Improving Loss Resilience with Multi-Radio Diversity in Wireless Networks

Allen Miu, Hari Balakrishnan, and Can Emre Koksal*
MIT Computer Science and Artificial Intelligence Laboratory
The Stata Center, 32 Vassar Street
Cambridge, MA 02139
{aklmiu, hari}@csail.mit.edu, emre.koksal@epfl.ch

This paper describes the *Multi-Radio Diversity* (MRD) wireless system, which uses path diversity to improve loss resilience in wireless local area networks (WLANs). MRD coordinates wireless receptions among multiple radios to improve loss resilience in the face of path-dependent frame corruption over the radio. MRD incorporates two techniques to recover from bit errors and lower the loss rates observed by higher layers, without consuming much extra bandwidth. The first technique is *frame combining*, in which multiple, possibly erroneous, copies of a given frame are combined together in an attempt to recover the frame without retransmission. The second technique is a low-overhead retransmission scheme called *request-for-acknowledgment (RFA)*, which operates above the link layer and below the network layer to attempt to recover from frame combining failures. We present an analysis that determines how the parameters for these algorithms should be chosen.

We have designed and implemented MRD as a fully functional WLAN infrastructure based on 802.11a. In our testbed, we measured throughput gains up to 2.3× over single radio communication schemes employing 802.11's autorate adaptation scheme.

Categories and Subject Descriptors

Computer Systems Organization [**Computer-Communications Networks**]: Network Architecture and Design—*Wireless Communication*

General Terms

Measurement, Performance, Design, Experimentation

*Computer and Communication Sciences, EPFL, Switzerland.

MobiCom'05, August 28–September 2, 2005, Cologne, Germany.

Keywords

path diversity, packet combining, frame combining, bit-error, wireless LAN, wireless networks, performance

1. INTRODUCTION

This paper describes the design and implementation of the *Multi-Radio Diversity (MRD)* system, which reduces the loss rate and improves the throughput observed by transport protocols and applications running over wireless local area networks (WLANs). Our approach uses *path diversity*, relying on multiple access points (APs) covering a given area (for uplink diversity) and multiple radios on the user's device (for downlink diversity). The hypothesis underlying this system is as follows: because frame losses are often path-dependent (e.g., due to multi-path fading), location-dependent (e.g., due to noise), and statistically independent between different receiving radios, multiple radios that all receive versions of the same transmission may together be able to correctly recover a frame, even when any given individual radio is not.

Most current WLAN deployments (e.g., those based on 802.11 [6]) use one or more APs that relay packets to and from a WLAN client. Each AP operates independently and each WLAN client can communicate with only one AP at a time. Because the properties of a single path vary with time and can undergo severe deterioration, the result is that communication often suffers from high packet loss rates, long delays, and even outages. These, in turn, degrade the performance of protocols like TCP and applications like mobile Internet telephony, streaming audio/video, and games.

In MRD, different APs with overlapping coverage and listening on the same radio frequency provide alternate communication paths for each frame transmission from a given WLAN client, while multiple wireless cards on the WLAN client achieve the same result for transmissions to the client. MRD coordinates packet receptions across the different radios to improve loss resilience against path-dependent bit corruption. The idea is simple: even when each individual reception of a data frame is erroneous, it might be possible to combine the different versions to recover the correct version of the frame. In MRD, the entity that performs this *frame combining* task is the *MRD Combiner (MRDC)*.

MRD's frame combining algorithm divides each frame into blocks. For each block, the algorithm assumes that at least one of the received copies of a frame (including any possible retransmissions) contains the correct bit values for that block. The algorithm then attempts to reconstruct the correct frame by trying every version received for each block. The process succeeds if a particular block combination passes the checksum embedded in the data frame, and fails once the search exhausts all possible block choices for each block. The computational complexity of this algorithm is exponential in the number of blocks for which different versions were received, which depends on the number of blocks in each frame. We show how to pick the block size and evaluate its performance using theoretical analysis and real-world experiments. This approach to frame combining is reminiscent of an old, well-studied idea called "retransmissions with memory" [31, 13], where retransmissions of erroneous frames are combined with the original transmission in an attempt to recover the correct version of the data. Our contribution is to generalize this idea using a block-based technique to incorporate the spatial dimension as well.

The MRDC can often recover a corrupt frame without requiring a retransmission from the client, but frame combining will not always succeed. MRD uses a lightweight retransmission scheme running above the WLAN link layer to further improve error recovery. At the sender, the *MRD Sender (MRDS)* buffers all frames that have not yet been acknowledged (or given up on), and retransmits any frame that it believes has not been successfully received by the MRDC (after frame combining). To prevent adverse interactions caused by ARQ schemes at two different layers, MRD turns off link-layer retransmissions altogether. To keep overhead low and to react quickly to channel contention, however, MRD uses two techniques: first, it retains 802.11's synchronous ACK mechanism, with the MRDS clearing frames thus acknowledged from its retransmission buffer. But because some frames can only be recovered after frame combining, and because the MRDC does not know whether any given link-layer ACK reached the MRDS, MRD uses a feedback protocol between the MRDC and MRDS. This protocol is designed to have low overhead, using a *request for ACK (RFA)* technique rather than traditional ACKs or NACKs. With RFA, the MRDS explicitly requests an ACK from the MRDC for certain frames, and decides whether and when to retransmit frames based on this feedback.

A noteworthy aspect of MRD is that it achieves significant improvements in loss rates while consuming only a small amount of additional bandwidth. As a result, it complements both automatic repeat request (ARQ) and rate adaptation [1, 10, 20], two common error-control techniques used in contemporary WLANs. ARQ-based retransmissions work well when the duration of channel degradation is short. But when the channel's quality deteriorates for a long period, link-layer retransmissions triggered by a missing link-layer ACK become ineffective and wasteful. Rate adaptation, on the other hand, can work well even when the wireless channel experiences severe deterioration. However, efficient rate adaptation is difficult to achieve when channel conditions vary quickly and unpredictably, as is the case in many real-world WLANs, particularly when users are mobile.

Sections 2 through 5 of this paper detail the different contributions of this paper: the MRD architecture, the frame combining algorithm and its theoretical analysis, the RFA scheme, and the MRD modifications to the 802.11 WLAN rate adaptation schemes. Section 6 describes our fully functional 802.11a/b/g-based Linux implementation of MRD. Section 7 presents the results of several experiments conducted over an in-building 802.11a-based testbed at MIT's Computer Science and Artificial Intelligence Laboratory. Experiments that experienced a high channel variability (e.g., a mobile WLAN client that moved over a relatively small area of about three square meters) show throughput improvements of up to 2.3× compared to contemporary 802.11a with "autorate adaptation" [1].

2. MULTI-RADIO DIVERSITY ARCHITECTURE

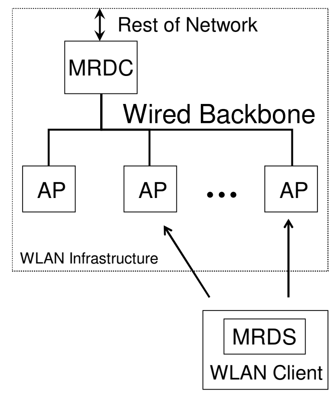


Figure 1: MRD system architecture.

For ease of exposition, we describe the MRD architecture in the context of uplink transmissions from the client to the WLAN infrastructure. The same architecture can be used when the MRD radios are co-located on the same device (either in a single AP or on the WLAN client).

Figure 1 shows the MRD system architecture. Each AP in the WLAN infrastructure offers a different physical communication path to the client. We configure the APs to listen on the same radio frequency so they can each receive a copy of the client's uplink transmission. The AP forwards all frames—including those that are corrupted—to the MRD Combiner (MRDC), which filters redundant data frames received by multiple radios and invokes the frame combining procedure when needed. The MRDC maintains a packet buffer to in-order packet delivery to the rest of the network.

At the WLAN client sender, the MRDS handles data transmissions and retransmissions. The MRDS operates in between the link-layer and the IP network layer. It keeps track of unacknowledged transmissions and schedules their retransmissions when it believes that the MRDC has failed to receive a clean copy of the transmitted frame from any of the APs or has failed to correct their errors frame combining. The MRDS uses the request-for-ACK (RFA) protocol to obtain the results of the frame combining procedure from the MRDC.

The MRD WLAN architecture does not preclude cellular frequency reuse. Frequency reuse is a common method to increase network capacity, which requires APs in neighboring cells to operate in different radio frequencies. In MRD, the APs that are not explicitly associated with the client need only listen for uplink transmissions *passively*. Thus, one strategy to achieve frequency reuse is to install *passive* radios in addition to the regular, *active* radio at each AP.[1]

[1] In fact, companies have begun selling radio chipsets that

The client associates with the active radio at each AP, which serves the regular function of transmitting management and control frames to the WLAN client, while the passive radios are configured to listen on the neighbors' radio frequencies. Because the passive radios never transmit a frame, they do not create any interference in the network. If installing multiple radios on a single AP is not possible, an operator can install additional passive access points in the network. As the costs of APs continue to decline, this approach is a viable way to add path diversity (for uplink communication) in WLANs.

MRD assumes that there is sufficient bandwidth in the wired backbone to handle the additional traffic generated by the passive APs. This assumption is reasonable because the number of APs within reception range of a transmitter is usually low and the speed of the wired backbone is usually at least one or two orders of magnitude higher than the wireless link.

MRD does not affect the functions of handoff and security in a WLAN. The WLAN client would associate with and handoffs between different APs using their active radios. Existing WLAN security standards such as WEP [6], 802.1x [7], and WPA/802.11i [4] handle encryption/decryption and other security functions in software and are easily implemented in the MRDS and the MRDC, assuming that the MRDC can establish a secure trust relationship with each MRD radio in the network.

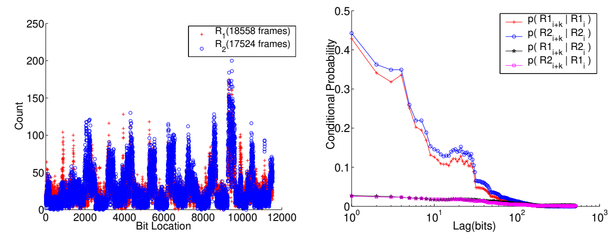
3. FRAME COMBINING

We describe how MRD recovers error-free versions of corrupted data frames and analyze its performance. One approach is to run a simple linear time algorithm that attempts to correct bit errors by selecting the majority bit value between three or more frames [14]. But this approach requires at least three copies of the same transmitted frame, which may not be available (without a retransmission) in the case when only two MRD radios are within receiving range of the sender. Therefore, we develop and analyze a block-based frame combining scheme that can work even when only two copies are available.

