Order Matters: Transmission Reordering in Wireless Networks

Justin Manweiler, Naveen Santhapuri, Souvik Sen, Romit Roy Choudhury, Srihari Nelakuditi, and Kamesh Munagala

***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 for enterprise wireless LANs to achieve it. Testbed and simulation results confirm the performance gains of the proposed system.**

***Index Terms*— Wireless LAN, wireless networks.**

I. 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 reengaging 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: An 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 [1], [2]. If, on the other hand, the SoI arrives before the interference and satisfies the required SINR, the signal can be successfully decoded. Fig. 1 shows the two cases.

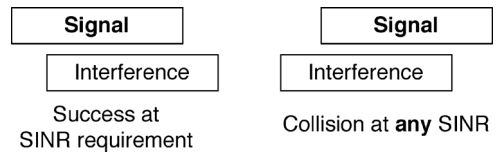


Fig. 1. Most simulators assume that the order of SoI and interference determines precisely whether an SoI can be received, given sufficient SINR.

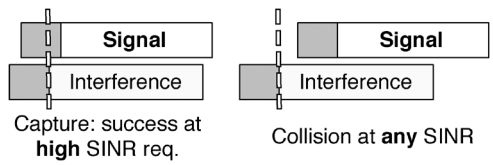


Fig. 2. Physical layer capture enables a later-arriving SoI to be received in principle, but only with overlap during the 802.11 preamble.

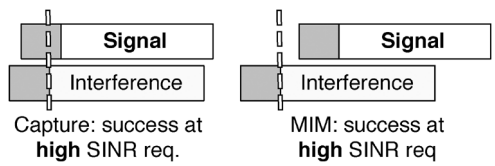


Fig. 3. With MIM, a later-arriving SoI with sufficient SINR can be received, even if there is no overlap of 802.11 preambles.

Physical layer capture was later understood through the systematic work in [3] and [4]. 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. Fig. 2 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.

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 [5]. Of course, the required signal-to-interference-plus-noise ratio (SINR) is higher for relocking onto the new signal. Conversely, if the SoI arrives earlier than the interference, the reception with MIM-capable hardware is same as traditional reception. Fig. 3 illustrates the MIM advantage.

Manuscript received December 29, 2010; accepted June 08, 2011; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor K. Papagiannaki. This work was supported by the National Science Foundation under Grants CNS-0747206, CNS-0448272, and CNS-0551650. This is a revised version of a paper presented at the 15th Annual ACM International Conference on Mobile Computing and Networking (MobiCom), Beijing, China, September 20–25, 2009.

J. Manweiler, N. Santhapuri, S. Sen, R. Roy Choudhury, and K. Munagala are with the Departments of Computer Science and Electrical and Computer Engineering, Duke University, Durham, NC 27708 USA (e-mail: romit@ee.duke.edu).

S. Nelakuditi is with the Department of Computer Science and Engineering, University of South Carolina, Columbia, SC 29208 USA.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNET.2011.2164264

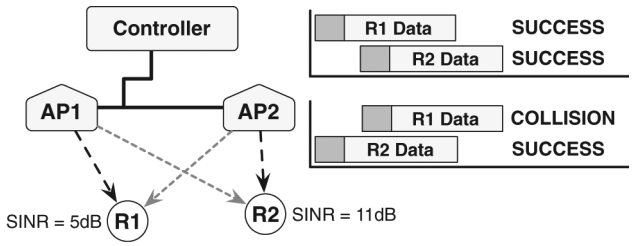


Fig. 4. AP1 → R1 must start before AP2 → R2 to ensure concurrency. If AP2 starts first, R1 locks onto AP2 and cannot relock onto AP1 later.

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 it 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 relocks onto 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.

A. Link Layer Opportunity

Consider the example in Fig. 4. For R1, AP1 is the transmitter, while AP2 is the interferer (and 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 relock 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 relock 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.

In a larger network, choosing the appropriate set of links from within a collision domain, and determining the order of optimal transmission, is a nontrivial 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. To address this problem, we propose *Shuffle*, an MIM-aware link-layer solution that reorders transmissions to extract performance gains. Our main contributions in this paper are the following.

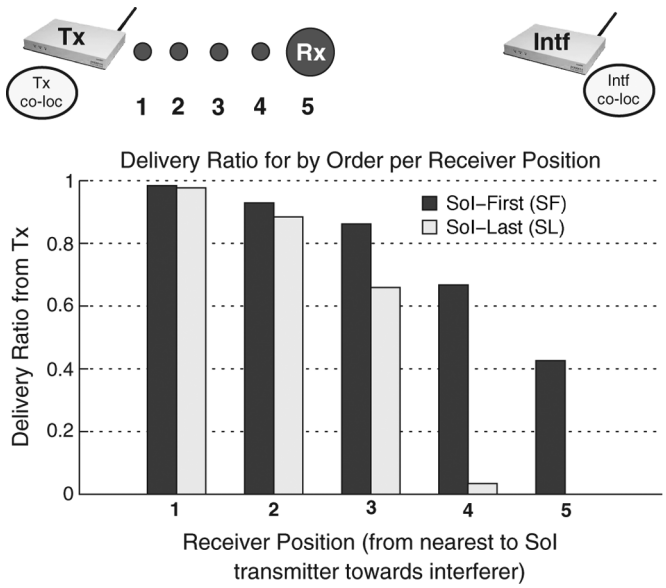


Fig. 5. Testbed confirms MIM capability. Rx receives from Tx (at five positions) in the presence of interference (Intf).

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 the optimal performance possible with MIM*. We show that MIM-aware transmission scheduling is NP-hard, and derive upper bounds on its performance using integer programming. CPLEX results show that the optimal gain from MIM is substantial, hence worth investigating.
3) *Design of an MIM-aware transmission scheduling 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. Additional QualNet simulations show the scalability of Shuffle system to larger networks.

II. Verifying MIM

We validate the existence of MIM capabilities in commodity hardware using a testbed of Soekris [6] embedded PCs, equipped with Atheros 5213 chipsets running the MADWiFi driver [7]. The experiment consists of two transmitters with a single receiver placed at various points in between (Fig. 5). This subjects the receiver to varying SINRs. To ensure continuous transmissions from the transmitters, we modify the MADWiFi driver to disable carrier sensing, backoff, and the interframe 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*.

