Designing an Efficient Retransmission Scheme for Wireless LANs: Theory and Implementation

Dimitrios Koutsonikolas
Chih-Chun Wang
Y. Charlie Hu
Ness Shroff

TR-ECE-10-5
June 30, 2010

School of Electrical and Computer Engineering
1285 Electrical Engineering Building
Purdue University
West Lafayette, IN 47907-1285

Contents

Abstract

Network coding is known to benefit the downlink retranmissions by the AP in a wireless LAN from exploiting overhearing at the client nodes. However, designing an efficient and *practical retranmission scheme remains a challenge. We present an (asymptotically) optimal scheme, ECR, for reducing the downlink retransmissions by the AP in a wireless LAN from exploiting overhearing at the client nodes. The design of ECR, consisting of three components: batch-based operations, a systematic phase-based network coding decision policy, and smooth integration of inter-flow and intra-flow coding, is accompanied by a theoretical underpinning, yet enables practical implementation on off-the-shelf 802.11 hardware. We analytically show ECR can achieve much higher reduction in packet retransmissions than previous schemes, and validate its performance gain via simulations and testbed implementation. To our knowledge, ECR is the first protocol that leverages both intra-flow and inter-flow network coding to solve a real-world problem in single-hop wireless networks.*

1 Introduction

Consider a typical scenario of an 802.11 WLAN. Multiple clients are associated to the Access Point (AP). The AP forwards traffic between each client and the wired Internet and uses 802.11 unicast for packet transmissions between itself and the clients. We focus on the downlink traffic, *i.e.,* in the direction from the AP to each client, as the downlink traffic dominates the uplink traffic in typical AP deployments.

To deal with packet losses due to the inherent lossy wireless medium and due to interference and collisions from other clients and other WLANs in the neighborhood, the 802.11 protocol employs a simple retransmission mechanism for unicast communication. A sender waits for acknowledgment (ACK) after each packet transmission and retransmits the packet if the ACK is not received after a short, fixed amount of time. Each packet is retransmitted up to a maximum number of times and is then dropped. Under high loss rates, *e.g.,* in the presence of a large number of clients, this simple retransmission mechanism can cause significant overhead and severely limit the throughput of the WLAN, as the AP may spend a significant portion of the airtime retransmitting lost packets. This motivates the need for novel, more efficient retransmission schemes.

Network coding is a novel technique that can help in designing efficient retransmission schemes. In particular, network coding can benefit the downlink retransmissions by exploiting overhearing at the client nodes. Consider an AP and two clients C_1, C_2. The AP has two packets, p_1, p_2, one for each client, respectively. In a lossy wireless network, it may happen that both packets are lost and the AP would have to retransmit both of them. However, due to the broadcast nature of the wireless medium, it can also happen that C_1 received the packet p_2 destined to C_2 and C_2 received the packet p_1 destined to C_1. In that case, the AP can XOR the two packets and broadcast the combined packet $p_1 \oplus p_2$. Then C_1 can extract its own packet p_1 by XORing $p_1 \oplus p_2$ with p_2 and similarly, C_2 can extract p_2 by XORing $p_1 \oplus p_2$ with p_1, thus saving one transmission. This simple idea can be extended beyond the two-client example, further improving the retransmission efficiency.

The potential benefits of inter-flow coding based retransmission schemes for wireless LANs were first considered by MU-ARQ [14]. [14] first sketches the basic principle about how to perform inter-flow coding when there are $M \geq 2$ clients, which we refer to as the MU-ARQ principle (or simply MU-ARQ as shorthand). The MU-ARQ principle then becomes the foundation of the theoretical analysis in [14] and the practical implementation in [21]. More explicitly, [14] shows that under some idealistic assumptions, the throughput benefit of a MU-ARQ-based scheme can be quantified analytically, and it increases monotonically when the number of clients increases. ER [21] is a practical implementation of the MU-ARQ principle. By addressing several practical issues,

such as feedback frequency, packet delay time-out, link asymmetry, that were previously ignored in MU-ARQ, the practical ER protocol aims to materialize the promised throughput benefits of inter-flow coding.