Suppose two copies of the same transmitted frame of size S bits are received at two different receivers. Before frame combining, if any of the data frames passes the link-layer cyclic redundancy checksum (CRC) check, it is decoded as the transmitted frame and forwarded (*soft selection*). Otherwise, we run the block-based combining algorithm to recover the packet. Block-based frame combining works by first subdividing both frames into blocks, and then reconstructing the frame by assembling the blocks selected from each received frame of the transmitted packet. The process succeeds if a block combination passes the CRC embedded in the data frame, and fails once the search exhausts all possible block combinations. We provide a summary of the block combining algorithm below:

1. The input of the algorithm is two frames $f = \{\mathcal{A}, \mathcal{B}\}$ of size S, divided into fixed-sized blocks $X = \{X_1^f, X_2^f, ..., X_{N_B}^f\}$. Let $\Delta = |\{i | X_i^{\mathcal{A}} \oplus X_i^{\mathcal{B}} \neq 0\}|$ (i.e., the number of blocks that do not have matching bit values).

can process and decode transmissions from multiple channels simultaneously (see, e.g., [3]).



(a) Bit-error location histogram

(b) Bit-error conditional probabilities

Figure 2: Bit-error analysis. Figure 2(a) shows that the bit-errors are clustered in a regular pattern within a frame. The number in the legend indicates the number of corrupt frames received at each node. The conditional probabilities in Figure 2(b) suggest that bit-errors occur in bursts within a frame but bit-errors between frames received at different locations have low correlation.

2. Assemble a combined frame that contains $X' = \{X_1^{f'}, X_2^{f'}, ..., X_{N_B}^{f'}\}$ blocks from either frame $\mathcal{A}$ or $\mathcal{B}$. Each iteration of this step generates a new combined frame by replacing $X_i^{f'}$ with either $X_i^{\mathcal{A}}$ or $X_i^{\mathcal{B}}$ for each i where $X_i^{\mathcal{A}} \oplus X_i^{\mathcal{B}} \neq 0$.

3. If either of the CRC value embedded in frames $\mathcal{A}$ or $\mathcal{B}$ matches the CRC value computed over X', return the combined frame containing X'. Otherwise, repeat step 2 until all possible combinations of X' have been tried. If none of the block combinations X' passes the CRC check, declare a frame combining failure.

There are many ways of dividing a frame into roughly fixed-size blocks. For simplicity, we divide each frame such that blocks $X_1^f, X_2^f, ..., X_{N_B-1}^f$ contain B bits and the size of the last block $|X_{N_B}^f|$ is $\leqslant B$. Thus, $N_B = \lceil S/B \rceil$.

When the block-based frame combining algorithm declares a failure, the MRDC can save the corrupt frames for possible frame combining (using either bit-majority or block-based combining) with any subsequent retransmissions of the frame. In our current implementation, the MRDC saves only one of the corrupt frames and apply block-based combining to two corrupt frames at a time.

The block-based frame combining algorithm is simple but its running time is exponential in Δ, the number of differing blocks. With two copies, it needs up to about 2^{Δ} CRC check operations to identify the correct combination. Since $\Delta \leq N_B$, one way to bound the number of CRC checks is to reduce N_B by increasing B. Inevitably, the frame combining failure probability will increase as the likelihood of simultaneous block errors increases with B. We evaluate this tradeoff next.

3.1 Frame Combining Failure

We analyze how the frame combining failure probability, p_f, varies with N_B under a burst bit-error channel model parameterized by a burst length b. p_f is the fraction of frames that cannot be corrected with combining out of those that could not be corrected by the soft selection in the first

$FLR(R_1)$	26.5%
$FLR(R_2)$	23.4%
$FLR(R_1)FLR(R_2)$	6.21%
$FLR(R_1 \cap R_2)$	7.09%

Table 1: Frame loss rates (FLR) observed at two receivers (Figure 11) in a broadcast experiment.

place. To find the overall retransmission probability we assume that each receiver observes independent losses, and multiply p_f with the independent frame loss rates (FLR) at each receiver $FLR(R_1) \times FLR(R_2)$ (i.e., the probability that the frame goes uncorrected by soft selection).

We observe that losses do occur independently at different receivers in practice. Table 1 shows the frame loss statistics of a broadcast experiment with three 802.11a nodes illustrated in Figure 11. In the experiment, a total of 500,000 frames were transmitted at a bit rate of 48 Mbps. We use $FLR(R_1 \cap R_2)$ to represent the number of broadcast transmissions that were lost simultaneously at receivers R_1 and R_2. Our results indicate that $FLR(R_1)FLR(R_2) \approx FLR(R_1 \cap R_2)$, which suggest that losses are largely independent at each receiver.

Using the same experiments, we validate the assumption that bit-errors occur in bursts by analyzing the bit-error patterns of over 36,000 corrupt data frames. Figure 2(a) plots a histogram of the bit-error locations, which shows that the error distribution is uneven, often clustered within 300-400 bits, spaced between 800-1200 bit positions apart. At the 48 Mbps bit-rate, 802.11a employs QAM-64 modulation at 2/3 coding rate. This burst pattern is also observed in other node placements on our testbed and also in another 802.11b testbed deployed in an industrial environment [34].

Figure 2(b) shows the auto-conditional and cross-conditional bit-error probabilities for all the corrupt frames. The cross-conditional probabilities remain flat even at the bit level. The cross-conditional bit-error probabilities for $k < 100$ are much lower than their counterpart auto-conditional probabilities, which suggests that bit-errors rarely occur simultaneously at nearby locations between two frames received at different physical locations. In contrast, the auto-conditional error probability at the bit level increases dramatically at small k (< 100). The increased auto-conditional probability corresponds to the burst bit-error behavior and is most likely related to the clustered bit-error patterns shown in Figure 2(a).

We believe that the periodic and burst nature of bit-errors observed in our experiments is due to the orthogonal frequency division multiplexing (OFDM) scheme employed in 802.11a. In this scheme, 52 separate sub-carriers are used to provide separate wireless pathways for sending the information in parallel. Four of them are used for control, and each of the remaining 48 sub-channels carries upto 1 Mbps summing to 48 Mbps. We believe that the non-uniformity of the losses is because different parts of a frame are carried by different channels, and the periodicity of bit-errors arises because the same set of data bits in each frame are consistently assigned to the same sub-channel. Indeed, QAM-64 implies that there are 8 bits/symbol on each sub-carrier and hence the bunching of $8 \times 48 \approx 400$ bits is consistent with this hypothesis. Also, the 1,200-bit spacing of the peaks may be because each sub-channel contributes three symbols at a time rather than just one.

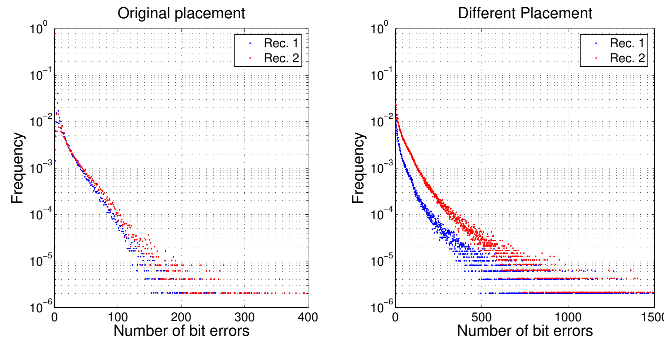


Figure 3: The PMF for the number of bit-errors for two different placements of the receiver pair.

These experimental observations motivated us to develop an analytic model that allows us to examine how p_f is affected by the bit-error burstiness in the communication channel. In our model, we assume that bit-errors occur in bursts of $b \geqslant 1$ bits. Moreover, we assume that these sequences of consecutive b bit-errors are spread uniformly over the frame. Thus, if there occur d such sequences in a given frame, then it means there are a total of bd bit-errors in that frame. We neglect the effect of two individual error sequences starting within b bits of each other.

Let $D_{b,i}$ represent the number of b-bit sequences with errors in a given frame received at receiver R_i. Then,

$$\mathrm{P}\left(D_{b,i} = d | D_{b,i} > 0\right) = \eta \sum_{d'=(d-1)b+1}^{db} \mathrm{P}\left(D_i = d'\right). \quad (1)$$

where $\eta = (1 - \mathrm{P}(D_i = 0))^{-1}$ and $\mathrm{P}(D_i = d')$ is the probability that a frame received by R_i contains d' bit-errors. We obtain the distribution of number of bit-errors empirically. Figure 3 shows the probability mass function of the number of bit-errors for two broadcast experiments using different node placements. We found that given a frame contains bit-errors, $\mathrm{P}(D_i = d')$ decays almost exponentially, i.e., as $e^{-\alpha d}$ where $\alpha \approx 0.01$—0.05.

In our model, we kept the average number of bit-errors per packet fixed (independent of b) and b controls only the burst size. This model of fixed sized bursts of error implies that the auto-conditional bit error probability distribution is a step function with a jump at b. Even though this model is approximate (as shown in Fig. 2(b)), it encompasses certain flavors of wireless channels where losses occur in bursts.

Let us denote the set of blocks with errors at receiver R_i by $\mathcal{N}_i$. Then $|\mathcal{N}_1 \cap \mathcal{N}_2|$ represents the set of blocks that contain simultaneous errors at both R_1 and R_2.

To derive the frame combining failure probability, p_f, we make the following simplifying assumptions. First, we ignore the possibility that two sequences at a given block of two different frames have exactly the same position. Second, we ignore the possibility that a sequence can spread over more than one block. Third, we assume the boundaries of the blocks are not fixed and that each one of them can hold more than B bit-errors whereas in reality each block can contain at most $\lceil B/b \rceil$ sequences of b bit error sequences. All these assumptions are reasonable when $b \ll B$, which is likely to be the case in reality.