Fig. 5 shows delivery ratios for different orders 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 1–3), 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 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 capability on commercial hardware and additionally confirms that enforcing the correct order among nearby transmissions can be beneficial.

III. MIM: Optimality Analysis

A natural question to ask is *how much throughput 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. Toward this end, 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 maximize the network throughput. Assume that 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* [8]. 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-aware link scheduling is NP-hard. ∎

A. Optimal Schedule With Integer Program

To quantify the performance gains from MIM, we model wireless networks with MIM-capable and MIM-incapable receivers 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 SF SINR requirement is 4 dB, while the SL requirement is 10 dB [5]. 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 IV-B, we consider MIM scheduling with rate control.

TABLE I
Integer Programming Parameters and Variables

Parameter	Value	Meaning
N		Set of all nodes.
$\mathcal{L}$		Set of all links l_{ab} s.t. RANGE$(a \rightarrow b)$
S_l		Signal strength on link l.
$I(m \rightarrow l)$		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 $(a \rightarrow b)$ in use.
	0	Otherwise.
y_{lm}	1	Link l starts before link m.
	0	Otherwise.

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

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 $I(m_{cd} \rightarrow l_{ab})$ denote interference received on link l (by node b) due to a concurrent transmission on link m (from node c). Table I summarizes the parameters and variables. Under an assumption of additive multiple interference and nonfading channels, the maximum link concurrency of a given wireless network under MIM-aware MAC can be found using the following integer programming formulation:

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

$$\textbf{Subject To}:$$

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

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

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

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

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

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

Theorem 2: Any 0/1 solution to the above integer program satisfies the following.

1) $x_l = 1$ encodes 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 integer program 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; 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 link 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. Furthermore, the left-hand side of the constraint is nonzero only if l itself is active. A similar reasoning shows the validity of constraint (5). ■

B. Results

We used CPLEX [9] to solve many instantiations of the integer program. In Fig. 6, we present results for topologies of grid-aligned access points and randomly placed clients. All clients associate to 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.

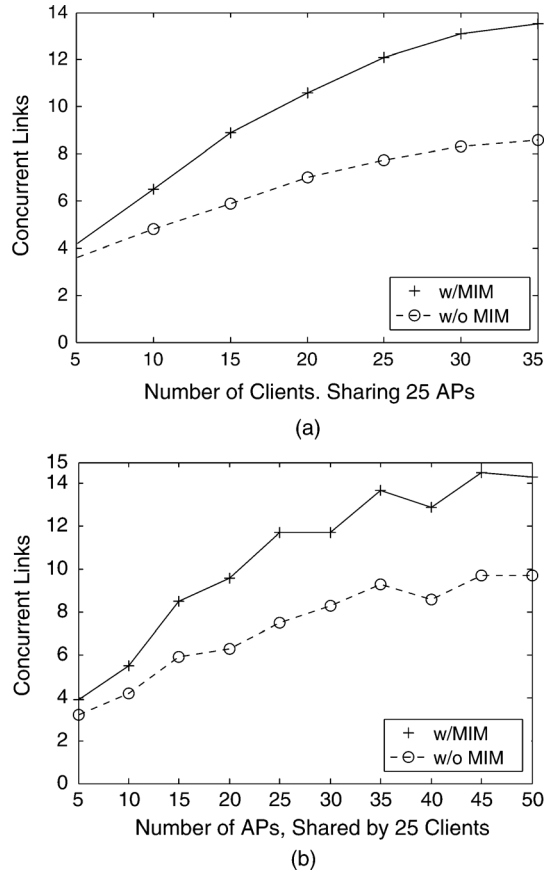


Fig. 6. MIM can provide large concurrency gains. These graphs show the number of links that can meet SINR requirements with and without MIM enabled. Gains improve with increasing network density. (a) Varying number of clients. (b) Varying number of APs.

IV. 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 [10], [11]. In EWLANs, multiple APs are connected to a central controller through a high-speed wired backbone (Figs. 4 and 10). The controller coordinates the operations of APs. The APs follow the controller's instructions for transmitting packets to their clients.

The rationale for targeting EWLAN architectures is twofold. 1) EWLANs are becoming popular in single-administrator environments [11]–[15]. 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 to solving it should perhaps start from a more tractable system. EWLAN presents a semicentralized platform, amenable to experimentation. Exploiting MIM capability on this architecture is itself a rich, unexplored research area that could lay the foundation for extending MIM-aware schemes to decentralized systems.

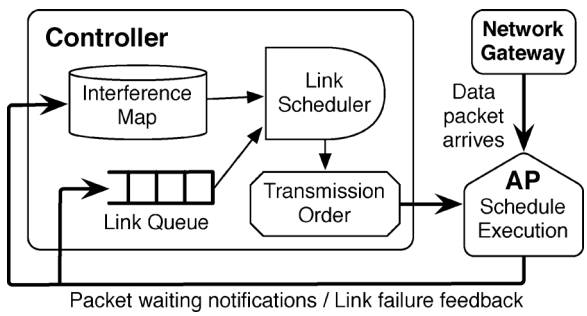


Fig. 7. Flow of operations in the Shuffle system. Data packets arrive from the network gateway and are enqueued at an AP. The AP notifies the controller of the waiting outbound packet. The controller inserts the corresponding AP–client pair into a network-wide link queue and eventually schedules this 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.

A. Protocol Design

We first sketch the three main operations of Shuffle. Fig. 7 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 transmissions, the controller notifies the relevant APs when these transmissions should occur. The APs maintain precise time synchronization with the controller so that expected transmission orderings may be accurately executed.

1) Conflict Diagnosis: Scheduling algorithms require the 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, packet losses may happen when the interference map becomes inconsistent with the time-varying network conditions. Shuffle copes with it 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 link 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. Fig. 8 shows the data structures maintained at the controller. When the delivery ratio between link 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.

Reviving Concurrency: Link–AP pairs that are currently 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.

