Mozart: Orchestrating Collisions in Wireless Networks

Tarun Bansal, Bo Chen, Kannan Srinivasan and Prasun Sinha
Deptt. of Computer Science, Ohio State University, Columbus, USA

***Abstract*—Distributed packet scheduling in a wireless network is non-trivial due to lack of knowledge of *which* node has packets to transmit. This paper describes and evaluates Mozart, a new and practical approach that encourages collisions rather than trying to avoid them. Upon receiving multiple collided packets, the receiver carefully selects the retransmitters, so that all the packets can be recovered in the fewest number of slots. Our design enables the nodes to transmit without any backoffs during this recovery phase. To enable Mozart, we also propose a novel algorithm for estimating the Received Signal Strength (RSS) of a packet in presence of high interference. We implement Mozart on the USRP N210 radio and evaluate its performance on a USRP testbed. Our experiments show that Mozart's throughput is up to 3.70x compared to IEEE 802.11. Using real world traces, we show that on average, Mozart provides a network throughput of 2.55x and 4.10x when compared to 802.11ec and 802.11, respectively.**

I. Introduction

Previous studies have shown that the throughput achieved in real wireless networks is often significantly lower than their capacity [22]. This difference is frequently attributed to *discrepancy between the interference at the transmitter and the receiver.* In wireless networks, it is impossible for transmitters to precisely estimate the interference at receivers which leads to hidden and exposed terminal scenarios. It has been shown [22] that hidden and exposed terminals can cause throughput loss in 30% and 61% links, respectively. Collisions due to hidden terminals are especially common in indoor wireless networks with obstacles [2]. Secondly, IEEE 802.11 protocol and its derivatives (e.g., 802.11 with RTS-CTS, 802.11ec [16], ZigZag [6] etc.) require the channel to remain idle when the nodes are undergoing backoffs. Such *idle listening* is necessary in these protocols to avoid collisions and leads to up to 30% loss of throughput [12].

The above two factors significantly impact the throughput. Most practical distributed scheduling algorithms in today's wireless networks such as the IEEE 802.11 family and IEEE 802.15.4 protocols are based on CSMA (Carrier Sense Multiple Access). Various improvements to these schemes have been proposed in recent years. RTS-CTS (IEEE 802.11) based approaches have been proposed for addressing the hidden terminal problems. Approaches to reduce the overhead of RTS-CTS packets [16], channel wastage due to collided packets [19], and overhead of backoff [20] have been proposed. Several mechanisms that attempt to salvage bits or entire packets out of collided transmissions have also been proposed [6], [14], [11], [8], [15]. However, these algorithms are insufficient (details in Sec. VII) as they either do not work in multi-collision domains [6], [8], abandon the bits under collision [11], [19], require multiple antennas per node [20] or have long critical periods resulting in lower packet delivery ratio [16].

This paper presents a new cross-layer algorithm called *Mozart*, that encourages collisions and hidden terminal transmissions in a planned way to enable fast recovery of colliding packets via a new approach called *Successive Packet Subtraction* (SPS). In Mozart, receivers encourage neighboring nodes to transmit simultaneously resulting in collision. The receiver then smartly suppresses transmissions in subsequent slots based on its estimation of signal strengths from various senders so as to best apply SPS to recover all the colliding packets (See Figure 1). *SPS allows wireless receivers in a multi-collision domain network to receive and decode multiple packets without knowing which node has data to send to which other node.*

Mozart eliminates backoffs before transmission of data packets. Mozart also embraces hidden terminal transmissions and therefore implicitly addresses it. Thus, Mozart provides throughput improvement for both *downlink* and *uplink* traffic. Through theoretical analysis, we show that the *critical period* of Mozart is 2μs. This is much shorter than the critical period of the state-of-the-art schemes that have critical period of 39.4 μs [16]. The shorter critical period ensures higher packet delivery ratio in Mozart. This paper, makes the following additional contributions:

- We propose Successive Packet Subtraction (SPS), a new approach for wireless nodes to simultaneously receive multiple packets and then decode them one-by-one while controlling the retransmission pattern. Unlike SIC (Successive Interference Cancellation [21]), SPS works even when all packets have low SINR.
- On receiving collided packets, receivers in Mozart need to identify the set of transmitters and estimate their received signal strengths with high precision. Doing both in the presence of multiple collisions is extremely hard. Existing techniques to measure signal strength do not work in the presence of collisions[8]. We present a novel iterative algorithm for estimating the RSS of multiple colliding packets. Our results show that even in the presence of interference from 14 other packets, Mozart is able to estimate the RSS within 1 dB with 96% accuracy compared to 21% accuracy of state of the art schemes.
- This paper proposes an algorithm to compute the set of nodes to be suppressed at the end of each slot such that the probability of correctly decoding the packets is maximized across all slots.
- We implement Mozart on a USRP testbed [1]. Our evaluation results show that Mozart's throughput is up to 3.70x compared to IEEE 802.11. Our trace-driven evaluations show that on an average, Mozart provides a throughput of 2.55x and 4.10x compared to 802.11ec and 802.11, respectively.

The rest of the paper discusses our algorithms, implementation and comparison results in detail.

II. Mozart: Detailed Description

Mozart works by *encouraging collisions* among transmissions. However, upon collision, the receiving node controls the

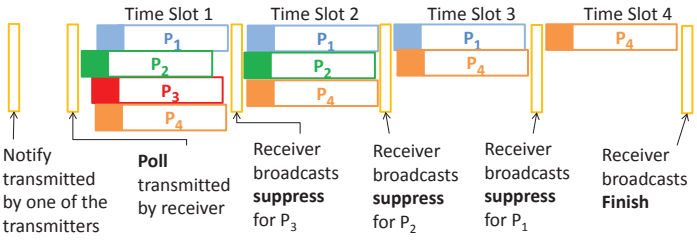


Fig. 1. Collision Recovery Period. Through control messages, the receiver ensures that the number of transmitters is reduced by 1 from the previous slot. Data transmissions in the same slot may arrive at different times at the receiver due to propagation delays and radio's TX-RX turn-around time. At the end of the recovery period, the receiver will reconstruct samples for P_4 and subtract it from samples received in slot 3. The remaining samples are then decoded to obtain P_1. Similarly, P_2 and P_3 are decoded from the samples of slot 2 and slot 1, respectively.

future retransmissions such that it is able to decode the collided transmissions in a short time. The next subsection explains the working of Mozart in detail. Section III explains how Mozart handles various practical challenges.