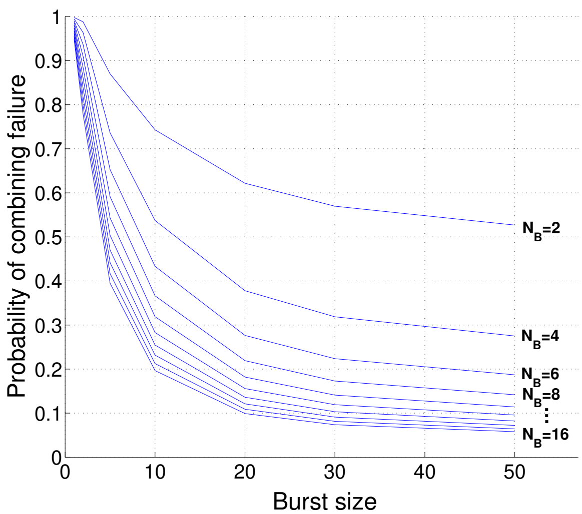


Figure 4: The upper bound on p_f as a function of the burst size, b. From top to bottom, the curves are plotted for the span of values of number of blocks, $N_B = 2, 4, 6, 8, 10, 12, 14$ and 16.

If the sequences of bit-errors are uniformly distributed over the frame, the probability of getting at least d simultaneous block errors, conditioned on the event that receiver R_i receives a frame with d_i trains of burst errors is at most

$$\mathrm{P}\left(|\mathcal{N}_1 \cap \mathcal{N}_2| \geqslant d | D_{b,1} = d_1, D_{b,2} = d_2\right) \leqslant \frac{\binom{N_B}{d}\binom{N_B+d_1-d-1}{d_1-d}\binom{N_B+d_2-d-1}{d_2-d}}{\binom{N_B+d_1-1}{d_1}\binom{N_B+d_2-1}{d_2}}. \quad (2)$$

for $d < \min\{d_1, d_2, N_B\}$. The analogy with a ball placement problem is as follows. We have d_1 red and d_2 blue balls to be placed in a total of N_B bins randomly. We evaluate the probability that at least d bins contain both red and blue balls. First, we place d red balls and d blue balls in a given combination of d bins so that each bin contains exactly one red and one blue ball. Then we distribute the remaining $d_1 - d$ red and $d_2 - d$ blue balls randomly in all possible N_B bins. We end up with an upper bound because we count certain combinations more than once.

Because a frame combining failure occurs when $d \geqslant 1$, the conditional frame combining failure probability is simply

$$p_f(d_1, d_2) = \mathrm{P}\left(|\mathcal{N}_1 \cap \mathcal{N}_2| \geqslant 1 | D_{b,1} = d_1, D_{b,2} = d_2\right). \quad (3)$$

Hence, the upper bound on the unconditional probability of combining failure can be found plugging (2) in

$$p_f \leqslant \sum_{d_1=1}^{N_B} \sum_{d_2=1}^{N_B} p_f(d_1, d_2) \prod_{i=1}^{2} \mathrm{P}\left(D_{b,i} = d_i | D_{b,i} > 0\right). \quad (4)$$

Figure 4 plots the upper bound on p_f as a function of the burst size b for several values of the block size, N_B. If the bit-errors are uniform ($b = 1$), p_f remains high (≈ 1) regardless of N_B. However, the auto-conditional probabilities in Figure 2(b) suggests that bit-errors indeed occur in bursts. In this case, we expect p_f to decrease with increasing N_B. As N_B gets larger, the difference between the two curves for a given b becomes very small, which suggests that increasing N_B beyond a certain point does not yield much improvement. Thus, we lose little performance by fixing N_B to some small value (say, 6-10) in order to bound complexity. Because p_f is a highly convex function of b, we expect the performance of frame combining to be sensitive with respect to the changes in the burstiness of the bit-errors in the channel. Moreover, the performance of frame combining will improve as the available computational power increases.

3.2 False Positives

We now comment on the possibility of false positives in the combining process caused by repeated trials for the CRC to check with distinct frames. In essence, CRC is an n-bit parity check field that detects any $k < n$ bit errors and misses detection with probability 2^{-n} when $k \geqslant n$. Thus, if a 32-bit CRC is used, as in 802.11, any number of bit errors < 32 is detected. Moreover, the probability that any randomly produced frame will check the CRC is 2^{-32}, which implies that it is almost impossible for a random bit error pattern to go undetected even if a frame contains more than 31 erroneous bits.

Now, with frame combining, even though a single check leading to a false positive is highly unlikely, if we try it repeatedly many times, we may end up getting a false positive. Indeed, if the number of differing blocks in two frames is Δ, the number of swaps (and the number of tests for the CRC to check) is 2^{Δ}. For independently produced 2^{Δ} frames, the false positive probability is

$$\mathrm{P}\left(\text{false positive}\right) = 1 - \left(1 - 2^{-32}\right)^{2^{\Delta}} \approx 1 - \exp\left(-2^{\Delta-32}\right).$$

Thus, if $\mathrm{E}\left[2^{\Delta}\right]$ is close to 2^{32}, it is likely that the combining procedure leads to false positives. Even if the available computational power can perform 2^{32} CRC tests, we pick a block size that is sufficiently large (i.e., N_B is sufficiently small) so that, even in the worst case, we do not perform too many CRC checks. Hence, we guarantee by design that $2^{\Delta} \ll 2^{32}$ and keep the false positive probability sufficiently small. Our implementation uses $N_B = 6$.

4. RETRANSMISSIONS WITH RFA

MRD disables link-layer retransmissions to allow the MRDC to recover packets that the active radios receive in error. The MRDS retransmits frames that the MRDC fails to recover with soft selection or frame combining (i.e., a frame recovery failure). To facilitate these retransmissions, the MRDS uses the *request-for-acknowledgment* (RFA) protocol to obtain the status (success or failure) of each frame transmission. This section describes the design of RFA.

4.1 Design

RFA operates in between the link-layer and the network layer, but uses the link-layer synchronous ACKs that is implemented in most WLANs such as 802.11. A synchronous ACK is a link-layer control packet that is sent by the active radio (see Section 2) immediately after it successfully receives a data frame. After each frame transmission, the MRDS checks the link-layer transmission status. A success implies that the active radio has received the transmission correctly, so the MRDS can proceed to transmit the next available packet. A failure implies either a corrupt link-layer ACK or a corrupt data transmission. In the former case, the MRDC simply forwards the correctly received data packet or buffers it in the reorder buffer (explained below in Section 4.3). In the latter case, the MRDC may recover the frame loss using soft selection or frame combining. If the recovery is unsuccessful, the MRDC saves the corrupt frames for possible frame combining with any subsequent retransmissions of the frame.

In either case, the MRDC always knows the final status of each frame transmission. Thus, when the MRDS fails to receive a link-layer ACK, it issues a "request-for-ACK" frame to the MRDC to obtain an *MRD acknowledgment* (MRD-ACK), which contains the authoritative status of the transmission. The MRDS needs to explicitly issue an RFA because only the MRDS knows which frames are ACKed by the link-layer. To save overhead, the MRDS signals a RFA by setting a flag in the frame header of subsequent data transmissions. We explain the implementation details of RFA in Section 6.2.

The MRDS buffers the frame that fails to receive a link-layer ACK for later retransmission and schedules the next available frame for transmission. The subsequent transmissions keep the wireless channel utilized while the MRDS waits for the frame recovery results from the MRDC, which can take many milliseconds. To limit the size of the retransmission buffer, the MRDS may transmit up to N different frames from the first unacknowledged one. A frame is removed from the transmission buffer after K unsuccessful retransmission attempts. The MRDS schedules a retransmission if the MRD-ACK indicates a frame recovery failure. If the MRDS never receives an ACK from the MRDC, the MRDS will schedule all outstanding unacknowledged packets for retransmission after a timeout, T_s. Our current implementation uses a static value of 90 ms.

There are two reasons why we chose to use the link-layer ACK, instead of eliminating it and letting MRDS and MRDC handle retransmissions using a standard automatic repeat request (ARQ) protocol that operates strictly above the link-layer. First, the synchronous ACKs are necessary for carrier-sense multiple access (CSMA) to operate properly. CSMA uses a randomized backoff window and relies on the absence of the synchronous ACK packet to detect contention and adjust the backoff window *after each frame transmission*. Because we allow transmissions to continue while the MRDS waits for an MRD-ACK from the MRDC, it is important to preserve the underlying CSMA channel access mechanism. [2]

Second, the wireless medium is already reserved for the transmission of synchronous ACKs. They are designed (by means of a smaller data-to-ACK frame spacing time) to not collide with transmissions from another nearby source. In contrast, the acknowledgments from the MRDC are asynchronous and must therefore contend for the channel and suffer potential collisions. Thus, it is a good idea to avoid sending asynchronous ACKs as much as possible, especially during times when the channel quality is good and link-layer losses are low.

[2]It is conceivable to use some other channel access schemes besides CSMA (e.g., TDMA). Doing that would require introducing a major modification to the medium access control (MAC) layer of 802.11.

4.2 Delaying Acknowledgments

To reduce the number of MRD-ACKs sent to the sender, the MRDC delays the return of an MRD-ACK frame by up to D frame-transmission times, where $0 < D < N$. D should be greater than 0 because the MRDC needs time to gather corrupt frame copies from the MRD radios and perform frame combining. A smaller D value would cause the system to incur higher overhead as the MRDC would send MRD-ACKs more often. A higher D reduces overhead, but can cause larger transmission delay when the frame requires retransmission. In practice, the added delay is of little concern to higher layer transport protocols and most multimedia applications because D is usually set to a few frame transmission times, on the order of a few milliseconds. If $D > 1$, the MRDC could process multiple frames before returning an MRD-ACK to the MRDS. We expand the MRD-ACK packet with a bit-vector to indicate the final status of several packets at once, instead of spreading the acknowledgment across several different MRD-ACK frames.

4.3 In-order packet delivery

The MRDC maintains a reorder buffer to ensure that packets are forwarded in-order to the rest of the network. When a frame requires retransmission, the MRDC inserts all subsequently transmitted frames into the reorder buffer until the missing frame has been successfully recovered or has been given up on.

There are many applications, such as audio and video streaming, which are sensitive to packet delays but do not require in-order packet delivery. To cater to these applications, we can mark specific frames for out-of-order delivery. Such frames can avoid being delayed inside the ordering buffer. Our current implementation does not include this feature but we plan to incorporate it in the future.