In this paper, we first observe that the MU-ARQ principle does not realize the full potential of inter-flow network coding due to the fact that the protocol is overly conservative and exploits coding opportunities in a passive way. More explicitly, when a client receives a coded packet that it cannot decode, the MU-ARQ principle simply discards it, even though retaining such a packet may allow more or more efficient (*e.g.*, mixing more packets into one that can benefit more clients) coding opportunities in the future.[1] The second drawback of the MU-ARQ principle is that it was originally proposed as a pure theoretic concept that admits simple throughput analysis but does not take into account the practical constraints of a real network environment. Therefore, converting the MU-ARQ principle to practical implementations generally requires complicated interplay between the feedback frequencies, the scheduling, and coding decisions.For example, [21] shows that determining the optimal coding strategy is an NP-hard problem. Therefore, several heuristics are used in the corresponding ER protocol [21], which further reduce the achievable throughput and make the performance of the ER protocol very sensitive to the underlying network environment.

Motivated by the inefficiencies of the MU-ARQ principle, we propose ECR, a network coding based retransmission protocol, designed from scratch, that *(i) realizes the full potential of network coding and (ii) smoothly combines the throughput enhancement of inter-flow coding and the practical feedback-reduction benefits of intra-flow coding.* ECR is based on the concept of "batches" similar to that of intra-flow coding. When the batch size N is sufficiently large, ECR attains the provably optimal inter-flow coding gain that is strictly better than that of the MU-ARQ principle, especially when the number of clients is from moderate to large. By inherently exploiting the simplicity advantage of intra-flow coding, the operation of ECR is straightforward and does not involve solving an NP-hard problem. There is thus no need to resort to suboptimal heuristics and the throughput enhancement of ECR is robust over various network environments. In short, ECR successfully bridges the gap between theory and practice of network coding in 1-hop WLANs over the downlink direction.

This work makes the following contributions. (1) We present the design of ECR, which is guided by rigorous theoretical analysis yet enables practical implementation on off-the-shelf 802.11 hardware. (2) We characterize analytically the throughput advantage of ECR over the MU-ARQ principle. (3) We compare ECR with the ER protocol, the heuristic-based implementation of the MU-ARQ principle, through extensive simulations and a testbed implementation. The empirical results show that ECR achieves robust performance and significantly outperforms ER in a variety of scenarios.

2 Previous Theoretical Result

The contribution of [14] is two-fold: The explicit description of the MU-ARQ principle and the corresponding throughput analysis under some idealistic assumptions. More explicitly, let us consider a WLAN with M clients. The MU-ARQ principle consists of two stages. In the first stage, the AP repeatedly sends out a batch of uncoded packets for each clients. Suppose the AP is currently sending the uncoded k-th packet of the i-th client C_i, which is denoted by $X_{i,k}$. The AP moves to sending the next uncoded packet only when $X_{i,k}$ is received by at least one of the clients C_j for some $j \in [M] \stackrel{\Delta}{=} \{1, \cdots, M\}$. The AP moves to Stage 2 when all uncoded packets are heard by at least one of the M clients. In Stage 2, the AP XORs the packets together according to the overhearing pattern in a similar way as in COPE [12].

The analysis of the MU-ARQ principle is straightforward under several idealistic assumptions: (i) The feedback is reliable and of zero cost. Therefore, the clients can send instant feedback to the AP after *every packet*

[1] [14] noticed that the MU-ARQ principle could potentially benefit from retaining the overheard coded packet. However, no discussion was made in [14] on how to exploit such overheard coded packet. The proposed ECR protocol explicitly specifies how to algorithmically exploit the overheard coded packets.