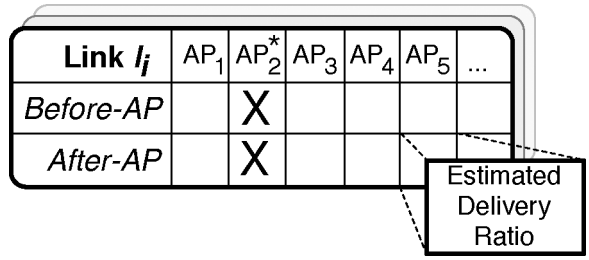


Fig. 8. Per-link data structure maintained at the controller for scheduling transmissions. AP_2 is the transmitter for link l_i.

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 IV-B, we discuss how the Shuffle controller incorporates active bit 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 $\mathrm{AP}(l_i)$ are unique; all $\mathrm{Client}(l_i)$ 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 would be to generate all $B \subset Q$ such that B is feasible. Assuming a fixed bit rate, the highest throughput schedule would be the feasible subset of Q with maximum cardinality. While this approach may be plausible with small queues, it is imperative that we develop more computationally efficient heuristics for wider

applicability. To this end, we present two suboptimal but practical (greedy) heuristics.

Greedy Algorithm: A simple greedy algorithm would be to consider the packets in the first-in–first-out (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** $\mathrm{noConflict}(p, j, B)$ **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 pairwise 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 to 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 packet 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 the Least-Conflict Greedy algorithm is also $O(n^2)$.

Algorithm 2: Least-Conflict $\mathrm{Greedy}(Q)$

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

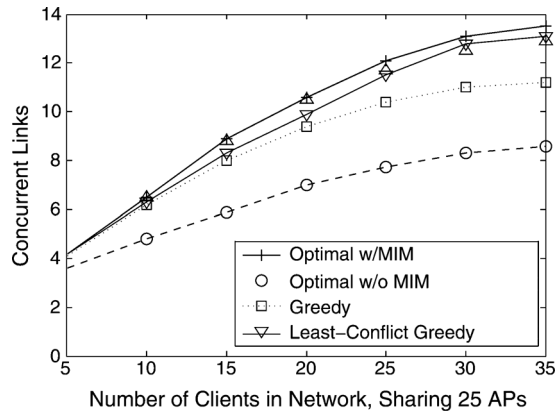


Fig. 9. Heuristics for MIM-aware scheduling.

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. Overall, this method is likely to achieve better concurrency compared to the simple greedy scheme at the expense of some unfairness.

Comparison to Optimal: Fig. 9 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 consistent 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 V), APs are able to execute the staggered transmission schedule, illustrated in Fig. 10. 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 (UTWs), reserved for ACKs and other client upload traffic. At the expiration of a UTW, the AP can deduce reception failures for the

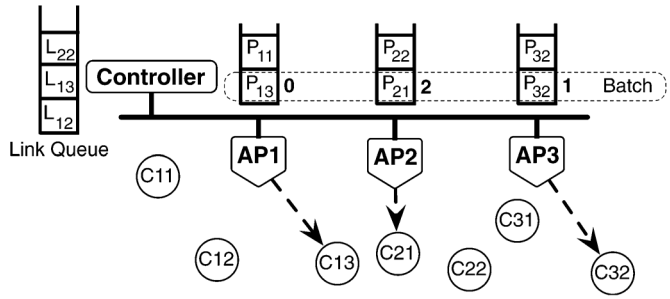


Fig. 10. Illustration of a scheduled batch of packets with the staggered transmission times. AP1 starts first, followed by AP3, then AP2.

packets transmitted in the preceding download period. For each failed 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.

B. 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 transmission bit 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 [16] 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 $\mathrm{RateControl}(l, i)$ 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 bit rate for their transmissions, 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 [16].

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 UTW. During each UTW, clients contend for channel access using the traditional 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 V), such division is practical.

Controller Placement: A number of placement options exist for installing the controller into the network. It 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 plug the controller directly onto the wired network (as an independent device or perhaps as a module in one of the APs). Because of our lightweight scheduling heuristics, we find relatively low CPU utilization for the controller process ($\approx$20%). Decoupling the controller from the gateway may provide higher flexibility and easier maintenance. If necessary, the controller can be reprogrammed with a better scheduling protocol and plugged back into the network. Advantages in time synchronization and retransmissions also arise as a by-product. However, the APs may need to be "thicker" than when the controller regulates the flow of packets.

V. Shuffle: Implementation

Testbed Platform: We evaluated our fully functional Shuffle implementation on a testbed consisting of laptops running Linux kernel ver. 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. One of the laptops served as the controller, while others as APs and clients. 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-aware transmission scheduling, and loss recovery) are implemented through element extensions to the *Click Modular Router* [17]. 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 1500-B 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

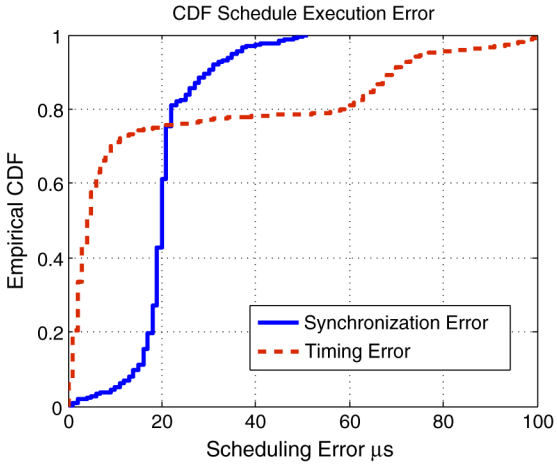


Fig. 11. AP-to-controller clock synchronization error and transmission deviance from the assigned schedule, relative to the local clock. AP and controller separated by approximately 20 m of CAT-5 cable, one switch, and one hub. Margin of error ≤ 5 μs, attributable to 802.11 TSF inaccuracy.

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.

Time Synchronization and Stagger: To enforce transmission staggers on the order of preamble durations (tens of μs), we need equally precise time synchronization between the 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 time-stamped 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.

Fig. 11 presents the empirical cumulative distribution function (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 synchronization error of 20 μs. Since the Atheros chipset TSF implementation is accurate to ± 5 μs (verified in a separate experiment by comparing the packet reception times at multiple TSF-synchronized 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 interpacket arrival times. The dashed line in Fig. 11 plots the CDF of this deviation. The mean timing error is around 4 μs.