5. RATE ADAPTATION IN MRD

Rate adaptation (or "autorate") works well when the communication channel severely deteriorates and should be used in MRD when soft selection and frame combining can no longer recover frame losses effectively. Traditional autorate algorithms try to maximize throughput by using loss or signal strength information observed by a single receiver. Current autorate algorithms behave sub-optimally under MRD because they do not use information observed at *all* of the diversity radios that are within range of the sender.

The interaction between rate adaptation and MRD error control is an interesting open topic. Here, we present some simple modifications to an existing rate adaptation. Although these modifications may not necessarily yield an optimal algorithm for MRD, we found them to work well in our experiments.

Our testbed implementation is based on 802.11 interfaces that use the Atheros 5212 chipset, which are driven by the Multiband Atheros Driver[3] for WiFi (MADWiFi) [1]. The MADWiFi driver implements an autorate algorithm that adjusts bit-rates based on the observed link-layer frame loss rate. Due to the popularity of MADWiFi, the MADWiFi autorate algorithm is becoming a *de facto* benchmark. Its performance has been studied extensively in [10] and [20] and is shown to outperform the Auto Rate Fallback (ARF) algorithm that is implemented in many 802.11 interfaces on

[3]pci: v.0.8.6.1, hal: v.0.9.9.13, wlan: v.0.7.3.2

market. We use the MADWiFi autorate algorithm as the basis of discussion, but the general ideas in this section can be applied to many other loss-based autorate algorithms.

Figure 5 provides a pseudo-code of the MADWiFi autorate algorithm. In our notation, *bitrate* is an integer with a range [0..MAX_BITRATE], which represents the set of discrete bit-rates available to the sender. There eight discrete bit-rates in 802.11a [6, 9, 12, 18, 24, 36, 48, 54] Mbps.

```
Init()
    stable ← 0
    numtx ← 0
    numtxok ← 0

TxCallback()
    numtx ← numtx + 1
    if (txsuccess)
        numtxok ← numtxok + 1

RateAdjust()
    if ((numtx > 0 and numtxok == 0) or
        (numtx ⩾ 10 and numtxok/numtx < D))
        if (bitrate > 0)
            bitrate ← bitrate − 1
            Init()
    elseif (numtx ⩾ 10 and numtxok/numtx > 0.90)
        stable ← stable + 1
        if (stable ⩾ S and bitrate < MAX_BITRATE)
            bitrate ← bitrate + 1
            Init()
    else
        stable ← stable + 1
```

Figure 5: Pseudo-code of the MADWiFi autorate algorithm.

```
MRDCallback()
    numtxok ←
        numtxok + min(numacked, numtx − numtxok)
```

Figure 6: A procedure that helps autorate maintain a better estimate of *numtxok* in MRD.

The MADWiFi algorithm starts by calling Init() and invokes TxCallback() to update the *numtx* and *numtxok* counters after each frame transmission. The algorithm invokes RateAdjust() once every T seconds. If the frame delivery rate is above 90% for at least S number of successive periods, increase the bit-rate. If it falls below a minimum delivery threshold $\mathcal{D}$, decrease the bit-rate.

The original algorithm adjusts the *numtxok* counter based on link-layer feedback. This approach can lead to an understated *numtxok* value in MRD because the MRDC can recover many frame transmissions using soft selection or frame combining. To fix this problem, we add the routine listed in Figure 6 to the MADWiFi autorate algorithm.

The MRDS invokes MRDCallback() whenever it receives an MRD-ACK. *numacked* is the number of frames reported in the MRD-ACK that have a successful delivery status at the MRDC. and is added to *numtxok*. Thus, MRD-Callback helps the autorate algorithm maintain a correct estimate for *numtxok* as long as it receives some MRD-ACKs. Even if MRD-ACK$_i$ packet is dropped for some reason, the *numtxok* can still be adjusted to the correct value by the subsequent MRD-ACKs because the MRD-ACKs cumulate the ACK bit vector for any unacknowledged packet. But because *numtxok* can be adjusted only upon receiving a MRD-ACK$_i$ packet, long delays between MRD-ACK receptions can still cause understatement in the *numtxok* value. This is not usually a problem in practice because 1) MRD-ACKs are always transmitted at the lowest (most robust) bit-rate to minimize loss, and 2) we set a low delay threshold (16 ms in our implementation) for transmitting MRD-ACKs.

Scheme	Mean (Mbps)	Median (Mbps)
Slow R1	4.95	4.68
Fast R1	8.25	7.07
Slow MRD-R1	19.29	19.85
Fast MRD-R1	18.76	19.06

Table 2: The mean and median throughput of one second non-overlapping window samples across all five trials in each experiment.

Another problem with the original MADWiFi algorithm is that the default minimum delivery threshold $\mathcal{D}$ is fixed at 50%, which, as noted in [10], is inefficient for 802.11a/g. Let $\mathcal{D}_r$ and R_r be the expected delivery rate and effective throughput[4] using bit-rate r. Then, the throughput achieved by the lower bit-rate is the same as the current bit-rate if $\mathcal{D}_{r-1} \times R_{r-1} = \mathcal{D}_r \times R_r$.

R_{r-1} and R_r are known values and in general, $\mathcal{D}_{r-1} > \mathcal{D}_r$ because lower bit-rates are more robust against loss. To minimize loss, we set $\mathcal{D}_{r-1} = 1$. Thus, the ideal minimum delivery threshold for bit-rate r is $\mathcal{D}_r = R_{r-1}/R_r$, the ratio of the effective throughput at the lower and higher rates.

In 802.11a, the typical value for R_{r-1}/R_r varies from 0.6 to 0.8. Thus, fixing $\mathcal{D} = 0.5$ is too low and causes the transmitter to maintain the current bit-rate even though its delivery rate is well below the break even point. We modified the MADWiFi algorithm to lower bit-rates according to the proper break-even ratios in our implementation.

Finally, the default values for T and S ($T = 1$ second and $S = 10$) cause the MADWiFi algorithm react too slowly to rapid changes in the channel. Instead, we set lower values $T = 0.25$ and $S = 2$ to improve its responsiveness. We ran an experiment with a high channel variability (by using mobile transmitter, described in Section 7.1) to compare the performance of the algorithm using different parameter values. Table 2 shows that the modified parameter values (Fast) helped increase throughput by about 67% over the default parameter values (Slow) for the single radio experiments using R1.

Intriguingly, the performance difference between Slow MRD and Fast MRD is negligible, suggesting that MRD is relatively insensitive to the particular parameter values chosen for rate adaptation. Being able to perform consistently under different parameter values is useful, because determining the optimal parameter values for any kind of adaptive algorithm is often difficult in practice.

[4]The effective throughput is lower than the bit-rate because of link-layer overhead.

6. IMPLEMENTATION

This section describes the MRD system implementation and the implementation of the RFA protocol in detail.

6.1 System Implementation

We implemented the MRD system using commodity contemporary Pentium class PCs running Linux Kernel 2.4.20 and 802.11a/b/g wireless interfaces based on the Atheros 5212 chipset. We modified the MADWiFi driver to implement the MRDS component for 802.11a/b/g WLAN clients.

As described before, the primary function of the MRDS is to schedule retransmissions. To handle retransmissions within the driver software, we disable the wireless interface from retransmitting packets by setting the retry limit to zero. During our experimental evaluation, we discovered that doing so caused the distribution of frame inter-transmission times to peak at the nominal packet transmission time, despite many transmission losses. In other words, setting a zero retry limit also disabled exponential backoff in the 802.11 interface. It turns out this is the behavior mandated by the original 802.11 standard [6]: the contention window should reset to the lowest value after a packet reaches its retransmission limit.

Consequently, our current MRD implementation does not include CSMA exponential backoff. However, future releases [5] of the MADWiFi driver [1] will include software support for 802.11e [16], which includes a software API to allow the driver to modify the contention window size. Meanwhile, we have disabled exponential backoff in *all* of our experiments to make fair performance comparison between the 802.11 standard and our MRD-enhanced 802.11 system.

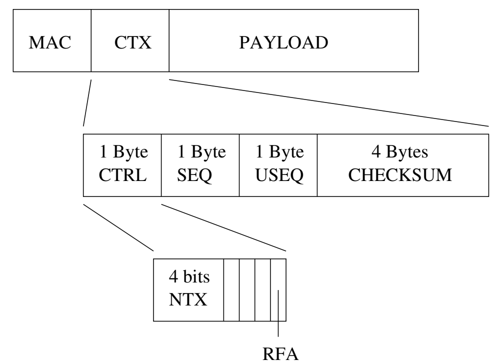
We used desktop PCs equipped with 802.11 wireless interfaces as access points. One AP acts as the active radio and is configured to run in the MADWiFi's "AP Master" mode. The passive radios are configured to run in MADWiFi's "Monitor" mode. On each of the APs, we run a user-level daemon to capture data frames from the wireless interface and forward them over a wired backbone (100 Mbps Ethernet in our experiments) to the MRDC running on another PC.

For increased efficiency, the AP daemon performs the CTX header checksum (see the next section) and drops frames that cannot be used for frame combining (i.e., those frames with a corrupt header). Because the RFA protocol does not require the client to acknowledge the receipt of an MRD-ACK, the AP daemon prepends the target client's MAC address in the MRD-ACK payload and transmits each MRD-ACK as a broadcast frame. Broadcasts saves the transmission of link-layer ACK frames in unicast and the benefit is much larger than the cost of expanding the size of the MRD-ACK payload. In our actual implementation, the AP daemon writes the target client's 6-byte MAC address in the source address field of the 802.11 header, thus saving us from expanding the MRD-ACK payload at all. We transmit the MRD-ACK packet at the lowest data rate (6 Mbps for 802.11a/g and 1 Mbps for 802.11b) for robust delivery.

Because the CRC computation is the bottleneck of the frame combining process, it is important to make it as efficient as possible. The MRDC currently implements a widely-used 8-bit table lookup algorithm to compute the 32-bit CRC checksum of a combined frame. Although the algorithm is simple, it is rather inefficient to process the entire frame to compute a new CRC value when the bit values for only a small portion of the frame changes during each iteration of the frame combining algorithm. In the future versions of the MRDC, we plan to implement an incremental CRC algorithm, which has been shown to reduce complexity by over an order of magnitude [12, 30].

We implemented the MRDC as a user-level daemon running on a 1.5 GHz Pentium 4 PC. Implementing the MRDC as a user-level daemon facilities debugging and running diagnostics. It forwards clean or corrected packets to the tunneling driver so that the Linux kernel can forward the packet using `iptables`.