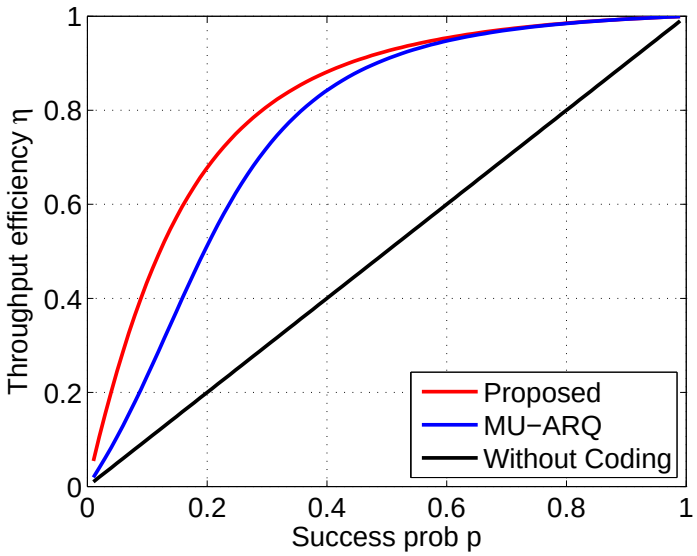


Figure 1. Comparison of Throughput Efficiency for MU-ARQ and the theoretically optimal ECR, $M = 20$ **clients.**

transmission so that the AP has the complete knowledge of the overhearing pattern; there is thus no ambiguity when making the coding decision; (ii) Infinitely large batch size, which ensures that on average the noisy forward direction can be viewed as a deterministic noiseless channel; (iii) Perfect symmetry between all clients so that the coding opportunities can always be perfectly paired. For example, the analysis of MU-ARQ assumes that the number of packets for client C_1 that are overheard only by client C_2 is equal to the number of packets for C_2 that are overheard only by C_1. Therefore, these two groups of packets can be perfectly paired with each other.

With the above three assumptions, [14] quantifies the *throughput efficiency* of the MU-ARQ principle, which is also known as the *sum-rate capacity* and is defined as

$$\eta = \frac{MN}{T} \tag{1}$$

where M is the number of clients, N is the number of packets at the AP for each client, and T is the overall number of time slots required to finish the transmission of the MN packets. Obviously, $\eta \leq 1$ by definition as each time slot can carry at most one packet even with perfect channels. The higher the throughput efficiency η, the larger throughput is for the entire system. The throughput efficiency of MU-ARQ for infinitely large N is computed in [14] by

$$\begin{aligned} &\lim_{N\to\infty} \eta_{\text{MU-ARQ}} \\ &= \frac{1-(1-p)^M}{1+\frac{1-p}{Mp^2}\left(1-(1-p)^M - Mp(1-p)^{M-1}\right)} \end{aligned} \tag{2}$$

where we assume that the probability that a packet sent by the AP is received by Client C_i successfully is p for all $i \in [M]$, and the success events for different Clients C_i and C_j are independent for all $i \neq j$. Figure 1 plots $\lim_{N\to\infty} \eta_{\text{MU-ARQ}}$ in (2) over different values of p.

The idealistic assumptions (i)–(iii) turn out to be quite restrictive in practice and require complicated mechanisms to coordinate the coding decisions with the real network dynamics. For example, practical protocols must

be based on a short-to-moderate finite batch size and periodic (non-instant) feedback. Therefore, the randomness of the channel will seriously impact the network dynamics and the intrinsically unreliable feedback will further exacerbate the problem. Moreover, for a dynamic (generally asymmetric) environment with finite batch size, one cannot assume all the coding opportunities could be perfectly paired with each other. [21] shows that in this case the transmission order of the coded packets will affect the throughput substantially and finding the optimal coding sequence can be a NP-hard problem.

2.1 ER

The entangled interplay between the batch size, feedback frequency, scheduling and the coding decision is the main subject of [21]. [21] proposes ER, the first *practical* protocol to exploit network coding to improve the efficiency of retransmissions by the AP. Thus, ER can be viewed as a practical implementation of the MU-ARQ principle. In the following, we describe the main design choices made in ER in relaxing the idealistic assumptions (i)–(iii) in [14].

Relaxing assumption (i) – perfect feedback. Unlike 802.11, where a receiver responds with an ACK every time it receives a packet, clients in ER send *periodic cumulative feedback* to inform the AP of missing packets, with the goal to minimize the impact of feedback losses while at the same time keeping the feedback overhead low. Each cumulative feedback message includes two fields: the starting sequence number (*start*) and a 64-bit bitmap, effectively informing the AP of the reception status of the last 64 packets.