Coordination and Dispatching: For every wireless-bound packet, the AP places the packet on a per-client queue and sends a notification 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 ACKs. 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 reducing out-of-order delivery. In the subsequent download window, the controller accounts for the failed packets and generates new ordered schedules. The process continues.

VI. 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 to 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 to a capture-aware TDMA (running on MIM hardware, but without imposed ordering). We begin our analysis with results using a fixed bit rate, showing how correct ordering improves delivery probability on weaker links. Next, we compare Shuffle and TDMA operating with full 802.11g rate control. We summarize our main findings as follows.

- We begin with four simple two-AP topologies. Shuffle outperforms 802.11 by about 40% and TDMA by 20% (Fig. 12). Shuffle's Jain Fairness Index is close to 1, while that for 802.11 and TDMA are around 0.95 and 0.93, respectively.
- Incorrect order of transmissions considerably degrades performance. In two-AP topologies, the difference in throughput between the correct and wrong order is almost 30% (Fig. 12).

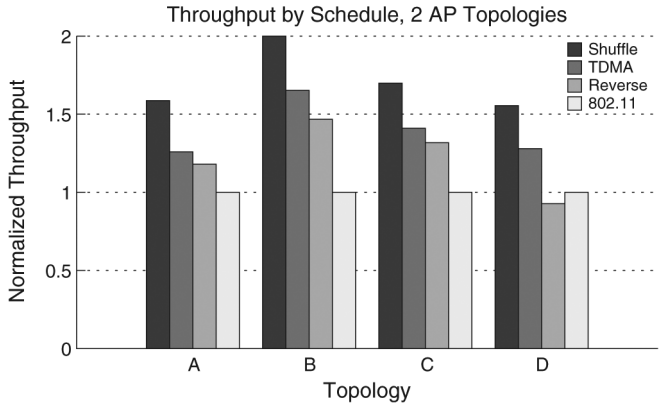


Fig. 12. Concurrency gains with only two links.

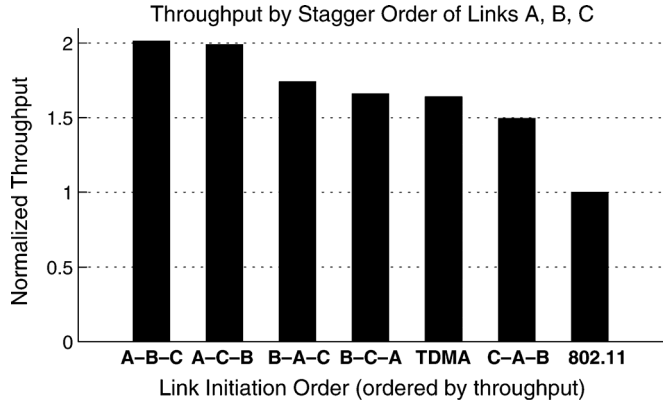


Fig. 13. Multiple Shuffle orders provide higher throughput than both TDMA and 802.11.

- The importance of order is more pronounced in three-AP topologies (Figs. 13 and 14). More concurrency opportunities offer higher gains with Shuffle—up to 100% over 802.11, and 20% over TDMA. Fairness improves as well. Results from 10-link topologies also attain around 70% gain over 802.11, and 20% over TDMA (Fig. 15).
- Fully functional Shuffle including 802.11g bit-rate control shows 29% gains over TDMA throughput in a favorable topology (Fig. 16) and yields 17% gain when different client positions are systematically tested (Figs. 17 and 18).

Throughput With Two Access Points: To understand the primitives of MIM-aware scheduling, we begin with topologies of two 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 toward 802.11, we disabled RTS/CTS and ensured that the topologies under test did not include hidden terminals. Fig. 12 presents the 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 relocking). 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 Three Access Points: The notion of ordering becomes more complex with three clients, each associated with a distinct AP. Fig. 13 shows the throughput comparison. 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 [18]. Shuffle overcomes these problems through centralized scheduling and overlapping transmissions.

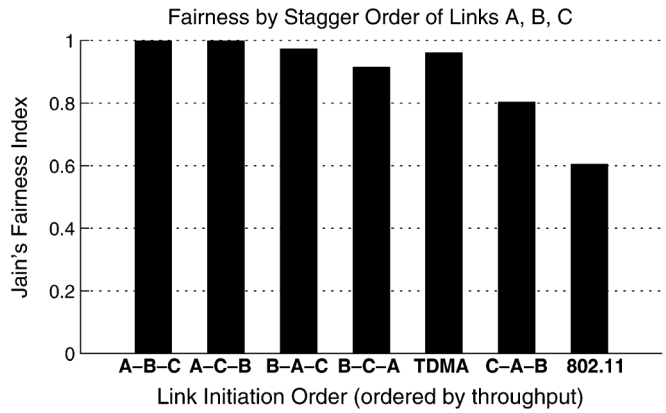


Fig. 14. Shuffle scheduling improves fairness.

Fairness: 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 Fig. 14, we characterize these gains using Jain's Fairness Index. The 802.11 backoff mechanism preferentially treats links that 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 three APs each connected to up to four clients. One of the large topologies with 10 links is illustrated in Fig. 15(a). For this experiment, equal traffic is generated for each client. Based on their interference relationships, not all scheduled batches can consist of three concurrent links. As a result, Shuffle sometimes schedules batches of two concurrent links (especially in view of fairness). Fig. 15(b) compares the throughput between Shuffle, TDMA, and 802.11. Performance