6.2 Implementation of RFA



(a) Headers in the transmitted data frame

(b) MRD-ACK Packet

Figure 7: MRD-ACK control information.

Figure 7(a) shows the headers used by RFA. For every data frame transmission, the MRDS inserts a 7-byte Combiner Transmit (CTX) header that is prepended to the payload of the MAC-layer frame. The CTX header contains a *ctrl* field, which uses 4 bits to indicate the number of attempted transmissions (*ntx*) for the current data frame, 1 *rfa* bit to indicate that the sender has pending unacknowledged frames and is requesting for acknowledgment, and 3 unused bits reserved for future options such as out-of-order delivery. The 1-byte *seq* field labels the sequence number of the data frame, while *useq* labels the oldest transmitted data frame in the MRDS buffer that has not been acknowledged by the MRDC. When frame *useq* exceeds its retransmission limit, the MRDS advances *useq* to the *seq* number of the next unacknowledged frame in the retransmission buffer (if any). This allows the MRDC to detect frames that failed all its retransmissions and flush the blocked frames from the reorder buffer.

The MRDC uses the source address in the MAC header and the *seq* value in the CTX header to identify the frames that belong to the same network-layer packet. When the MRDC receives at least 2 corrupt data frames that correspond to the same packet, it attempts frame combining on the payload part of the data frame. Since it is important that the MRDC correctly identifies the frames that belong to the same packet, RFA uses a 4-byte CRC to protect the MAC and CTX header. If either the MAC or the CTX header is corrupted, the MRDC drops the entire frame.

The MRD-ACK packet contains a 2-byte "magic" value that is used to distinguish the MRD-ACK packet from other downlink data payload,[5] a 1-byte sequence number, and an N-bit bit vector to indicate the success or failures of up to N consecutive frames. The sequence number is the *seq* value of the first data frame in the bit vector being acknowledged. The MRDS uses the link-layer data frame checksum to detect errors in the MRD-ACK packet.

The size of the MRD-ACK payload is small (25 bytes in our implementation). Thus, its overhead is largely dominated by the preamble and header associated with the 802.11 frame. We can potentially decrease overhead further by piggybacking MRD-ACK packets on data frames being transmitted in the same direction.

Our RFA implementation allows the MRDC to delay ACK transmissions in terms of the number of successive transmissions made by the MRDS. Thus, MRDC can delay an ACK either by a timeout of length equal to D packet transmission times or by counting D packet transmissions from the MRDS. Delaying ACKs by counting packets removes the requirement for sub-millisecond-granularity timers and allows the MRDC to be implemented in user space. Note that retransmitted frames are counted as a transmission while extra frames that are simultaneously received by different MRD radios should not be counted. Because both types of frames have identical *seq* values, the MRDC uses the *ntx* value to distinguish the retransmitted frames.

The MRDC sends MRD-ACKs to the MRDS via the active radio (i.e., the AP with which the WLAN client is associated for MRDS running in the WLAN clients). The MRDC may also independently use fine-grained path selection [24] to choose the most reliable diversity radio for transmitting the MRD-ACK packet to the WLAN client.

7. EVALUATION

We conducted several experiments to evaluate the performance of MRD under different environments. We divide the presentation of the results into two categories, *HIVAR* and *LOVAR*, based on whether the WLAN client was experiencing a high or low degree of channel variability during the experiment. To create a high channel variability environment in HIVAR, we use a client transmitter that is set in motion during the experiment, while we use a stationary transmitter in LOVAR. In the rest of this section, we describe our experimental setup and analyze the results of each experiment.

7.1 Setup

We chose to conduct experiments in 802.11a mode to avoid interfering traffic from the production 802.11 WLAN in our lab. In all our experiments, we configure one of the AP receivers (R1 or R2) to be an active AP running in *Master* mode. We configure the other AP receiver to run passively in *Monitor* mode. We configure the client sender C to run in 802.11 *Managed* mode. We run the MRDS on the WLAN client to evaluate the performance for upstream traffic.

[5]Instead of using "magic", we should label the MRD-ACK with a unique value in the Ethernet `type` field [32]. We used the magic value in our implementation to facilitate logging using standard tools like `tcpdump` [2] during our experiments.

In all of the experiments, we set a maximum retransmission limit of 7 (initial transmission plus seven retransmissions). The MRD experiments used a MRD-ACK delay of $D = 8$ packet transmissions, a sender buffer size of $N = 64$ packets, and a retransmission timeout of $T_s = 90$ ms. We pick $B = 256$ bytes ($\therefore N_B = 6$), such that the maximum processing time to search through 2^{N_B} block combinations is less than S/r, where S is the transmitted frame size and r is the bit-rate. Bounding B in this way helps prevent the processing queue at the MRDC from building up.

In each experiment, the WLAN client sends 100,000 1472-byte UDP packets as fast as possible to saturate the wireless link. We repeat each experiment for five trials. On the first transmission of each packet, we insert a timestamp into the frame's payload. The timestamp remains unchanged on frame retransmissions. The timestamp allows us to measure and compare the packet delivery delay between MRD and the single radio communication schemes. Also, the payload of the packet contains a known bit pattern so that we can post-process the trace to analyze the probability of frame combining failure p_f as a function of different block sizes B.

Each MRD experiment involves two sub-experiments: in the first set (MRD-R1), we configure R1 to be the active AP with which the client associates and R2 to be the passive AP. In the second set (MRD-R2), R2 is active and associates with the client. We compared the performance using different active APs because the MRDS schedules retransmissions based on the link-layer feedback from the active AP.

As mentioned in Section 6, performing software-based retransmissions in the driver effectively disables exponential backoff in the wireless interfaces' firmware. To make a fair performance comparison between communication schemes, we used software-based retransmissions (and thus, disabling exponential backoff) in all of our experiments, including the single radio communication schemes. We discuss how disabling exponential backoff might affect our evaluation results in Section 8.

Because wireless communication is sensitive to the physical environment, we do not claim that the results of the experiments presented here are exhaustive and representative of *all* situations. Our main objectives are to conduct a set of experiments to illustrate the performance gains that MRD can achieve in the implemented system under a *real* environment with different degrees of channel variability, and to analyze the properties of the MRD system in depth.

We present the results of our HIVAR and LOVAR experiments in the following sections. The HIVAR experiments used the modified autorate algorithm as described in Section 5, but the LOVAR experiments were conducted before we implemented the modifications. Thus, the LOVAR experiments use the standard MADWiFi autorate algorithm, which could have reduced the performance of MRD for those experiments.

7.2 HIVAR Experiments

We compare the performance of single radio communication schemes against MRD when the client experiences a high degree of channel variability. Figure 8 illustrates the location of our APs and client in our HIVAR experiments.

7.2.1 Throughput

We define throughput to be the sum of the bits from unique frames received divided by the time elapsed between

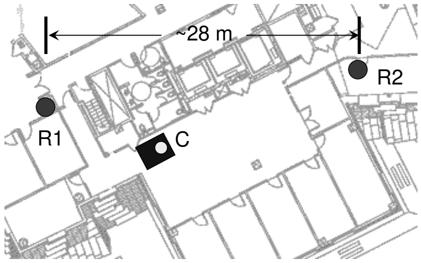


Figure 8: Setup for the HIVAR experiments. R1 and R2 are stationary receivers. C is a laptop transmitter client that was carried by a walking person who covered a 1.5 m × 2 m area during the experiments.

the first and last frame receptions. Note that the throughput metric accounts for the overhead of the CTX header, MRD-ACK transmissions, and all the processing delay associated with MRD.

The average throughput over five trials for the single radio experiments R1 and R2 were 8.25 Mbps and 6.42 Mbps, which are far below 802.11a's theoretical maximum UDP throughput of 31 Mbps. The high channel variability caused by mobility and distance has resulted in a large throughput reduction in the single radio experiments. Despite the harsh channel conditions, MRD-R1 and MRD-R2 maintained a average throughput of 18.7 Mbps and 18.36 Mbps respectively, which constitute improvements of 2.27× and 2.23× over R1 (and even more over R2).

We plot the throughput distribution of the one-second non-overlapping window samples in Figure 9(a). For R1 and R2, 80% of the samples are between 4-10 Mbps and fewer than 10% of the samples achieved a throughput more than 15 Mbps. In contrast, MRD achieves a throughput greater than 15 Mbps for more than 85% of the samples. These results suggest that even if we allow the WLAN client for the non-MRD cases to perform handoffs every second, the average throughput will remain well below 15 Mbps.

Both MRD-R1 and MRD-R2 achieved similar throughput results. This suggests that the performance of MRD is relatively insensitive to the choice of active AP, even when there is a significant difference in link quality between the two APs.

7.2.2 Source of Improvement

The large throughput improvement comes from the reduction in frame loss rate achieved by MRD. Table 3 summarizes the statistics of the raw frame loss rate (FLR) observed at the active AP in each sub-experiment and the ratio of the lost frames that were recovered (frame recovery rate, FRR) by MRD. The active APs in both sub-experiments suffered a raw FLR of about 35% and 39% but MRD was able to recover 50% and 57% of them, respectively. Because MRD was able to conceal a large number of losses from the rate adaptation algorithm, the sender was able to maintain a high bit-rate throughout both sub-experiments, as depicted in Figure 9(b), where over 90% of the frames were transmitted at a bit-rate of 24 Mbps or higher. In contrast, the single radio communication schemes suffers a high loss rate at the high bit-rates. Consequently, these schemes operate at low bit-rates. Actually, the selected bit-rates in R1 and

Experiment	FLR	FRR	FRR_{SS}	FRR_{FC}
MRD-R1	0.345	0.497	0.423	0.073
MRD-R2	0.391	0.573	0.515	0.058

Table 3: Frame loss and frame recovery rates of the high channel variability experiments.

R2 are spread across several of the low bit-rates due to the high degree of channel variability experienced by the client.

These results highlight the importance of MRDCallback(). If the procedure were not added to the autorate algorithm in MRD, the link-layer frame losses observed at the active AP would have been exposed to the autorate algorithm, causing MRD's to operate at the same low bit-rates as $R1$ and $R2$ in Figure 9(b).