A. Successive Packet Subtraction (SPS)

In Mozart, packet transmissions and the subsequent decoding takes place in multiple phases (Fig. 1):

1. **Notification**: If a node B has data for A, then B sends a notification message to A, indicating that B has outstanding data for A.
2. **Polling**: Upon receiving the notification message, A backs off for a random duration (between 1 and 5 μs as described in Subsection III-D) and sends a *poll* message to solicit transmissions in its neighborhood. After sending the poll, A is said to be undergoing *recovery*. However, in Mozart, a node (say A) can poll only if all the following conditions are true: (i) It observes (through virtual sensing) that no other polling node in its neighborhood is *recovering*; (ii) No other node in its neighborhood is transmitting data to its receiver; and, (iii) A's backoff is over. If any of these condition is not true, A undergoes a backoff and then attempts to poll again.
3. **Data Transmission**: Upon receiving a poll from a neighboring node (say A), a node (say B) sends a data packet to A if both these conditions are true: (i) There is no other node (except A) in its neighborhood undergoing recovery; and, (ii) B has data to send to the polling node.
4. **Suppress**: The polling node may simultaneously receive data packets from multiple nodes resulting in a collision. From the received packets, the polling node selects one transmitting node and transmits *suppress* to it. Upon receiving suppress, the node stays silent for the remaining duration of the recovery period. Other nodes upon overhearing the suppress, re-transmit the same data packets until they either receive the *suppress* or the *finish packet* from A. Section III-C explains how the polling node selects the node to whom the suppress is sent. The receiver also stores the received samples in a buffer for decoding in the future. Mozart does not have any synchronization overhead since synchronization happens *implicitly* through the poll and suppress messages transmitted by the receiver.
5. **Finish**: In each slot one of the transmitting nodes becomes silent after receiving the suppress from A. So, the number of colliding packets decreases by one in each slot. Eventually, in some slot, only one node transmits. At the end of that slot, the polling node decodes all the packets (Explained in detail in Section II-A2). Upon successful decoding, A broadcasts a *finish* packet. The finish packet serves two purposes: (i) It indicates successful decoding, thereby acknowledging the transmitted data, allowing nodes to transmit new data; and, (ii) Indicates the end of the recovery period, thus allowing nearby nodes to transmit poll packets or data packets.

By reserving the channel in the neighborhood of the receiver (using the poll message) as well as by encouraging collisions among transmitters, Mozart is able to implicitly handle *hidden transmissions*. Similarly, before transmitting a poll packet or a data packet, a node ensures that no other node in its neighborhood is recovering. This check prevents its transmissions from interfering with any data packets that the recovering node in the neighborhood might receive. To determine if any other node is recovering, nodes use virtual sensing by monitoring poll/suppress packets and the matching finish packets. If a node receives a poll packet but not the corresponding finish packet, it indicates that the other node is recovering. In case when the node does not receive the matching finish packet, timeout after the last received poll/suppress packet can be used to indicate the end of recovery. The timeout duration is set to twice the channel access time as discussed in Section III-B. If a node receives no packet transmissions after polling, then it sends a finish and enters a backoff period before polling again.

1) Packet Structure: To transmit the suppress at the end of each slot, the receiver needs to determine the id of at least one transmitter. However, with colliding transmissions, the receiver cannot decode the transmitter's ID from the packet's MAC header. To that end, Mozart requires that transmitters identify themselves by sending a Correlatable Symbol Sequence (CSS) before the preamble. It has been shown [16] that CSSs[1] can be correlated without correcting for frequency or sampling offsets. Another property of CSSs is that they can be correlated even under collision. This allows a node to receive a control message even if its neighboring node is transmitting. Thus, Mozart allows neighboring nodes to simultaneously transmit, thereby eliminating the *Exposed Terminal Problem*. Further, usage of PN sequences reduces the transmission duration of the packet considerably as demonstrated before [16].

Mozart uses 5 different packet types: (i) **Notify**: It consists of the PN sequence of the receiver; (ii) **Poll**: It consists of the PN sequence of the polling node; (iii) **Suppress**: It contains the PN sequence of the receiver followed by the PN sequence of the node being suppressed; (iv) **Data packet**: This packet has the PN sequence of the transmitter, PN sequence of the receiver, preamble, MAC header and the data body; and, (v) **Finish**: It contains the PN sequence of the receiver, followed by PN sequences of the nodes whose transmissions were successfully decoded. *By using PN sequences that are correlated[16] by the receiver, Mozart significantly reduces the rate of loss of the control packets due to collisions.*

2) Packet Decoding: In Mozart, the receiver decodes all received packets in the *reverse chronological order*. Decoding starts with the last slot in which only one node transmitted. Figure 1 shows a recovery period in which four nodes send data in response to A's poll. Three suppress packets were sent at the

[1]Also referred to as Pseudo Random Sequences (PN sequences)

end of the first three slots, and, so in the fourth slot, only one node transmits. In that slot, P_4 is available in the clear and can simply be decoded. To decode P_1, after correcting for different offsets (details in Section III-F), A recreates samples for P_4 as received in slot 3 and subtracts them from samples received in slot 3. After subtraction, it is left with only the samples corresponding to P_1 which A decodes. Similarly, to decode P_2, A re-creates samples for both P_1 and P_4 and subtracts them from the samples received in slot 2. After subtracting, it decodes the remaining samples to get P_2. This process is repeated for each slot until A has decoded all the collided packets. Thus, by carefully selecting the retransmitters, A is able to decode 4 packets in 4 slots.

SPS allows the receiver to control the retransmission pattern without precisely knowing the complete set of transmitters in each slot. This is in contrast with a naive approach where all transmitters transmit their PN sequences in the first slot and then the receiver schedules different transmitters in a TDMA fashion. In this naive approach, the receiver may not detect PN sequences with low SINR (See Section IV) resulting in their starvation. On the other hand, when using SPS, such transmitters will be detected in later slots when the number of colliding packets become small. *Thus, the flexible approach of SPS increases fairness.* This approach is also different from 802.11 polling mode where the APs poll all the potential transmitters. This results in wasted polls and throughput loss when a transmitter has no data to send in response to a poll. In contrast to that, receivers in Mozart receive packets from all the transmitters and decode them by controlling the retransmission pattern.