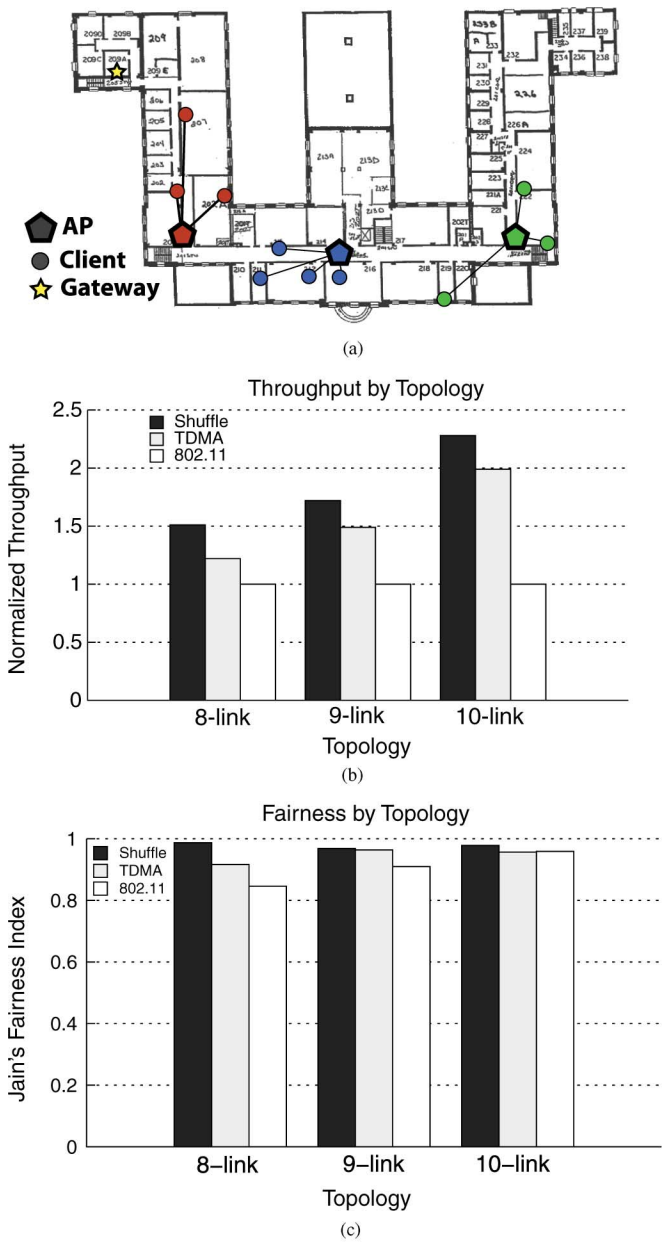


Fig. 15. (a) Example 10-link topology in our building. (b) Throughput and (c) fairness on entire Shuffle testbed.

improvements in this and other topologies are reasonable and consistent. Fairness among the links is also observed to be high, as illustrated in Fig. 15(c).

Complete Shuffle With Rate Control: We evaluate the benefits of MIM-aware ordering with rate control enabled. Our transmission bit-rate control mechanism is similar in principle to SampleRate and utilizes all 802.11g rates. Bit-rate control and MIM-aware transmission 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

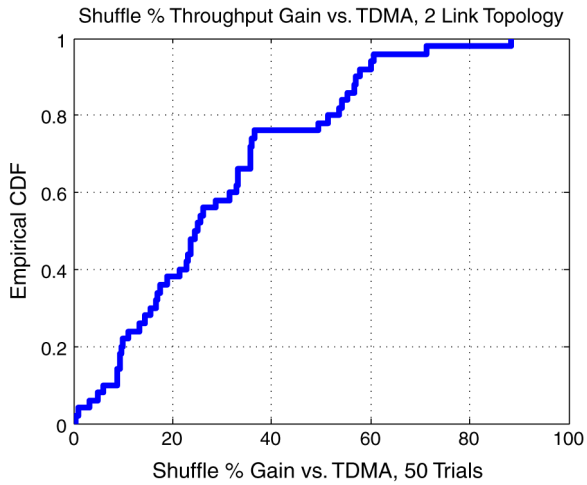


Fig. 16. Throughput for Shuffle versus TDMA using 802.11g with 6–54-Mbps rate control enabled.

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, with this testing methodology, results for 802.11 are not reported in this section.

In Fig. 16, we present results from one topology consisting of two mutually interfering links, similar to that presented in Fig. 4. One link is 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. The mean throughput gain from Shuffle is 29%.

Although the potential for gains with MIM ordering is topology-dependent, it is not highly sensitive. As depicted in Fig. 17, 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 to each of the 54 seating positions, we create a diverse set of channel conditions. Fig. 18 shows these results. Mean throughput is 32 and 27 Mbps for Shuffle and TDMA, respectively (a Shuffle gain of 17%).

A. Simulation Results

We performed QualNet simulations to evaluate performance in larger topologies. MIM capabilities were carefully modeled

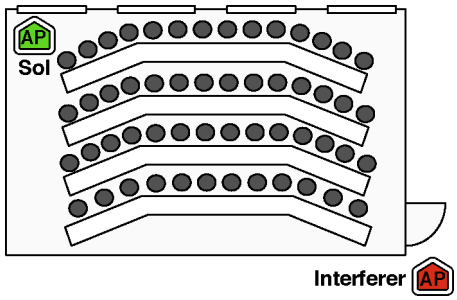


Fig. 17. Classroom environment with 54 seats. Leaving the AP and one client fixed, we tested with a client placed on the desk in front of each chair.

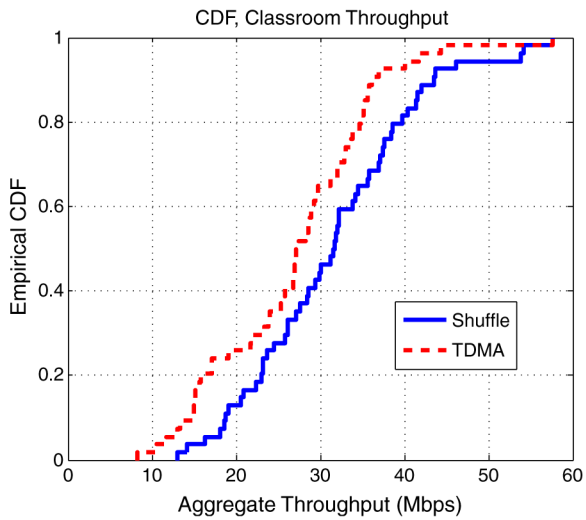


Fig. 18. CDF of throughput for classroom test.