We decompose the recovered frames into frames recovered by soft selection (FRR_{SS}) and block-based combining (FRR_{FC}). Thus, $FRR = FRR_{SS} + FRR_{FC}$. Our results show that 85% and 90% of the gains in MRD-R1 and MRD-R2 were achieved by soft selection (i.e., those frames that were received correctly by the passive AP but not by the active one).

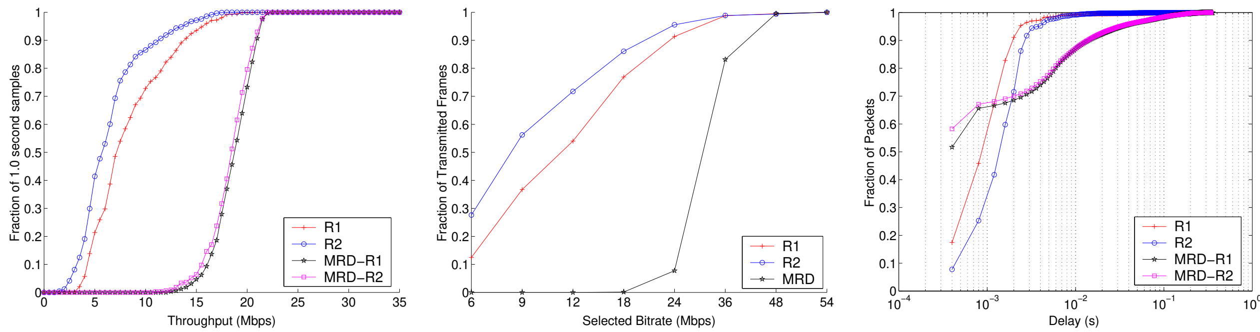
There are two possible explanations for the relatively small fraction of frames recovered by frame combining: i) there were few opportunities for running the packet combining either because most of the transmissions were already corrected by soft selection or because the MRDC did not collect enough valid corrupt frames (due to corrupt headers, etc.) to perform the combining; or ii) there were many frame combining attempts but most of them failed to recover the correct frame.

We analyzed the number of successful and failed frame combining attempts. The total number of frame combining attempts was high, constituting 34% and 26% of the total number of frames that were not successfully received by the active AP in MRD-R1 and MRD-R2. Although there were many opportunities for error recovery with frame combining, about 80% of those attempts failed to correct the errors in the transmitted frame in both sub-experiments.

One cause for the high failure rate is the low number of block subdivisions in a frame in our implementation ($N_B = 6$). We post-processed the data trace of our experiments to analyze how p_f varies with other values for N_B and plot the results in Figure 10(a).[6] The plot shows that p_f drops as N_B increases, which is consistent with the analytic model for burst bit-error channels that we derived in Section 3. For example, p_f drops from 80% to 60% when $N_B = 91$ (i.e., $B = 16$ bytes).

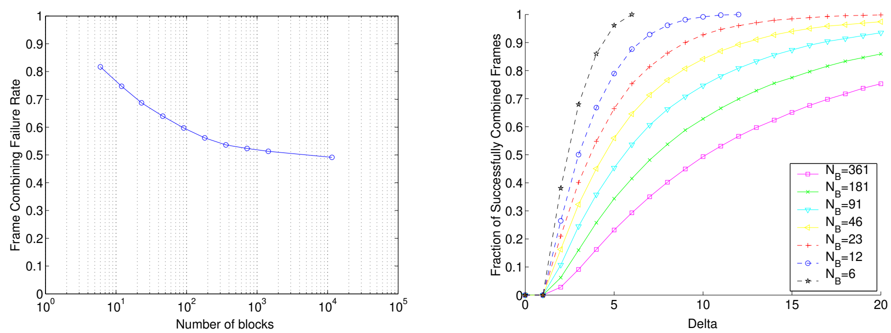
As discussed in Section 3, increasing N_B can potentially increase Δ, the number of differing blocks between two frames. To avoid overloading the MRDC, we may need to abort the frame combining operations for frames received

[6]Recall that p_f excludes those frames that are successfully delivered by soft selection (Section 3.1). While the majority of frame combining attempts were performed for corrupt frames that were simultaneously received by the APs, a significant fraction of the frame combining attempts were performed with retransmitted frames. For simplicity, we excluded the retransmitted frames in our post processing analysis. Nonetheless, our results should remain representative because the retransmitted frames should have an independent bit error behavior similar to the simultaneously received frames.



(a) One-second throughput avg. (b) CDF of selected bit-rate. (c) One-way delivery delay.

Figure 9: HIVAR Experiment Analysis. Leftmost: Distribution of throughput averaged over non-overlapping one-second window samples. Center: Distribution of selected bit-rate for each transmission. Right: Delay above the minimum one-way packet delivery time.



(a) Failure rate. (b) CDF of the number of differing blocks, Δ.

Figure 10: Trace-driven simulation of p_f and Δ for various values of N_B in the HIVAR experiments.

with a large Δ. Thus, a high Δ for a large fraction of combining attempts can offset the performance gain from increasing N_B. Figure 10(b) plots the distribution of the number of unmatched blocks (Δ_{succ}) for the successfully combined frames at various N_B. For $N_B \leqslant 91$, the 75th percentile Δ_{succ} value are much smaller than N_B (e.g., for $N_B = 91$, the 75th percentile of Δ_{succ} is 10).[7] This suggests that we could improve the performance of frame combining by re-running our experiments with a larger N_B value.

Finally, the low overhead of RFA allows MRD to achieve high gains. The number of MRD-ACKs transmitted constitute fewer than 7.5% of the total number of transmitted packets and fewer than 0.1% of the total number of transmitted bytes. The overhead of inserting an extra 7-byte CTX header to the 1500-byte packet payload is also negligible.

[7]Performing $2^{\Delta} = 2^{10}$ frame combining checksum operations for a 1500-byte packet takes about four milliseconds on a 3.2 GHz Pentium IV PC. The processing time is rather large and may cause the processing queue to build up at the MRDC. However, it should be possible to reduce the processing time substantially by using an incremental CRC update algorithm [12, 30] or using specialized hardware to perform the CRC calculation.

7.2.3 *Delay Analysis*

A number of compelling wireless applications such as telephony and video streaming require a relatively low packet delivery delay not exceeding $100 - 150$ ms [19]. We analyze MRD's delay performance here.

As described previously, we insert a timestamp in the payload of a packet's first transmission attempt to measure the one-way packet delivery delay. Because it is difficult to synchronize PC clocks to within a few tens of microseconds,[8] we do not measure the absolute packet delivery delay. Instead, we measure the delay jitter above the minimum one-way packet delivery time d_i for packet i, which does not require clock synchronization between the sender and the receiver. Let s_i and r_i be the start and receive timestamps associated with packet i for all $0 < i < 100,000$ packets transmitted in an experiment. Then $d_i = r_i - s_i - \min_i(r_i - s_i)$.

We also applied a piecewise linear regression algorithm [26] to remove clock skew between the sender and the receiver (we measured clock drifts on the order of 50 microseconds per second). Note that we can compute the

[8]We require the fine clock synchronization granularity because the nominal transmission time of 802.11a at high bitrates is less than 0.5 millisecond.

one-way packet delivery delay by adding $\min_i(r_i - s_i)$, which includes the nominal transmission time and processing delay. In practice, this number is less than one millisecond. We will ignore this minor adjustment and use the terms "delay jitter" and "delay" interchangeably.

Figure 9(c) shows the one-way delay distribution for our HIVAR experiments. The MRD median delay is below 1 ms and has 25% more packets delivered than R1 and R2. The low median delay is due to its ability to maintain a high bit-rate throughout the experiments. However, about 35% of the packets in MRD were delivered with a significantly higher delay than R1 and R2. Nonetheless, MRD was able to deliver 95% of the packets within a delay of 35 ms, which is well below the delay bound of 150 ms that can be tolerated by telephony and video applications.

We attribute the increased packet delivery delay in MRD to the fact that there were a significant number of frames that required retransmissions. In the design of the MRDC, we assumed an in-order packet delivery service and added a re-order buffer at the MRDC (Section 4.3). Whenever a retransmission is required, the reorder buffer blocks subsequent packets from being forwarded and increases the packet delivery delay for all of them.

Another source of delay comes from the losses of MRD-ACKs on the reverse channel, which delays the trigger to retransmit a packet. Also, the user space implementation of the MRDC is inefficient as interrupts and user space buffering can add delays in generating and sending MRD-ACKs.

7.3 LOVAR Experiments

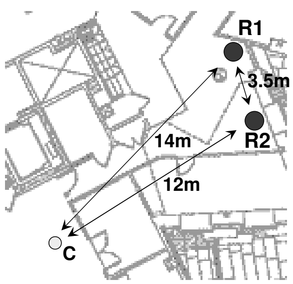


Figure 11: A diagram that illustrates the relative positions of the transmitter C and the receivers R1 and R2 in the LOVAR experiments.

We evaluate the performance of MRD in a scenario where the channel variability is low, using the setup depicted in Figure 11. The parameters and methods we use for the LOVAR experiments are the same as the HIVAR experiments, except that we use a stationary desktop transmitter instead of a mobile one. Also, our measurements for the LOVAR experiments were collected before we introduced our modifications to the MADWiFi autorate algorithm in Section 5. Thus, the autorate results presented in this section understate the performance of the MRD system. Nevertheless, our results provide an interesting comparison of the system operating under different situations.

7.3.1 Throughput

Figure 12(a) shows the throughput averaged over five trials for the LOVAR experiments. We ran different experiments using two different fixed bit-rates (36 and 48 Mbps) and using the standard rate adaptation algorithm (Auto) implemented in the MADWiFi WLAN driver. The figure shows that the MRD schemes at fixed bit-rate of 48 Mbps performed better than all other schemes. The dashed lines marks the maximum 802.11a UDP throughput for a fixed bit-rate 36 and 48 Mbps links, which are 23 and 27 Mbps respectively. The MRD throughput is between 94.4% and 96.6% of the maximum UDP throughput at 48 Mbps. Despite the overhead of transmitting MRD-ACK packets, MRD-R1 increases throughput over R1 by 54.6% at the fixed bit-rate of 48 Mbps, while MRD-R2 improves throughput over R2 by 20.2% at 48 Mbps. Similar to the HIVAR experiments, both MRD-R1 and MRD-R2 achieved very similar throughput results, again suggesting that the performance of MRD is relatively insensitive to the choice of active AP in our experiments.