Relaxing assumption (ii) – infinite batch size. Being a practical protocol, ER tries to achieve a balance between high coding efficiency and low delay. Instead of working with a fixed batch in rounds, the AP in ER uses a dynamic, threshold-based heuristic to determine (i) when a packet needs a retransmission and (ii) whether it should retransmit a packet or transmit a new one every time the medium is free. Each packet in the retransmission queue has a timeout which is calculated based on the time arrivals of cumulative ACKs in the past. If the timeout expires, the packet is considered ready for retransmission. The AP retransmits a packet if the number of packets ready for retransmission reaches a threshold (retransmission queue threshold) or the time a packet has spent in the retransmission queue exceeds another threshold (retransmission queue timeout).

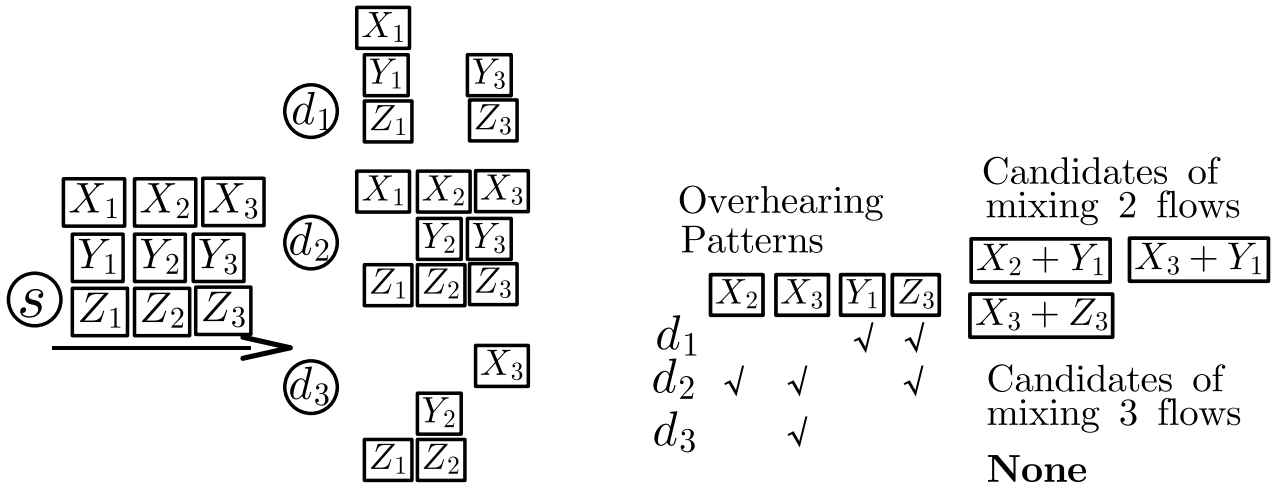
Relaxing assumption (iii) – perfect coding opportunities. In practice, in an asymmetric environment and with a finite batch size, one cannot assume all the coding opportunities can be perfectly paired with each other. [21] showed that in this case, finding the optimal coding strategy is an NP-hard problem, and proposed two heuristics to solve this problem: (i) a simple "sort-by-time" heuristic, in which packets awaiting for retransmission are sorted according to their arrival time and the AP always starts with the packet that arrived the earliest and tries to combine it with as many subsequent packets as possible, and (ii) a greedy "maximum clique" heuristic that tries to mix together as many packets as possible.

3 Motivation

In this section, we use a few illustrative examples to explain the fundamental limitations of MU-ARQ. These examples will also motivate the straightforward design of our proposed ECR protocol in Sections 4, 5.

Take Figure 2 for example. Suppose in the initial stage, the AP s transmits 9 native packets among which X_1 to X_3 are intended for client d_1; Y_1 to Y_3 are intended for d_2; and Z_1 to Z_3 are intended for client d_3, respectively. All 9 packets are sent using wireless broadcast, and due to the randomness of the wireless channel, clients d_1 to d_3 may have different overhearing patterns, as illustrated in Figure 2(a).