B. Challenges towards practical implementation

Although the idea behind Mozart is simple, however, multiple challenges need to be solved to make Mozart practical:

1. **Identifying set of transmitters and estimating their RSS:** To maximize the decoding accuracy, at the end of each slot, receivers in Mozart need to carefully determine the transmitter to be suppressed. As explained in Section III-C, this requires receivers in Mozart to determine the following additional information: (i) ID of transmitters; and, (ii) SINR of transmitters. However, determining this information for all packets in the presence of interference is challenging. Existing techniques[8] to measure signal strength do not work in the presence of collisions and require transmitters to transmit one-by-one, thus constituting a significant overhead.
2. **Determining transmitter to suppress:** Once the set of transmitters is determined, the receiver in Mozart needs to determine which transmitter should be suppressed among all transmitters such that the decoding accuracy is maximized. Different transmitters may have different SINR and may transmit at different physical layer data rates. This makes it difficult to determine which transmitter should be suppressed.
3. **Handling heterogeneous data rate and packet sizes:** The channel access time is fixed for all transmitters in Mozart. However, if some transmitters only have small amount of data to send, this may lead to channel wastage.

III. Practical Considerations

This section explains how we handle the aforementioned challenges. In Subsection III-A, we explain how receivers in Mozart identify the set of transmitters and estimate their RSS. Subsection III-B explains how transmitters in Mozart determine the physical layer data rate to be used. Once the receiver knows the id of transmitters, their RSS and the modulation scheme used, the receiver then uses the algorithm described in Subsection III-C to determine the set of nodes to be suppressed at the end of each slot. Subsection III-D explains how compared to existing algorithms, Mozart significantly increases the probability of successful packet decoding. The next subsection explains how receivers in Mozart handle decoding errors. Finally, Subsection III-F explains how offset correction and PN sequence assignment is done in Mozart.

A. Identification and RSS estimation of collided packets

It is beneficial for Mozart to identify the transmitters of all the collided packets, since with more IDs, it is more likely that the receiver will find the right set of transmitters to suppress. Observe that the receiver does not need to determine the source ID of all the packets. However, more IDs it can determine, higher are the chances that the receiver will suppress the right set of transmitters (explained later in Section III-C).

For -8dB SINR, state of the art correlation schemes have been shown[16] to suffer from as much as 70% false negatives when 127-symbol Gold code sequence (a type of CSS) is used. Clearly, this false-negative rate is too high for Mozart where multiple transmissions may collide resulting in low SINR. Our identification algorithm is also based on the general approach of computing the cross correlation, however, there are significant differences when compared to existing schemes[6], [19], [16]:

1. **Bounding cross correlation by circular padding**: To improve the accuracy of correlation, we harness a property of Gold codes that guarantees that the circular cross correlation of two instances of Gold code is bounded [5]. For 127-symbol sequence Gold codes, the bound is 1/7. Thus, the correlation works better when Gold codes collide only with other Gold codes, and not with samples from arbitrary data packets. Since in Mozart, the nodes do not undergo backoffs before transmitting data, the PN sequences for the collided packets are expected to be *approximately-synchronized*. However, due to propagation delays and hardware artifacts, it is possible that the PN sequences do not collide with other PN sequences at the receiver. Based on the IEEE 802.11 standard[10], the arriving time for different PN sequences can differ by at most 4μs (2μs for radio's turn-around time[16] and 1 μs for propagation delay in each direction[16]). So the transmitters in Mozart cyclically pad 2μs symbols of the Gold code before and after the actual Gold code. The symbols padded before a certain Gold code instance are the last few symbols of Gold code while the symbols padded after the end are the first few symbols. This ensures that the Gold code samples interfere with only the combination of samples of other Gold codes resulting in low cross-correlation. Figure 2(b) illustrates the padding process.
2. **Improving correlation for low RSS packets through cancellation**: Instead of trying to detect all the collided Gold codes at the same time, Mozart decodes one Gold code at a time. Then, it subtracts the samples of the detected Gold code from the received samples. Figures 2(c) and 2(d) show two packets with Gold codes and data before subtraction and after Gold code of P_1 was subtracted from the collided samples. This process is similar to the one that we used during the

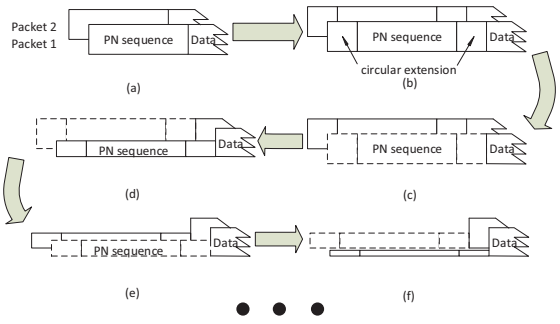


Fig. 2. Iterative algorithm for transmitter identification and RSS estimation.

packet cancellation step (See Section II). This allows Mozart to detect those packets that have low RSS even if they are interfered by high RSS packets since after subtracting the high RSS packet, the correlation value for the lower RSS packet increases. Subtracting packet 1 reduces the noise for P_2 and thus, helps in improving the SINR of the Gold code in P_2 as shown in Figures 2(c) and 2(d).

3. **Iteratively improving the RSS estimation**: In the above two steps, it is possible that due to high interference from other packets, the computed correlation is not correct resulting in inaccurate estimation of RSS. In such a case, even after subtraction, a part of the packet (residual samples after subtraction) would still be left in the original samples (albeit with much lower RSS). To correct the RSS estimate, Mozart performs correlation and cancellation repeatedly to detect these copies of the packet. For this, Mozart records the correlation value and the location (i.e., starting time) for each detected packet. When Mozart detects some packet again that was already detected in previous steps, then the power level of the packet is updated to be the sum of the current and old correlation values. As Mozart detects the same packet multiple times, its estimate of the RSS of the packet improves while the residual signal strength of the packet decreases. For examples, Mozart detects Gold codes of Packet 1 first in Figure 2(c) and later in Figure 2 (e) among the residual samples. Upon, detection for the second time, the receiver estimates the RSS of the residual Gold code of Packet 1 and adds that estimate to the previous estimate of RSS of Packet 1. This step is repeated as long as the correlation value is at least twice the strength of the residual samples.

Thus, the first technique is used by the transmitter while second and third techniques are iteratively used by the receiver for improving the accuracy of identifying the set of transmitters and estimating their RSS. Apart from this, the receivers in Mozart need to determine the their modulation and coding scheme used by each of the transmitters. In our algorithm, multiple CSSs are assigned to represent different physical layer data rates. The transmitter of the data packets include the CSS that corresponds to the data rate used for that particular data packet.

