit-rate selection in wireless networks," M.S. thesis, Massachusetts Institute of Technology, 2005.

[12] Mythili Vutukuru, Kyle Jamieson, and Hari Balakrishnan, "Harnessing exposed terminals in wireless networks," in *NSDI*, 2008.

[13] M. Durvy, O. Dousse, and P. Thiran, "Modeling the 802.11 protocol under different capture and sensing capabilities," in *Proc. IEEE Infocom*, 2007.

[14] K. Whitehouse et al, "Exploiting the capture effect for collision detection and recovery," in *Emnets*, May 2005.

[15] H. Chang et al., "A general model and analysis of physical layer capture in 802.11 networks," in *Proc. IEEE Infocom*, 2006.

[16] T-S Kim and et al., "Improving spatial reuse through tuning transmit power, carrier sense threshold, and data rate in multihop wireless networks," in *Proc. ACM Mobicom*, 2006.

[17] K. Jamieson et al, "Understanding the real world performance of carrier sense," in *ACM SIGCOMM E-WIND Workshop*, 2005.

[18] K. Mittal and E. M. Belding, "RTSS/CTSS: Mitigation of exposed terminals in static 802.11-based mesh networks," in *WiMesh*, 2006.

[19] N. Santhapuri et. al, "Message in message (mim): A case for reordering transmissions in wireless networks," in *HotNets VII*, 2008.

[20] R. Murty, A. Wolman, J. Padhye, and M. Welsh, "An architecture for extensible wireless lans," in *HotNets VII*, 2008.

[21] D. N. C. Tse and S. Hanley, "Linear Multiuser Receivers: Effective Interference, Effective Bandwidth and User Capacity," *IEEE Trans. Inform. Theory*, vol. 45, pp. 641–675, Mar. 1999.

[22] J. Padhye et al., "Estimation of link interference in static multi-hop wireless networks," in *IMC*, 2005.

[23] J. Lee et al., "RSS-based carrier sensing and interference estimation in 802.11 wireless networks," in *SECON*, 2007.

[24] C. Reis et. al, "Measurement-based models of delivery and interference in static wireless networks," in *Proc. ACM Sigcomm*, Sept. 2006.