Under autorate (Auto), the throughput gains by MRD-R1 and MRD-R2 diminish to 3.7% and 8.1% respectively. The main reason for the diminished gains is that the LOVAR MRD experiments used the unmodified version of the autorate algorithm. The algorithm ignores information from the MRD-ACK, so it adapts its bit-rate based only on the observed loss rate of the link-layer transmissions to the active AP. Consequently, the algorithm selects a suboptimal bit-rate. For example, Figure 12(b) shows that MRD-R2 (Auto) selected 36 Mbps roughly 70% of the time even though our fixed rate experiments shows that it could achieve a high throughput at 48 Mbps.

We believe that the MRD throughput results would improve if we used the modified autorate algorithm for MRD in our LOVAR experiments. The potential throughput performance could be as high as (if not higher than) the 48 Mbps fixed rate MRD experiments, which improved throughput by 9% to 16% over R1 Auto and R2 Auto.

Another reason for the diminished gain is the low variability of the channel. Although the frame loss rate was substantial at 48 Mbps, there was almost no loss at 36 Mbps. Thus, the throughput for $R1$ and $R2$ is lower bounded at the 36 Mbps bit-rate and caps the maximum achievable throughput gain for MRD.

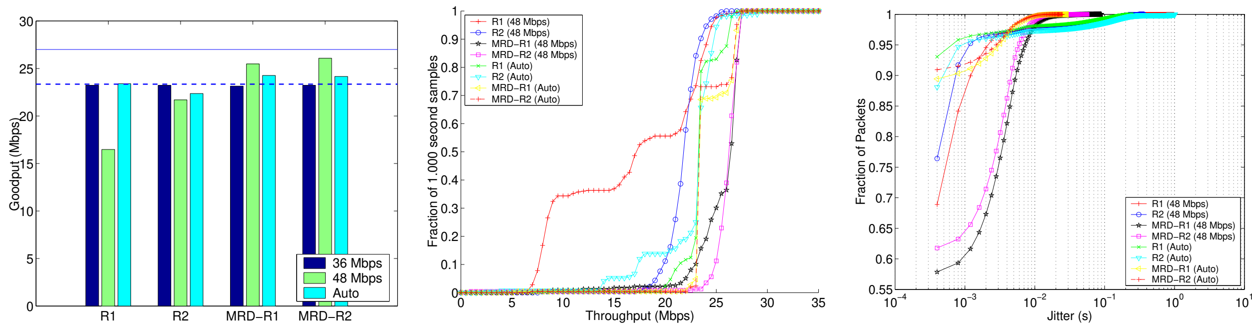
Experiment	FLR	FRR	FRR_{SS}	FRR_{FC}
MRD-R1	0.359	0.895	0.694	0.200
MRD-R2	0.354	0.958	0.819	0.139

Table 4: Frame loss and frame recovery rates of the low channel variability experiments.

7.3.2 Source of Improvement

We analyze the sources of improvement for the 48 Mbps fixed bit-rate LOVAR experiments and summarized the results in Table 4. The active APs in the LOVAR experiments observed similar frame loss rates to the ones observed in the HIVAR experiments, but the FLR is much higher. It ranged between $90\% - 96\%$ for LOVAR compared to $50\% - 57\%$ for HIVAR. There were also a larger number of frames recovered by frame combining in the LOVAR experiments.

We found that the total number of frame combining attempts was proportionately similar to the HIVAR experiments. It represents 37% and 25% of the total number of frames that were not successfully received by the active AP in MRD-R1 and MRD-R2. Thus, the increased number of



(a) Throughput. (b) One-second throughput avg. (c) One-way delivery delay.

Figure 12: LOVAR Experiment Analysis. Left: Throughput averaged over 5 trials. The dashed line marks the maximum achievable UDP throughput (23 and 27 Mbps) for 802.11a fixed at 36 Mbps and 48 Mbps bit-rates. Center: Distribution of throughput averaged in non-overlapping one-second window samples. Right: Delay above the minimum one-way packet delivery time.

frame combining recoveries were due to a large reduction in frame combining failure rate. Indeed, the frame combining failure rate p_f was about 45% in both sub-experiments, which is a large drop from the 80% in the HIVAR experiments.

Like the HIVAR experiments, the average p_f drops dramatically if the frame were subdivided into smaller blocks. Our simulation shows that $p_f = 17\%$ for $N_B = 91$ (i.e., $B = 16$ bytes). At the same time, Δ_{succ} (defined in Section 7.2.2) remains low for the successfully combined frames: the 95th percentile of Δ_{succ} for $N_B = 91$ is 10.

7.3.3 Delay Performance

We repeat the delay analysis in Section 7.2.3 for the LOVAR experiments. Figure 12(c) shows the one-way delay distribution for the fixed and autorate experiments. Compared to the HIVAR experiments, MRD delivered packets with a lot smaller delay because it was able to recover almost all corrupt frame transmissions to the active AP. As a result, the LOVAR experiments required much fewer frame retransmissions than the HIVAR experiments. Our LOVAR experiments show that MRD delivered 99% of the successfully received frames within 20 ms.

Finally, we observe a long tail in the one-way delay distribution (but representing only a tiny fraction of the transmitted packets) for the single radio communication schemes that last up to several hundred milliseconds. This tail is mostly an artifact of handling retransmissions in the driver, where kernel interrupts can happen in between retransmissions. We suspect that the long tails arise from packets that require a large number of retransmissions because such packets may experience more interrupts from the kernel than packets delivered with fewer or no retransmissions. The MRD scheme does not have this long tail because it successfully delivers packet with a lot fewer (re)transmissions.

In summary, we found that MRD produced throughput gains in all experiments, regardless of the channel variability experienced by the client. In our HIVAR experiments, we found that MRD was able to increase throughput up to 2.3× that of the best AP when only a single radio is used. In our LOVAR experiments, MRD, at a *fixed* bit-rate of 48 Mbps, the throughput improvement is less impressive but still significant, between 9% and 16%.

8. DISCUSSION

The conventional wisdom of managing link quality in WLANs is to have the clients adapt to the channel conditions (i.e., adapt the bit-rate) before changing to an alternate link (e.g., AP) with a better channel quality. MRD can be viewed as taking the opposite approach, where the clients use multiple links simultaneously before changing their bit-rate to adapt to the underlying channel.

Our experimental results suggest that, with MRD, even a simple rate adaptation algorithm, such the one based-on MADWiFi, can perform well in different environments. In contrast, we observed a rather large performance difference between the HIVAR and LOVAR experiments in the non-MRD schemes. We believe that the large performance difference is attributed not only to the increased frame loss rates observed at the individual APs, but also to the sub-optimal bit-rates that might have been chosen by the MADWiFi autorate algorithm in the high channel variability environment. (In Section 5, we tuned the algorithm to work well with multiple radios and frame combining, but did not alter the fundamental mechanisms used in the algorithm.)

In fact, we can use the results from the previous section to show that there is room for improvement in the rate adaptation algorithm. Table 3 and Figure 9(b) show that the frame loss rate to the active AP in MRD-R1 was 35% and that MRD-R1 selected a bit-rate of at least 24 Mbps over 90% of the time. Multiplying $1 - FLR$ with the the effective throughput of the 24 Mbps (17.8 Mbps) bit-rate yields 11.6 Mbps. Thus, we could have fixed the bit-rate to 24 Mbps for the non-MRD HIVAR experiment ($R1$) to improve the performance by 1.4× over the MADWiFi autorate algorithm, which achieved 8.25 Mbps. (Although the improvement is significant, it is not as great as MRD-R1, which achieved a 2.3× improvement at 18.7 Mbps.)

We are not suggesting that a fixed bit-rate should be used for non-MRD wireless links operating in a channel with high variability: selecting an optimal fixed bit-rate for such a channel still requires an adaptive algorithm. Rather, our intent is to use the example to motivate the following open questions: 1) Could other existing autorate schemes (e.g., RBAR [15], AARF [20], MiSer [27], OAR [29], SampleRate [10]) be used to improve performance of the non-MRD schemes in our HIVAR experiments? 2) Can we design an autorate algorithm for a non-MRD WLAN that performs well under a variety of channel conditions in a real environment? These are open questions, but we have demonstrated—using real-world experiments—that MRD can use a simple rate adaptation algorithm to produce good performance under different and difficult channel conditions, and that with MRD, the need for a finely tuned rate adapatation algorithm is not as important as with single-radio schemes.

As mentioned in Section 6, the MRDS needs to assume control over all retransmissions. Performing software-based retransmissions in the driver, however, also has the side effect of disabling the exponential backoff controlled by the firmware.

We acknowledge that the relative throughput improvement by MRD may be reduced when exponential backoff is enabled. That is because the link layer increases the backoff window whenever a client fails to successfully transmit a data frame to the target receiver (i.e., the active AP) at the link layer. In our current design, the link layer is oblivious to MRD. Even if the data frame is recovered through soft selection or block-based combining, the link layer may not reduce the contention window (which is what CSMA does when the link-layer transmission succeeds). Consequently, the backoff window may increase unnecessarily and reduce MRD's performance.

Conceivably, we can alleviate the problem by creating an interface that allows MRD to inform the link-layer backoff mechanism about the results of frame recovery at the MRDC. Designing a medium access control algorithm that can adapt to MRD's error recovery results is an interesting open problem.

Despite the above caveat, MRD effectively reduces frame losses and the total number of transmissions required to deliver a packet, without increasing the nominal frame transmission time as in other existing approaches like using lowering data rates or employing forward error correction.

9. RELATED WORK

The idea of coordinating multiple radios in WLANs has recently received considerable attention. The authors in [9] proposed to embed multiple radios on a single device for better energy and mobility management, capacity enhancement, and avoiding channel failures. A system that uses fine-grained path selection (FGPS) that switches transmissions from among a set of nearby APs was demonstrated to effectively reduce path dependent losses in WLANs [24] and to improve the quality of video streaming applications [23]. FGPS takes advantage of path diversity at the *transmit side* of the system. MRD compliments FGPS to reap the benefit of path diversity at the *receive side* of the system.