B. Heterogeneous data rate and packet sizes

Due to varying SNR, different transmitters may select different physical layer data rates when transmitting to the same receiver. Since, wireless channels are bidirectional in nature (shown in [7]), so the transmitters in Mozart use the received poll packet to estimate the channel to the receiver. To that end, the transmitters correlate the received poll packet with the known PN sequence. The peak value of the correlation indicates the channel quality between the transmitter and the receiver with higher correlation value indicating better channel quality. This is similar to estimating the channel quality through preamble and has been well studied[6] in the context of improving the accuracy of recreating the samples. However, the key difference is that here, the transmitter estimates the channel to the receiver using the received poll packet and then, transmits the data packet at a rate that is suitable for the channel's current condition. *Thus, the poll packet transmitted by receivers in Mozart also allows transmitters to pick the appropriate physical layer data rate without extra overhead.*

During each recovery, Mozart fixes the channel access time for each slot. Thus, a node with higher data rate may be able to send more bits compared to a node with lower data rate. However, if a node does not have enough pending data, then it will reduce its data rate such that its transmission still fits in the slot size. This reduced data rate sometimes allows the receiver to simultaneously decode two transmissions using Successive Interference Cancellation (SIC) [21]. SIC is a well known physical layer technique that is used by the wireless receivers to simultaneously decode two packets where the packet with the higher *noise tolerance* is decoded first, followed by the packet with the lower noise tolerance. Since in Mozart, multiple transmitters transmit simultaneously, this increases the probability that the receiver will find a pair of packets among the received transmissions that satisfy the requirements for simultaneous decoding.

For the example scenario of Figure 1, let's say that the transmitter of P_1 has small amount of pending data. Thus, on receiving the poll, it would transmit P_2 at the lowest possible data as explained above. At the end of first slot, the receiver observes that P_1 has been transmitted at low physical layer data rate and high power, and thus can be decoded even in the presence of P_3 as noise. So, it will send suppress to both P_1 and P_3. When decoding, the receiver decodes P_1 and P_3 using SIC and sends a *finish* packet indicating successful decoding. Mozart is able to harness SIC benefits since here, the transmitters *proactively* reduce their data rate. Thus, the results described in this paper are not in conflict with [21] where the authors had argued that SIC has limited applicability when not applied proactively.

C. Determining set of nodes to suppress

Observe that during reverse chronological decoding, the residual noise is higher in the earlier slots since more packets need to be subtracted before actual decoding (*e.g.* when decoding P_3 in Fig.1, the residual noise would come from P_1, P_2 and P_4). Therefore, to improve the decoding accuracy, receivers in Mozart suppress transmissions with high noise tolerance in earlier slots. This approach maximizes the probability of successful decoding across all the slots of the recovery, thereby improving the resilience. Further, to minimize the number of slots, Mozart first checks if it is possible to suppress two nodes simultaneously as explained in Sec. III-B.

Next, we explain Algorithm 1, that determines the set of nodes to be suppressed at the end of the current slot. Here, T_{ij} denotes the *noise tolerance* (in mW) of the pair of packets P_i and P_j, which denotes the maximum residual noise level that

can be present during successful decoding of these two packets. If a pair of packets has high value of T_{ij}, it implies that it is possible to decode both the packets using SIC even if residual noise left after canceling other packets is high. The algorithm also computes (Line 4) the noise threshold (in mW), τ that indicates the expected residual noise that will be left after the receiver has subtracted all packets that were received in this slot. τ is approximated by dividing sum of RSS (in mW) of all packets received in this slot by 100 (i.e., 20 dB)[2]. Then, in Lines 5-10, all those pairs of packets are added to the set $\mathbf{G}$ that have sufficient noise tolerance. If noise tolerance of P_j is higher than that of P_i (Line 6-7), then P_j would be decoded in presence of interference from P_i. Thus, its noise tolerance would be $\frac{RSS_j}{r_j} - RSS_i$ where r_j is the minimum SINR at which P_j can be successfully decoded with high probability[8]. On the other hand, noise tolerance of P_i would simply be $\frac{RSS_i}{r_i}$ since P_j would already be subtracted. For successful simultaneous decoding, we need to ensure that the residual noise is less than the minimum of these two values (Lines 7-9).

Finally, the algorithm returns the pair that has the highest noise tolerance (Line 10). However, for some slots, it may not be possible to do SIC and thus no pair of packets may have the required tolerance. Then (Lines 12-15), Mozart finds a single packet that has the highest noise tolerance. It is also possible that during the recovery period, the receiver does not find any node with noise tolerance above the expected residual noise. In that case, the receiver handles the errors as explained in Subsection III-E.

Algorithm 1: Computes the set of nodes that should be sent suppress in this slot

1 **Input:** For each packet P_i received in this slot: its power level expressed in mW (RSS_i) and the minimum SINR level (r_i, expressed as a dimensionless ratio) at which it can be decoded. r_i is contingent upon the physical layer data rate used to transmit P_i.
2 **Output:** Set of packets (or corresponding transmitters) that should be sent suppress in this slot.
3 $\mathbf{P} \leftarrow$ {Set of packets received in this slot}, $\mathbf{G} \leftarrow \{\}$
4 $\tau \leftarrow \frac{Sum\ of\ RSS\ of\ all\ packets\ in\ \mathbf{P}}{100}$
5 **for** $(P_i, P_j) : P_i, P_j \in \mathbf{P}$ **do**
6 **if** $\frac{RSS_j}{r_j} \geq \frac{RSS_i}{r_i}$ **then**
7 $T_{ij} \leftarrow \min\{\frac{RSS_j}{r_j} - RSS_i, \frac{RSS_i}{r_i}\}$
8 **else** $T_{ij} \leftarrow \min\{\frac{RSS_i}{r_i} - RSS_j, \frac{RSS_j}{r_j}\}$
9 **if** $T_{ij} > \tau$ **then** $\mathbf{G} \leftarrow \mathbf{G} \cup \{(P_i, P_j)\}$
10 **if** $\mathbf{G} \neq \{\}$ **then return** $\arg\max_{(P_i,P_j)\in\mathbf{G}} T_{ij}$
11 **else**
12 $T_i \leftarrow \frac{RSS_i}{r_i}\ \forall\ P_i \in \mathbf{P}$
13 $\mathbf{S} \leftarrow \{P_i : P_i \in \mathbf{P} \wedge T_i > \tau\}$
14 **if** $\mathbf{S} \neq \{\}$ **then return** $\arg\max_{P_i\in\mathbf{S}} T_i$
15 **else** Send finish

D. Near-Zero Critical Period

We define *Critical period* of a MAC algorithm as the duration of the interval, before and after the beginning of a transmission, during which an interfering transmission may corrupt *both the transmissions*. *MAC protocols with longer critical periods are expected to have a higher collision rate and thus, lower throughput.* The longer critical period also requires the nodes to spend excessive time in backoff. In Mozart, if a single receiver receives multiple colliding packets, it can still decode all those packets by suppressing one transmitter in each slot. So, intuitively, Mozart should have shorter critical period compared to existing algorithms. In [4], we formally show that the critical period of Mozart is 2 μs[3]. By comparison, the critical period of some standard protocols are [4]: 152 μs for 802.11 with RTS-CTS and 39.4 μs for 802.11ec.