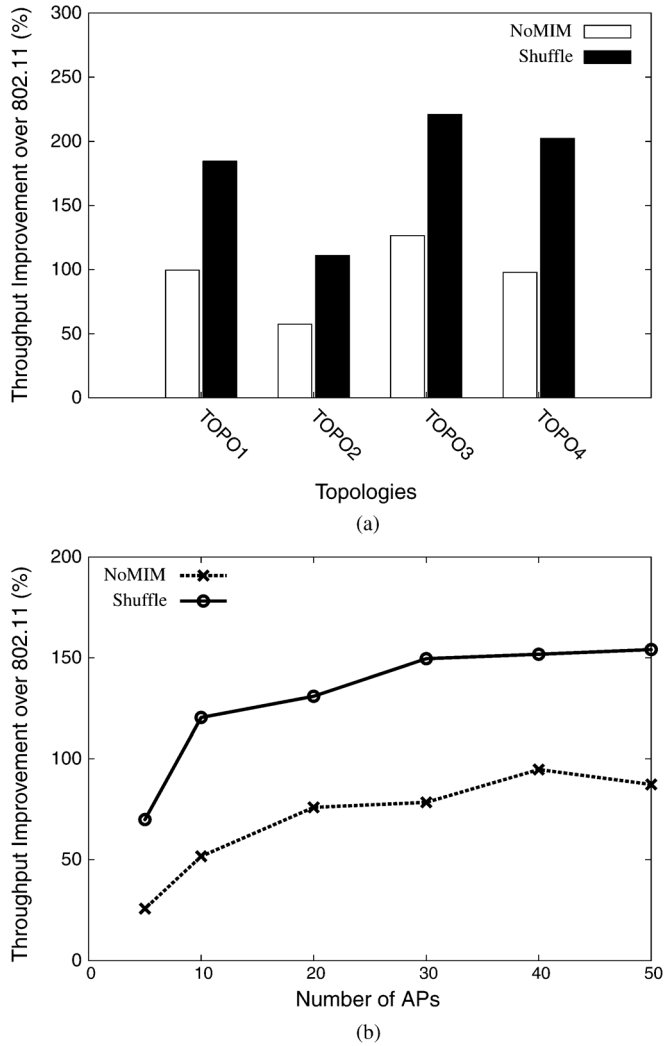


Fig. 19. Performance evaluation on real and synthetic topologies. (a) Throughput for real-life AP placements. (b) Higher AP density increases concurrency.

into the PHY and MAC layer of the simulator. The EWLAN controller was assumed to have a processing latency of 50 μs, and the wired backbone was assigned 1-Gbps data rate. We used 802.11a with transmission power 19 dBm, two-ray propagation model, transmission rate 12 Mbps, and a PHY-layer preamble duration of 20 μs.

Fig. 19(a) presents throughput comparisons for topologies taken from Duke University buildings with different numbers of APs on the same channel; each AP was associated to around six clients. As a special case, the second topology has APs associated to 20 clients, resembling a classroom setting. Shuffle consistently outperforms NoMIM, confirming the potential of MIM-aware reordering.

Impact of AP Density: To understand Shuffle's scalability in high-AP-density environments, we generated synthetic topologies in an area of 100×150 m^2. We placed an increasing number of APs (ranging from 5 to 50) at uniformly random locations in this region. Each AP is associated with four clients, and the controller transmits CBR traffic at 1000 packets/s to each of the clients. Fig. 19(b) presents results of this setting. It shows that Shuffle offers consistently better throughput than NoMIM regardless of the density of APs.

Impact of Fading: The earlier results were obtained without accounting for channel fading. However, the impact of channel fading can be severe, and the Shuffle system needs to adapt to it over time. To evaluate our opportunistic rehearsal mechanisms, we simulate Ricean fading with varying K-factors and log-normal shadowing. Fig. 20 shows the percentage throughput improvement of Shuffle over 802.11 for different values of K. For $K = 0$ (Rayleigh fading), the fading is severe, and the improvements are less than at higher values of K. Still, the improvements are considerable, indicating Shuffle's ability to cope with time-varying channels. The improvements were verified to be a consequence of opportunistic rehearsals; when opportunistic rehearsal was disabled, performance degraded.

VII. Limitations and Discussion

We discuss some limitations with Shuffle implementation 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 packet loss

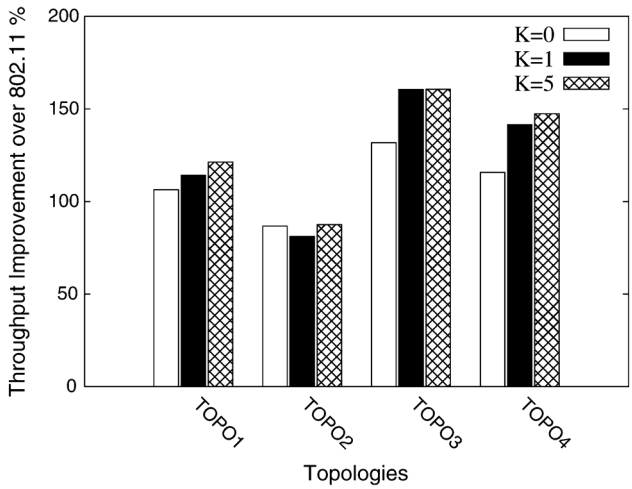


Fig. 20. Throughput improvement under different channel fading conditions. Shuffle performs well under Rayleigh and Ricean fading.

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 Shuffle 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 transmitting 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 direct improvement for retransmissions of lost packets, recall that retransmissions get higher priority than new packets to the same client. This is expected to make the retransmission delay tolerable.

Client Mobility: As a client moves, interference relationships between links 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 with Shuffle scheduling. 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. Though more experiments with larger topologies would be desirable to confirm the scalability of Shuffle, our simulation results indicate that Shuffle scales well.

VIII. Related Work

Capture and MIM: Theoretical models have been proposed to explain physical layer capture [19]. The first empirical evidence of capture was presented in [3]. The recent study in [5] quantifies SINR threshold requirements for 802.11a networks under different packet arrival timings. Capture awareness has been used for collision resolution in [20]. Bit error rate (BER) models for capture were proposed in [21].