Diversity reception is a common technique used to mitigate the effects of fading, and interference in wireless systems. Almost all WLAN devices have embedded more than one antenna that gets selected based on packet loss rates. Recently, the IEEE incorporated a more advanced antenna diversity technique Multiple-Input Multiple-Output (MIMO) [25] into the physical layer specifications of their next generation WLAN devices known as 802.11n [35]. In general, this class of techniques, known as *microdiversity*, are tightly integrated with the physical layer and mostly help in mitigating path-dependent effect localized at one receiver. In contrast, the MRD WLAN operates above the physical layer and may be used to collect data frames received by radios distributed across different access points at different locations; thus, our system achieves diversity at the macro level. Furthermore, a WLAN operator can build a MRD WLAN using 802.11n hardware to exploit the path diversity gains at different scales.

Code Division Multiple Access (CDMA) cellular phone networks have long used "macrodiversity" to improve performance and to provide seamless handoff between base stations [28]. Both [21] and [33] applied the idea to use macrodiversity and frame combining on frames received from adjacent access points to improve uplink WLAN performance in the same way as MRD. However, [21] presents simulated results based on a capture model and ignores protocol level issues such as ARQ. The contributions of [33] lie in the theoretical performance evaluation in the context of a WLAN based on Bluetooth [11] radios. In contrast, our contributions lie in the design of a macrodiversity system that works well in CSMA-based WLANs and in conducting a performance study of a fully implemented receiver macrodiversity system on a real testbed.

The idea of recovering a frame by combining it with a retransmitted version was first proposed in [31] and then further analyzed in [13, 14]. Hybrid ARQ is an extension of this technique, which combines forward error correction (FEC) and retransmission to recover unsuccessful transmissions [22]. Although numerous hybrid ARQ schemes are available, we chose block-based combining because it has the advantage that 1) there is no encoding and it inserts no extra FEC bits into the frame, 2) it uses hard decision (i.e., it performs correction using only the received data bits without requiring extra information from the physical layer), and consequently 3) the algorithm is easy to implement. A scheme proposed in [8] uses collaborative decoding to improve link reliability. The scheme is complex and requires the receivers to exchange soft decision estimates of each data symbol, which is not accessible from any wireless device on market today.

A proposal that uses 802.11's request-to-send(RTS)/clear-to-send(CTS) control packets to convey the results of packet combining is outlined in [13]. Because RTS/CTS is required for every transmission, the proposed method (which was not implemented and experimentally evaluated) will produce much higher overhead than our RFA protocol.

Like MRD, multi-user diversity [18] and medium-access diversity [17] also exploit the fact that losses at different receivers occur independently. In both techniques, an AP has a queue of packets destined for different clients and attempts to improve network performance by scheduling transmissions to the client receiver that has the best channel condition in a given moment. In contrast to MRD, the technique requires explicit receiver selection and channel quality feedback from each receiver. These requirements are necessary if the receivers are not inter-connected by a high bandwidth back-channel that MRD relies upon.

10. CONCLUSION

MRD uses wireless path diversity to improve loss resilience in wireless local area networks. It coordinates wireless receptions among multiple radios—either co-located on the same device or distributed across different access points in the WLAN infrastructure—to increase loss resilience against path-dependent corruptions in the wireless medium. Using multiple radios, MRD performs frame combining, which attempts to correct bit errors by combining corrupt copies of data frames received by each radio in our system. Because losses are often independent among different receivers, MRD is able to achieve significant improvement in loss rates.

Our experiments in an in-building testbed using commodity PCs and 802.11a/b/g wireless interfaces demonstrate throughput gains of up to 2.3× that of single radio communication schemes, with the corresponding one-way delay bounded to 35 ms for 95% of the delivered packets.

From the experience we gathered in building and evaluating MRD, we discovered a number of performance optimizations such as marking packets for low-latency and out-of-order delivery, and sharing MRD feedback with the link-layer to improve rate adaptation and contention window adjustments. We plan to pursue to design and integrate these optimizations to further improve the performance of MRD in the future.

11. ACKNOWLEDGMENTS

We thank our shepherd Rajive Bagrodia for his useful suggestions in the revision of this paper. We also thank Godfrey Tan, Srikanth Kandula, Eugene Shih, John Bicket, Vladimir Bychkovsky and the anonymous reviewers for their insightful comments, and Michel Goraczko for his technical assistance.

This material is based upon work supported by the National Science Foundation under Grant No. 0205445. The views expressed in this paper are not necessarily those of the National Science Foundation.

12. REFERENCES

[1] Madwifi: Multiband Atheros Driver for WiFi. http://madwifi.sourceforge.net/.

[2] tcpdump/libpcap. http://www.tcpdump.org.

[3] Engim product overview. http://www.engim.com/products.html, 2003.

[4] IEEE P802.11i/D10.0. Medium Access Control (MAC) Security Enhancements, Apr. 2004.

[5] Madwifi mailing list archive. http://news.gmane.org/gmane.linux.drivers.madwifi.devel, Feb. 2005.

[6] IEEE 802.11b/d3.0 Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specification, Aug. 1999.

[7] IEEE Standards for Local and Metropolitan Area Networks: Standard for Port Based Network Access Control, Oct. 2001.

[8] Avudainayagam, A., Shea, J., Wong, T., and Li, X. Reliability exchange schemes for iterative packet combining in distributed arrays. In *Proc. of IEEE WCNC* (New Orleans, LA, Mar. 2003), pp. 832–837.

[9] Bahl, P., Adya, A., Padhye, J., and Wolman, A. Reconsidering wireless systems with multiple radios. *ACM CCR* (Oct. 2004), 39–46.

[10] Bicket, J. C. Bit-rate selection in wireless networks. Master's thesis, Massachusetts Intitute of Technology, Cambridge, MA, Feb. 2005.

[11] Specification of the bluetooth system. http://www.bluetooth.com/, Dec. 1999. Bluetooth Special Interest Group document.

[12] Braun, F., and Waldvogel, M. Fast incremental CRC updates for IP over ATM networks. In *Proc. of IEEE HPSR* (Dallas, TX, May 2001).

[13] Chakraborty, S., Yli-Juuti, E., and Liinaharja, M. An ARQ scheme with packet combining. *IEEE Communications Letters 2*, 7 (July 1998), 200–202.

[14] Daraiseh, A.-G. A., and Baum, C. W. Methods for packet combining in HARQ systems over bursty channels. *Mobile Networks and Applications 2* (Oct. 1997), 213–224.

[15] Holland, G., Vaidya, N. H., and Bahl, P. A rate-adaptive MAC protocol for multi-hop wireless networks. In *Proc. of ACM MobiCom* (Rome, Italy, July 2001), pp. 236–251.

[16] IEEE 802.11 Working Group. Draft Supplement to International Standard for Information Exchange between systems - LAN/MAN Specific Requirements, Nov. 2001.

[17] Ji, Z., Yang, Y., Zhou, J., Takai, M., and Bagrodia, R. Exploiting medium access diversity in rate adaptive wireless LANs. In *Proc. of ACM MobiCom* (Philadelphia, PA, Sept. 2004), pp. 345–359.

[18] Knopp, R., and Humblet, P. A. Information capacity and power control in single-cell multiuser communications. In *Proc. of IEEE ICC* (Seattle, WA, June 1995), pp. 331–335.

[19] Kopsel, A., and Wolisz, A. Voice transmission in an IEEE 802.11 WLAN based access network. In *Proc. of ACM WoWMoM* (Rome, Italy, July 2001), pp. 23–32.

[20] Lacage, M., Manshaei, M. H., and Turietti, T. IEEE 802.11 rate adaptation: A practical approach. In *Proc. of ACM MSWiM* (Venezia, Italy, Oct. 2004), pp. 126–134.

[21] Leung, V. C. M., and Au, A. W. Y. A wireless local area network employing distributed radio bridges. *Wireless Networks 2* (1996), 97–107.

[22] Lin, S., Costello, D., and Miller, M. Automatic-repeat-request error-control schemes. *IEEE Communications Magazine 22*, 12 (Dec. 1984), 5–17.

[23] Miu, A., Apostolopoulos, J., Tan, W. T., and Trott, M. Low-latency wireless video over 802.11 networks using path diversity. In *Proc. of IEEE ICME* (Baltimore, MD, July 2003), vol. 2, pp. 441–444.

[24] Miu, A. K., Tan, G., Balakrishnan, H., and Apostolopoulos, J. Divert: Fine-grained path selection for wireless LANs. In *Proc. of ACM MobiSys* (Boston, MA, June 2004), pp. 203–216.

[25] Molisch, A. F., and Win, M. Z. MIMO systems with antenna selection. *IEEE Microwave Magazine* (Mar. 2004), 46–56.

[26] Moon, S. B., Skelly, P., and Towsley, D. Estimation and removal of clock skew from network delay measurements. In *Proc. of IEEE INFOCOM* (New York, NY, Mar. 1999), vol. 1, pp. 227–234.

[27] Qiao, D., and Choi, S. Goodput enhancement of IEEE 802.11a wireless LAN via link adaptation. In *Proc. of IEEE ICC* (Helsinki, Finland, June 2001), pp. 161–175.

[28] Rappaport, T. *Wireless Communications*. Prentice Hall, Upper Saddle River, N.J., 1996.

[29] Sadeghi, B., Kanodia, V., Sabharwal, A., and Knightly, E. Opportunistic media access for multirate ad hoc networks. In *Proc. of ACM MobiCom* (Atlanta, GA, Sept. 2002), pp. 24–35.

[30] Satran, J., Sheinwald, D., and Shimony, I. Out of order incremental CRC computation. To appear: IEEE Trans. on Computers, 2005. http://www.haifa.il.ibm.com/satran/ips/crc_23feb2003.pdf.

[31] Sindhu, P. Retransmission error control with memory. *IEEE Trans. on Communications 25* (May 1977), 473–479.

[32] Stevens, R. W. *TCP/IP Illustrated*. Addison-Wesley, Reading, MA, 1994.

[33] Valenti, M. C. Improving uplink performance by macrodiversity combining packets from adjacent access points. In *Proc. of IEEE WCNC* (New Orleans, LA, Mar. 2003), pp. 636–641.

[34] Willig, A., Kubisch, M., Hoene, C., and Wolisz, A. Measurements of a wireless link in an industrial environment using an IEEE 802.11-compliant physical layer. *IEEE Trans. on Industrial Electronics 43* (Dec. 2002), 1265–1282.

[35] Wilson, J. M. The next generation of wireless LAN emerges with 802.11n. Device Forge, Aug. 2004. http://www.deviceforge.com/articles/AT5096801417.html.