E. Handling Decoding Errors

While decoding, it is possible that the channel noise or residual noise may cause a failed checksum resulting in decoding error. In that case, the receiving node follows one of the three options: (i) It first requests re-transmission of data for that slot. For example, in Figure 1, if receiver A is unable to decode P_2 during slot 2, then it will ask P_2's transmitter to re-transmit. The receiver performs this notification by sending a special *nack* message to the transmitter of P_2. Upon receiving the retransmitted data, A can decode P_2 and continue to decode P_1 using the slot 1 samples. (ii) However, if P_2's transmitter is not able to transmit P_2 because one of its other neighbors is undergoing recovery, then A decodes the remaining packets by simply subtracting the samples of P_2 from the earlier slots (without actually decoding P_2). The receiver would decode as many packets as possible using this scheme. (iii) Finally, if the receiver decoded only a subset of packets, then it would send *finish* with the PN-sequence of only the transmitters whose packets it was able to decode.

F. Offsets Correction and PN Sequence Assignment

The efficiency of successive packet subtraction in Mozart depends on the accuracy of re-creating the samples. To increase the accuracy, the phase, frequency and sampling offsets need to be compensated [6]. This problem of computing the offsets in presence of interference has been well studied [14], [6], [7]. In our implementation, after evaluating multiple schemes, we decided to use the one proposed in ZigZag[6] as it gave the best results. Mozart also requires that every node in the network be assigned a PN sequence. Further, no two neighboring nodes should be assigned the same PN sequence. Magistretti et al. [16] argue that such an assignment can be either done by APs or can be done using hash functions. In the experiments and evaluations of Mozart, the PN sequences were assigned using the former approach.

IV. Experiments

In this section, we describe the results from our experiments performed on the GNU radio platform and a testbed of Universal Software Radio Peripheral (USRP) N210 version 4 [1] radios. We used WBX daughterboards[1] as the RF front end. Mozart performs decoding at the sample level, and thus, is independent of the modulation and coding choice for transmission. In our experiments, we used BPSK with 1/2 convolutional code. Our re-creation process compensates for the phase offset, frequency offset and the sampling offset as

[2]20 dB denotes the cancellation that can be achieved by subtracting the packet in most cases (See Section IV for detailed experiment results).

[3]To reduce the probability of two neighboring transmissions from colliding, we require nodes to backoff for a short duration (5 μs) before sending a poll.

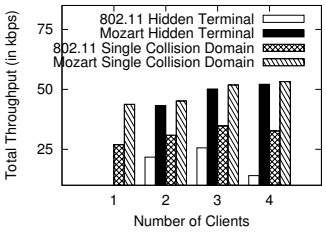
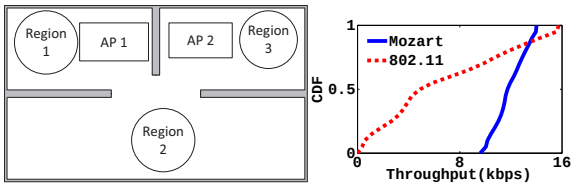
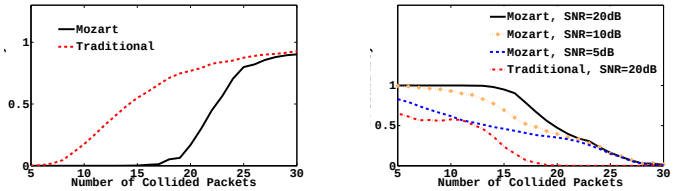


Fig. 3. Experiment Results with single AP.



(a) Experiment Topology (b) CDF of throughput

Fig. 4. Experiment topology and results with multiple APs.



(a) (b)

Fig. 5. Detection and RSS Estimation of colliding packets for Mozart and traditional approach [16] under equal RSS setting (worst case analysis).

described in Section III-F. In this section, we also measure the accuracy of transmitter identification and RSS estimation for the proposed algorithm (Sec. III-A) as well as the existing state of the art technique. The frequency used was 1078 MHz and the physical layer data rate was set to 62.5 Kbps. The rate was kept low to ensure that the delay between the host computer and the radio was *small* compared to the packet duration. Thus, the relative delay values would approximately match the delay values from off-the-shelf wireless cards.

In Mozart's implementation, the receiver stores all the received samples offline which are later used to compute the number of successful transmissions and the throughput. Besides Mozart, we also implemented a version of the IEEE 802.11 protocol. One of the challenges in implementing 802.11 was that USRP has different hardware parameters compared to the commercial 802.11 cards. For example, due to higher latency from the radio to the host computer, the packet decoding time was observed to be around 150 μs which prevents the receiver from sending an ACK within the SIFS duration. So, using experiments, we re-measured the optimum values of all 802.11 parameters (SIFS, DIFS, slot size, ACK timeout) for the N210 hardware as per their definitions. For example, we ensured that slot size was such that if two neighboring nodes choose consecutive values of backoff, then one of the nodes will sense the other's transmission and will not send its own packet. Due to the aforementioned reasons, the packet size was kept constant, and thus the receivers had no opportunity to use SIC to do simultaneous decoding.

A. Testbed Results

Single AP: In this experiment, we use a single N210 node as the AP and vary the number of clients. We place the client in various positions around the AP to create two types of topologies: (i) **Single collision domain:** Where all clients can hear each other; and, (ii) **Hidden Terminal:** Where no two clients can hear each other. The throughput results are shown in Figure 3 with varying number of clients. Mozart increases throughput due to fewer collisions and close to zero backoff as compared to the IEEE 802.11 protocol. When the nodes are hidden to each other, the throughput gain provided by Mozart is much higher due to increased collisions in 802.11. With 4 hidden nodes, Mozart provides 2.70x more throughput than IEEE 802.11 protocol.

Multiple APs: In the next experiment, we set up two N210 nodes as APs and four others as clients. For this experiment, we placed the two APs in two different rooms (as shown in Figure 4a). Each of the four clients were placed at different locations in the three regions so as to create ten different topologies (hidden, non-hidden, mix etc.). In all the topologies, the clients associated to the AP with the strongest signal. Figure 4b shows the CDF of the throughput of clients for both Mozart and 802.11. Averaging over all topologies, 802.11 provides throughput of 6.9 Kbps per node while Mozart (no SIC) provides 12.3 Kbps, an increase of 78%.