Spatial Reuse: Schemes like [22] and [23] make use of power control and carrier-sense tuning to achieve improved spatial reuse. Prior work has considered RTS/CTS variants to schedule nonconflicting links [24]. However, most existing deployments do not use RTS and CTS [18], and even those with RTS/CTS do not exploit concurrency well. In CMAP [18], the authors propose a distributed scheme that 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 our earlier work [25], we have made a case for reordering transmissions. In this paper, we presented the details of the integer programming formulation, rehearsal, scheduling, and transmission mechanisms. The most significant addition is the implementation and evaluation of Shuffle on a testbed.

Enterprise Wireless LANs and Scheduling: Enterprise wireless LAN architecture is increasingly becoming popular to improve throughput, monitoring, and management. SMARTA [14] 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 [8] were proposed and in the context of EWLANs. Our controller–AP interaction is similar to the one proposed in a recent work [26], [27]. The speculative scheduling solution in [11] and [28] proposed a conflict-graph-based centralized scheduling mechanism to improve spatial reuse. Our conflict graphs are based on asymmetric link conflicts where conflicts change based on transmission ordering. Police [29] builds a conflict graph for both uplinks and downlinks in an EWLAN. This conflict graph can be dynamically updated and used for allocating airtime for links. Shuffle will benefit even more with such a scheme. DIRC [30] uses the conflict-graph-based approach in EWLANs for improving spatial reuse with directional antennas. We need to study the significance of MIM and ordering with directional antennas.

Characterizing and Measuring Interference: In [31], the authors analyze the effects of combined interference and suggest that an additive interference mechanism like the one used in QualNet [2] is a very close approximation. This assertion is further supported in [32]. An $\mathrm{O}(n^2)$ algorithm for estimating link state interference in multihop wireless networks was proposed in [33], and a linear order algorithm that takes capture into account was presented in [34]. These measurement schemes aim to estimate the pairwise interference between links with few measurements. In [35], 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 paper, we use a hybrid approach of measuring individual link RSSI values to prune the initial set of concurrent links and a decision-making scheme based on concurrent delivery ratios similar to [18].

IX. 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, we have shown that links that conventionally conflict with each other may be made concurrent if they are initiated in a specific order. We then presented Shuffle, a system that reorders transmissions to improve spatial reuse. Theoretical analysis has shown that the optimal improvements with MIM can be significant. A functional testbed validated that MIM-awareness is practical, while results of experimental evaluation confirm consistent performance improvements.

Acknowledgment

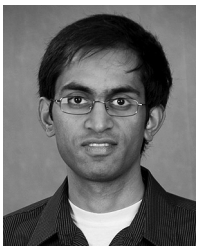
The authors sincerely thank V. J. Orlikowski (Duke OIT) for help during the early stages of the testbed installation. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

[1] "NS2," [Online]. Available: http://www.isi.edu/nsnam/ns/
[2] "Qualnet v2.6.1," Scalable Network Technologies, Los Angeles, CA [Online]. Available: www.scalable-networks.com
[3] A. Kochut, A. Vasan, A. U. Shankar, and A. Agrawala, "Sniffing out the correct physical layer capture model in 802.11b," in *Proc. IEEE ICNP*, 2004, pp. 252–261.
[4] T. Nadeem and L. Ji, "Location-aware IEEE 802.11 for spatial reuse enhancement," *IEEE Trans. Mobile Comput.*, vol. 6, no. 10, pp. 1171–1184, Oct. 2007.
[5] J. Lee, W. Kim, S.-J. Lee, D. Jo, J. Ryu, T. Kwon, and Y. Choi, "An experimental study on the capture effect in 802.11a networks," in *Proc. ACM WinTECH*, 2007, pp. 19–26.
[6] "Soekris Net4521 and Net4826-Series," Soekris Engineering, Inc., Santa Cruz, CA [Online]. Available: http://www.soekris.com/
[7] "Madwifi-ng driver," Linux-Consulting, Reno, NV [Online]. Available: http://madwifi.org/
[8] S. Ramanathan and E. L. Lloyd, "Scheduling algorithms for multihop radio networks," *IEEE/ACM Trans. Netw.*, vol. 1, no. 2, pp. 166–177, Apr. 1993.
[9] Solver CPLEX. Ilog, Inc., Armonk, NY, 2003.
[10] P. Bahl, J. Padhye, L. Ravindranath, M. Singh, A. Wolman, and B. Zill, "DAIR: A framework for managing enterprise wireless networks using desktop infrastructure," presented at the HotNets IV, 2005.
[11] N. Ahmed, V. Shrivastava, A. Mishra, S. Banerjee, S. Keshav, and K. Papagiannaki, "Interference mitigation in enterprise WLANs through speculative scheduling," in *Proc. ACM Mobicom*, 2007, pp. 342–345.
[12] Meru Networks, "Revolutionizing wireless LAN deployment economics with the Meru Networks radio switch," Meru Networks, Sunnyvale, CA, White Paper, 2005.
[13] R. Murty, J. Padhye, R. Chandra, A. Wolman, and B. Zill, "Designing high performance enterprise Wi-Fi networks," in *Proc. NSDI*, 2008, pp. 73–88.
[14] N. Ahmed and S. Keshav, "SMARTA: A self-managing architecture for thin access points," in *Proc. ACM CoNEXT*, Dec. 2006, Article no. 9.
[15] Y. Chung Cheng, J. Bellardo, P. Benko, A. C. Snoeren, G. M. Voelker, and S. Savage, "Jigsaw: Solving the puzzle of enterprise 802.11 analysis," in *Proc. ACM SIGCOMM*, 2006, pp. 39–50.
[16] J. Bicket, "Bit-rate selection in wireless networks," Master's thesis, Massachusetts Institute of Technology, Cambridge, MA, 2005.
[17] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, "The click modular router," *Trans. Comput. Syst.*, 2000.
[18] M. Vutukuru, K. Jamieson, and H. Balakrishnan, "Harnessing exposed terminals in wireless networks," in *Proc. NSDI*, 2008, pp. 59–72.
[19] M. Durvy, O. Dousse, and P. Thiran, "Modeling the 802.11 protocol under different capture and sensing capabilities," in *Proc. IEEE INFOCOM*, 2007, pp. 2356–2360.
[20] K. Whitehouse, A. Woo, F. Jiang, J. Polastre, and D. Culler, "Exploiting the capture effect for collision detection and recovery," in *Proc. IEEE EmNetS*, May 2005, pp. 45–52.
[21] H. Chang, V. Misra, and D. Rubentein, "A general model and analysis of physical layer capture in 802.11 networks," in *Proc. IEEE INFOCOM*, 2006, pp. 1–12.
[22] T.-S. Kim, H. Lim, and J. C. Hou, "Improving spatial reuse through tuning transmit power, carrier sense threshold, and data rate in multihop wireless networks," in *Proc. ACM MobiCom*, 2006, pp. 366–377.
[23] K. Jamieson, B. Hull, A. Miu, and H. Balakrishnan, "Understanding the real world performance of carrier sense," in *Proc. ACM SIGCOMM E-WIND*, 2005, pp. 52–57.
[24] K. Mittal and E. M. Belding, "RTSS/CTSS: Mitigation of exposed terminals in static 802.11-based mesh networks," in *Proc. IEEE WiMesh*, 2006, pp. 3–12.
[25] N. Santhapuri, J. Manweiler, S. Sen, R. R. Choudhury, S. Nelakuditi, and K. Munagala, "Message in Message (MIM): A case for reordering transmissions in wireless networks," presented at the HotNets VII, 2008.
[26] R. Murty, A. Wolman, J. Padhye, and M. Welsh, "An architecture for extensible wireless lans," presented at the HotNets VII, 2008.
[27] R. Murty, J. Padhye, A. Wolman, and M. Welsh, "Dyson: An architecture for extensible wireless LANs," in *Proc. USENIX*, 2010, p. 15.
[28] V. Shrivastava, N. Ahmed, S. Rayanchu, S. Banerjee, S. Keshav, K. Papagiannaki, and A. Mishra, "CENTAUR: Realizing the full potential of centralized WLANs through a hybrid data path," in *Proc. ACM MobiCom*, 2009, pp. 297–308.
[29] K. Jang, M. Carrera, K. Psounis, and R. Govindan, "Passive on-line in-band interference inference in centralized WLANs," University of Southern California, Los Angeles, CA, Tech. Rep., 2010.
[30] X. Liu, A. Sheth, M. Kaminsky, K. Papagiannaki, S. Seshan, and P. Steenkiste, "DIRC: Increasing indoor wireless capacity using directional antennas," in *Proc. ACM SIGCOMM*, 2009, pp. 171–182.
[31] D. N. C. Tse and S. Hanley, "Linear multiuser receivers: Effective interference, effective bandwidth and user capacity," *IEEE Trans. Inf. Theory*, vol. 45, no. 2, pp. 641–675, Mar. 1999.
[32] R. Maheshwari, S. Jain, and S. R. Das, "A measurement study of interference modeling and scheduling in low-power wireless networks," in *Proc. ACM SenSys*, 2008, pp. 141–154.
[33] J. Padhye, S. Agarwal, V. Padmanabhan, L. Qiu, A. Rao, and B. Zill, "Estimation of link interference in static multi-hop wireless networks," in *Proc. ACM SIGCOMM IMC*, 2005, p. 28.
[34] J. Lee, S.-J. Lee, W. Kim, D. Jo, T. Kwon, and Y. Choi, "RSS-based carrier sensing and interference estimation in 802.11 wireless networks," in *Proc. IEEE SECON*, 2007, pp. 491–500.
[35] C. Reis, "Measurement-based models of delivery and interference in static wireless networks," in *Proc. ACM SIGCOMM*, 2006, pp. 51–62.

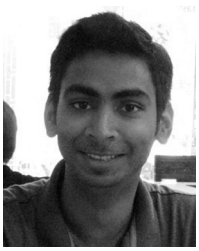
Justin Manweiler received the B.S. degree from The College of William and Mary, Williamsburg, VA, in 2007, and is currently pursuing the Ph.D. degree in computer science at Duke University, Durham, NC.

His research interests include wireless networking and mobile computing.



Naveen Santhapuri received the B.Tech. degree from Andhra University, Visakhapatnam, India, in 2002, and the M.S. degree in wireless network security and Ph.D. degree in wireless networks from the University of South Carolina, Columbia, in 2005 and 2009, respectively.

He was a Post-Doctoral Associate with Duke University, Durham, NC, from 2009 to 2010, doing research in wireless networks and mobile computing. He is currently a Software Developer in New York, NY.



Souvik Sen received the B.E. degree in computer science from Bengal Engineering and Science University, Howrah, India, in 2007, and the M.S. degree in computer science from Duke University, Durham, NC, in 2011, and is currently pursuing the Ph.D. degree in computer science at Duke University.

His research interests include wireless networking and mobile computing.

Romit Roy Choudhury received the Ph.D. degree in computer science from the University of Illinois at Urbana–Champaign in 2006.

He is an Associate Professor of electrical and computer engineering and computer science with Duke University, Durham, NC, which he joined in 2006 after completing his Ph.D. studies. His research interests are in wireless protocol design mainly at the PHY/MAC layer and in mobile social computing at the application layer.

Dr. Roy Choudhury received the NSF CAREER Award in January 2008.

Srihari Nelakuditi received the B.E. degree from Andhra University College of Engineering, Visakhapatnam, India, in 1989, the M.Tech. degree from the Indian Institute of Technology, Madras, India, in 1991, and the Ph.D. degree in computer science from the University of Minnesota, Minneapolis, in 2001.

He is currently an Associate Professor with the University of South Carolina, Columbia. He was previously a Software Design Engineer with Texas Instruments, Bangalore, India. His research interests are in resilient routing, wireless networking, and mobile computing.

Dr. Nelakuditi is a recipient of the NSF CAREER Award in 2005.

Kamesh Munagala received the Ph.D. degree in computer science from Stanford University, Stanford, CA, in 2003.

He is an Associate Professor of computer science with Duke University, Durham, NC, where he has been since Fall 2004. He spent a year as a Post-Doctoral Scholar doing computational biology at the Department of Biochemistry, Stanford University School of Medicine. His research interests span approximation algorithms, stochastic optimization, and computational economics.

Dr. Munagala is a recipient of the NSF CAREER Award, an Alfred P. Sloan Research Fellowship, and a Best Paper Award at the World Wide Web Conference in 2009.