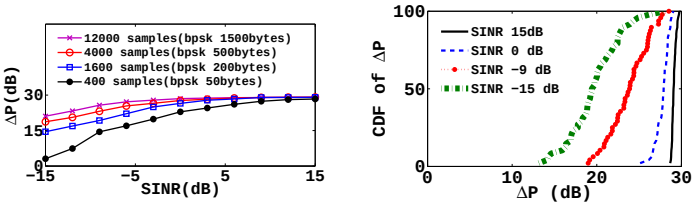
B. Experimental Analysis of Micro Benchmarks

In this section, we present micro benchmark results from our experiments. The computed micro benchmarks are also used as input to our evaluations (Sec. V).

1) Sender Identification and RSS Estimation: We measure the accuracy of sender identification through multiple experiments. For this (See Figure 5a), we ensured that all the packets have equal RSS (Equal RSS is the worst case for evaluations since *all* transmitters have low SINR in this case). As Figure 5a shows that when 20 packets collide, Mozart identifies 12 more senders compared to the existing approach[16]. The false positives were observed to be less than 1% for all the schemes.

Next, we studied the accuracy of RSS estimation of Mozart compared to existing techniques[16]. Figure 5b shows the probability that the estimated RSS of a packet is within 1dB difference of the actual RSS under different SNR values (computed without considering other packets as noise). When 15 packets (each having 20 dB SNR) collide, the probability that Mozart estimates RSS of the collided packets within 1dB of the actual RSS is 0.96 compared to 0.21 for existing techniques[16], an improvement by a factor of 3.57x.

2) Subtraction Accuracy: In this experiment, we measure the cancellation accuracy. To quantify the subtraction accuracy independent of modulation and coding, we define metric, ΔP that represents the power reduction achieved through cancellation. ΔP quantifies the achieved reduction in power after subtracting P from a set of collided packets. We define $\Delta P(in\ dB) =$ RSS of P (in dBm) $-$ Residual Power of P left after subtracting P (in dBm). A high value of ΔP implies that the residual noise is lower and it increases the decoding probability of the remaining packets.



(a) (b)

Fig. 6. Evaluation of packet subtraction: Higher ΔP implies lower residual noise and better cancellation accuracy. (a) Variation in ΔP with varying SINR of the subtracted packet. (b) CDF of ΔP when packet has 4000 samples.

Results: Figure 6 shows the variation in cancellation accuracy (ΔP metric) with variation in number of samples in the packet and the SINR of the packet being subtracted. The figure shows that for any packet consisting of more than 1600 samples, the cancellation is quite efficient. Figures 6b presents the Cumulative Distribution Function (CDF) of the distribution of ΔP for 4000 samples per packet. From Figure 6b, we see that as expected, the SINR for the subtracted packet decreases, the variation in cancellation accuracy (ΔP metric) increases. The results from our experiments are also fed in the ns-3 simulator as explained in the next section.

V. Comparison Results

To evaluate the performance of Mozart, we conducted extensive trace-driven ns-3 evaluations. In this section, we explain our evaluation setup and the results.

To make evaluations more realistic, we first setup a testbed of 40 nodes (comprised of laptops) and collected the RSS values between all pairs of nodes. The nodes were distributed over two floors of our building and spanned multiple rooms. This RSS data was then fed into the ns-3 simulator. We randomly designated varying number of the nodes as APs while the remaining nodes were designated as clients. Each client associated with the AP from which it received the strongest signal. In the evaluations, TCP connections were established between each client and its AP. For this, we downloaded the previously collected traffic traces during SIGCOMM[18] and computed the pdf distribution of packet sizes and also the pdf distribution of packet inter-arrival time over all connections. These two pdf distributions were used to generate both uplink and downlink traffic. To create network saturation condition, the number of connections between each client and its associated AP was set to 20.

Further, the results from the experiments (See Section IV) were fed into the ns-3 simulator as follows: (i) **PN-Sequence detection accuracy**: In the evaluations, we used the values from Figure 5 for determining if a PN-sequence can be detected or not; (ii) **Residual noise level**: The power of the residual noise was fed from the data collected from experiments (See Figure 6); and, (iii) **Imprecise Signal Strength Estimates**: For Mozart, imprecision in signal strength estimations could lead to unsuccessful decoding. We fed the imprecision from our experiments (See Figure 5b), into our evaluations.

Apart from Mozart, we also implemented and evaluated the following algorithms: (i) **Optimal omniscient zero-overhead slotted TDMA**: Here, we assume a central scheduler knows the interference between all links at all possible data rates. This scheduler computes the set of links that should be active in a given slot. For this, at the beginning of the slot, the APs first collect information about which node has data to send to which other node. The nodes convey this information using short PN sequences. The APs forward this information to a central scheduler that computes the set of active links and sends this information to APs. This information is then broadcasted back by the APs (again using short PN sequences). To put the optimal TDMA in the best light and to eliminate the effect of the backbone capacity, the APs and the central scheduler were connected through wired ethernet with unlimited bandwidth and zero latency. (ii) **802.11ec [16]**: 802.11ec reduces the overhead of RTS-CTS packets by encoding them as CSSs. (iii) **IEEE 802.11 without RTS-CTS**. In Mozart, transmitters picked the best data rate based on channel conditions as explained in Section III-B. Channel quality information was also provided to transmitters in Optimal TDMA out-of-band (*i.e.* without any overheads.). In all other algorithms, the transmitters varied their data rate using ARF algorithm [13].

A. Results for bidirectional TCP Traffic

Next, we evaluated the performance of different algorithms for TCP traffic. When Mozart is used for downlink traffic (AP to client), then the length of the recovery period was always 1 since the client would receive packet from at most one AP.

1) Throughput: We vary the number of APs in the network and compute the total throughput over all nodes (Fig. 7a). On average, Mozart provides 4%, 155% and 310% more throughput than TDMA, 802.11ec and 802.11 algorithms, respectively. These differences are primarily because of three factors:

1. **Collisions**: From Fig. 7c, we see that in other algorithms the percentage of transmissions that are acknowledged is significantly lower resulting in retransmissions and lower throughput. In Mozart, some of the transmissions may not be decoded by the receiver due to error in correlation of the PN-sequence or because of higher residual noise. However, when the number of colliding transmissions is low, Mozart decodes all of the transmissions resulting in $\geq$ 95% acknowledgment rate. The high acknowledgment rate also implies that the decoding accuracy of Mozart exceeds 95%. This can be attributed to high accuracy of transmitter identification, RSS estimation and careful selection of the transmitters to be suppressed (Sec. III-C). Although, 802.11ec reduces the overhead of control packets, still its critical period (See Section III-D) is sufficiently large leading to high collision rate. For 802.11 and 802.11ec, with increase in number of APs, the higher SINR of different links results in higher acknowledgment rate.
2. **Backoff**: Figure 7d shows the total time spend by nodes in backoffs between every successful transmission (averaged over all successful transmissions). Observe that transmitters in other algorithms spend a significant amount of time in backoffs. In IEEE 802.11 and its derived protocols, nodes decrement their backoff counter only if the channel is idle, implying that the channel resource is wasted in these protocols due to high backoff. On the other hand, nodes do not experience backoffs during the recovery phase in Mozart as discussed in Section III-D.
3. **SIC Applicability**: In our evaluations, we observed that on average, Mozart was able to apply SIC in 10.25% of the

slots. This allows Mozart to have a higher throughput than Optimal Omniscient TDMA since the TDMA scheduler uses fixed slot lengths, irrespective of the amount of the data to be transmitted by the user. Although, receivers in Mozart are also unaware of the amount of pending data, still the SPS-based decoding enables them to leverage SIC. This allows the Mozart receivers to decode multiple packets simultaneously (Sec. III-B), resulting in higher throughput.

2) Fairness: We use Jain's Fairness Index to compare the fairness for different protocols. Fig. 7b shows that with fewer APs, Mozart provides higher fairness compared to 802.11 and 802.11ec. This long term starvation in IEEE 802.11 and its derived protocols is consistent with the previous literature [9]. With an increase in the number of APs, the fairness index for all algorithms is almost equal due to higher SNR of the links.

Other Results including energy consumption and latency: Apart from this, we also evaluated performance of different algorithms for UDP VoIP traffic using different metrices (e.g. energy consumption, latency). Due to lack of space, these results are included in our technical report [4].

VI. Discussion

Here, we discuss how Mozart can be extended to make it more suitable for real networks.

Co-existence with legacy 802.11 devices: Mozart uses the NAV feature of IEEE 802.11 to ensure coexistence with legacy 802.11 devices. Upon overhearing a MAC packet, the node reads the "Duration" field of the MAC header and does not initiate any transmissions for that duration. To ensure coexistence with 802.11, packets in Mozart can be modified as follows: (i) Before transmitting poll or suppress PN sequences, the receivers transmit a zero-payload packet with duration field equal to the duration of one slot; and, (ii) Similarly, transmitters also transmit a zero-payload packet before transmitting the data packet. These packets will prevent 802.11 from interfering with Mozart. To give a fair chance to 802.11, receivers in Mozart also need to undergo certain amount of backoff before transmitting. We leave the detailed methodologies of coexistence that ensure fairness for our future work.

Handling interference from partially overlapped channels: Partially overlapping channels in WiFi can lead to loss of throughput as the data packets may not be decodable due to interference from devices operating on partially overlapped channel. To handle such interference, the receivers in Mozart can transmit the poll packet at reduced power level. This will force the transmitters to use data rates lower than optimal, thereby enabling successful decoding of packet even in the presence of interference from partially overlapping channels. The amount by which the receiver reduces its power level depends on the interference it experiences from partial overlapping channels. We leave the detailed discussion and analysis of Mozart's performance in presence of such interference for future work. Further, the receivers can also employ the approaches mentioned before (Sec. III-E) to handle decoding errors.

Compatibility with MIMO nodes: Mozart is compatible with wireless nodes with multiple antennas. If a node has N antenna, then Mozart allows such nodes to receive and decode N packets per recovery slot. To enable this, the receivers in Mozart would suppress N packets in each slot, thus reducing the length of the recovery period by a factor of N.

VII. Related Work

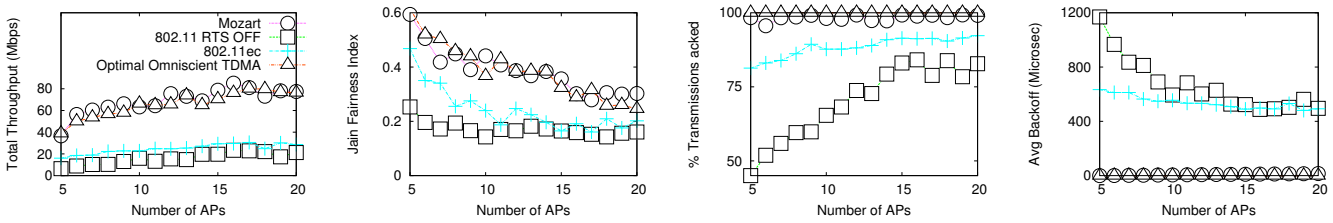
Collisions are known problems for wireless networks. Currently, carrier sensing (CSMA) is used in WLANs (Wireless LAN Networks) to avoid collisions. To further reduce the collision probability, various other schemes have been introduced such as RTS-CTS. However, transmitting RTS-CTS control packets at low physical layer data rate leads to significant overheads that become worse at higher data rates such as those observed in 802.11g or 802.11n networks. Further, RTS-CTS packets do not prevent collisions when interference range is higher than the transmission range even though they cause unnecessary blocking of transmissions [17].

Recently, Magistretti *et. al.* have proposed 802.11ec [16] that employs Correlatable Symbol Sequences (CSSs) to replace the RTS-CTS packets, thereby significantly reducing the overhead of control packets. However, the transmissions of nodes may still end up colliding due to the longer critical window of duration 39.4μs (discussed in Section III-D) resulting in backoffs and unnecessary retransmissions. When the density of the clients increases, more collisions will happen because of increase in simultaneous transmissions, resulting in further loss of throughput. On the other hand, with a shorter critical period of 2μs, Mozart has fewer collisions.

Apart from preventing collisions, other protocols have been proposed that either try to abort the collided transmissions or salvage the bits that did not undergo a collision. CSMA-CN [19] proposes to use two antennas at the transmitter for receiving the collision notifications from the receiver. Partial Packet Recovery [11] tries to recover bits that did not undergo collision. However, both CSMA-CN and PPR will abandon the collided samples, resulting in loss of throughput.

ZigZag [6] tries to utilize the collided information by employing ZigZag decoding. This reduces the number of useless retransmissions but the nodes still waste time in backoffs. Further, in ZigZag collided packets will be wasted if they are intended for different destinations since one of the transmitter may receive ack from its intended receiver and may not retransmit its data packet. In CRMA [15], nodes transmit the same information on different sub-channels. The receiver decodes the collided transmissions by solving a set of linear equations. However, for optimal channel utilization, nodes in CRMA require good estimation about the network load at other transmitters as well.

Mozart is a receiver-driven cross-layer algorithm that takes a different approach where it encourages multiple transmitting nodes to collide. Receiver-driven protocols have been proposed before in context of wireless sensor networks [23]. However, there the main objective was to reduce the energy consumption instead of maximizing the throughput. Recently proposed, AutoMAC [8] also encourages collisions among transmissions. However, AutoMAC implicitly assumes that all nodes are in a single collision domain. All its analysis, experiments and traces are also for single collision domain. In multi-collision domain networks, the receivers may face interference from transmitters that are transmitting to some other receiver and thus, can't be suppressed using Speculative Ack [8]. It is



(a) Throughput (b) Jain's Fairness Index (c) Percent transmissions acked (d) Backoff

Fig. 7. Comparison of different algorithms for TCP traffic.

possible that if two AutoMAC receivers are in the range of two transmitters such that one receiver can suppress only one transmitter, then the decoding may take a very large number of slots. This also makes extending AutoMAC to multi-collision domain networks non-trivial. Secondly, receivers in AutoMAC estimate the channel state by requiring transmitters to send non-overlapping training symbols one-by-one. On the other hand, Mozart's receivers can estimate the channel state of multiple colliding transmitters resulting in lower overhead.

Successive Interference Cancellation (SIC) [21] has been used before to decode interfering packets in WLANs [6]. Mozart uses SIC only when at least one transmitter has reduced its physical layer data rate to fit the slot width. Thus, Mozart is able to derive benefit from SIC and is not in conflict with existing literature [21].

Symphony [3], an extension of Mozart, has been proposed recently. Symphony works *only* in enterprise WLANs where APs are connected through ethernet backbone. In Symphony, APs may receive multiple colliding packets. To identify the set of transmitters and to estimate their RSS, Symphony makes use of algorithms (PN sequence padding, cancellation, iterative estimation) proposed in Mozart. Further, in contrast to Symphony, Mozart does not require APs to be connected with each other.

VIII. Conclusions

In this paper, we presented Mozart, a cross-layer algorithm that takes a new approach of encouraging collisions among nodes. By doing so, Mozart handles the hidden terminal collisions as well as reduces the critical period of transmissions. To implement Mozart in practice, we presented novel algorithms for identifying multiple transmitters as well as estimating their RSS in presence of interference. USRP-testbed based evaluations show that, Mozart throughput is up to 3.70x compared to IEEE 802.11. Evaluations performed using real-world traces show that Mozart's throughput is 2.55x and 4.10x when compared to 802.11ec and 802.11, respectively.

References

[1] "Ettus Research," http://www.ettus.com/.

[2] A. Bachir and *et al*, "Hidden Nodes Avoidance in Wireless Sensor Networks," in *Proc. of IEEE WirelessCom*, 2005.

[3] T. Bansal, B. Chen, P. Sinha, and K. Srinivasan, "Symphony: Cooperative Packet Recovery over the Wired Backbone in Enterprise WLANs," in *Proc. of ACM MOBICOM (Accepted)*, 2013, available at http://www.cse.ohio-state.edu/~bansal/symphony.pdf.

[4] T. Bansal, B. Chen, K. Srinivasan, and P. Sinha, "Mozart: Orchestrating Collisions in Wireless Networks," Tech. Rep., http://www.cse.ohio-state.edu/~bansal/mozartTechRep.pdf.

[5] R. Gold, "Optimal binary sequences for spread spectrum multiplexing (Corresp.)," *Information Theory, IEEE Transactions on*, vol. 13, no. 4, pp. 619 –621, 1967.

[6] S. Gollakota and D. Katabi, "Zigzag Decoding: Combating Hidden Terminals in Wireless Networks," in *Proc. ACM SIGCOMM 2012*.

[7] S. Gollakota, S. D. Perli, and D. Katabi, "Interference Alignment and Cancellation," in *Proc. ACM SIGCOMM 2009*.

[8] A. Gudipati, S. Perreira, and S. Katti, "AutoMAC: Rateless Wireless Concurrent Medium Access," in *Proc. of ACM MOBICOM*, 2012.

[9] C. Hua and R. Zheng, "Starvation Modeling and Identification in Dense 802.11 Wireless Community Networks," in *In Proc. IEEE INFOCOM*, 2008.

[10] *IEEE Std 802.11-2007 - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.* , IEEE, 2007, http://standards.ieee.org/about/get/802/802.11.html.

[11] K. Jamieson and H. Balakrishnan, "PPR: Partial Packet Recovery for Wireless Networks," in *Proc. of ACM SIGCOMM*, 2007.

[12] A. P. Jardosh, K. N. Ramachandran, K. C. Almeroth, and E. M. Belding-Royer, "Understanding Congestion in IEEE 802.11b Wireless Networks," in *Proc. of Internet Measurment Conference*, 2005.

[13] A. Kamerman and L. Monteban, "WaveLAN-II: A High-Performance Wireless LAN for the Unlicensed Band," *Bell Lab Technical Journal*, pp. 118–133, Summer 1997.

[14] S. Katti, S. Gollakota, and D. Katabi, "Embracing Wireless Interference: Analog Network Coding," in *Proc. of ACM SIGCOMM*, 2007.

[15] T. Li and *et al*, "CRMA: Collision-Resistant Multiple Access," in *Proc. of ACM Mobicom*, 2011.

[16] E. Magistretti, O. Gurewitz, and E. Knightly, "802.11 ec: Collision Avoidance Without Control Messages," in *Proc. of ACM MOBICOM*, 2012.

[17] A. Rahman and P. Gburzynski, "Hidden Problems With the Hidden Node Problem," in *In Proc. IEEE Biennial Symposium on Communications, 2006*.

[18] A. Schulman, D. Levin, and N. Spring, "CRAWDAD data set umd/sigcomm2008," Downloaded from http://crawdad.cs.dartmouth.edu/umd/sigcomm2008.

[19] S. Sen, R. Roy Choudhury, and S. Nelakuditi, "CSMA/CN: Carrier Sense Multiple Access with Collision Notification," in *Proc. of ACM Mobicom 2010*.

[20] S. Sen, R. R. Choudhury, and S. Nelakuditi, "No Time to Countdown: Migrating Backoff to the Frequency Domain," in *MOBICOM*, 2011.

[21] S. Sen, N. Santhapuri, R. R. Choudhury, and S. Nelakuditi, "Successive Interference Cancellation: Carving Out MAC Layer Opportunities," *IEEE Trans. Mob. Comput.*, vol. 12, no. 2, pp. 346–357, 2013.

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

[23] Y. Sun, O. Gurewitz, and D. B. Johnson, "RI-MAC: A Receiver-Initiated Asynchronous Duty Cycle MAC Protocol for Dynamic Traffic Loads in WSNs," in *Proc. of ACM SenSys*, 2008.