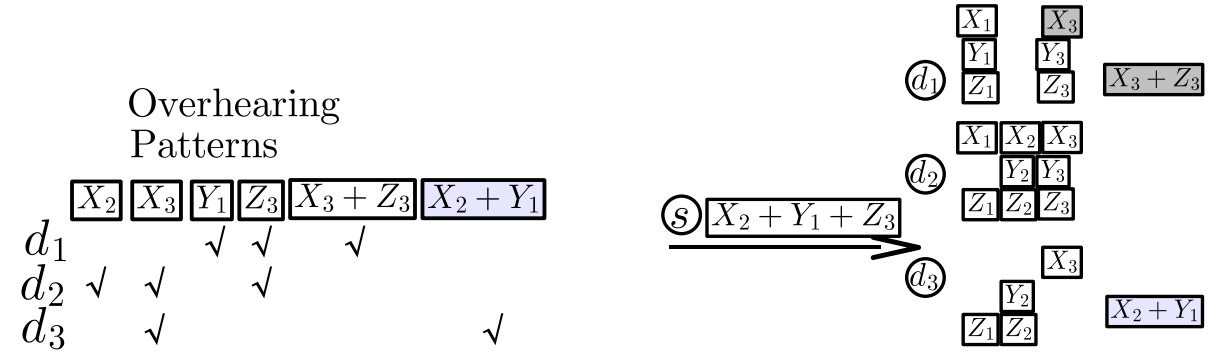
MU-ARQ takes advantage of the "diversity" of the overhearing patterns at the clients. For example, packet X_2 is heard by d_2 but not by $\{d_1, d_3\}$, packet X_3 is heard by both d_2 and d_3 but not d_1. Packet Y_1 is heard only by d_1 and packet Z_3 is heard only by $\{d_1, d_2\}$. See Figure 2(b) for the list of the 4 overheard packets that



(a) The overhearing patterns

(b) Coding opportunities

Figure 2. Illustration of the MU-ARQ protocol.



(a) The overhearing patterns when the AP sends $[X_3 + Z_3]$ and then $[X_2 + Y_1]$.

(b) Sending $[X_2 + Y_1 + Z_3]$ will take advantage of the previously overheard $[X_2 + Y_1]$ packet.

Figure 3. Aggressive exploitation of the overheard coded packets.

may lead to potential coding opportunities. MU-ARQ then searches for the inter-flow coding opportunities that combine 2 flows (or even 3 flows) together. For the example in Figure 2, there are three possible candidates for mixing two flows together. For example, since X_2 is heard by d_2 and Y_1 is heard by d_1, an XOR coded packet $[X_2 + Y_1]$ combining flows 1 and 2 will realize the inter-flow coding benefits. Similarly, $[X_3 + Y_1]$ and $[X_3 + Z_3]$ are candidates of mixing 2 flows together. Since there is no Y packet that is heard by both $\{d_1, d_3\}$, there is no coded packet that can simultaneously mix 3 flows. realize the inter-flow coding benefit, for the next time slot, the AP s has the freedom to send any one of the three candidates.

Nonetheless, MU-ARQ does not realize the full potential of the network coding due to the following reasons.

3.1 Inherent Limitation of MU-ARQ

An inherent limitation of MU-ARQ is the following: **Issue 1: The protocol is overly conservative and exploits coding opportunities in a passive way.** Continue our example in Figure 2. Suppose the AP decides to send $[X_3 + Z_3]$ and then $[X_2 + Y_1]$. Due to the channel randomness, $[X_3 + Z_3]$ and $[X_2 + Y_1]$ are heard by d_1 and by d_3, respectively; see Figure 3(a) for the overhearing pattern table. By hearing $[X_3 + Z_3]$, d_1 can now decode X_3

as illustrated in Figure 3(b).

The remaining question is thus *what is the optimal choice of the next coded packet.* MU-ARQ answers this question in the following conservative way. We first notice that the coded packet $[X_2 + Y_1]$ heard by d_3 cannot be used to decode any additional information, since d_3 has heard neither X_2 nor Y_1 in the past. As a result, MU-ARQ discards the overheard coded packet $[X_2 + Y_1]$ at d_3. Since each of d_1 to d_3 only needs one additional packet (more explicitly, they need X_2, Y_1, and Z_3, respectively), MU-ARQ checks whether they can be mixed together. Since X_2 is now heard only by d_2 (resp. Y_1 is heard only by d_1) as illustrated in Figure 3(a), these three packets will not be mixed together by MU-ARQ. As a result, MU-ARQ needs at least two additional packets to complete transmission.

We illustrate now that we can further reduce the number of necessary transmissions and send only 1 coded packet that simultaneously serves the needs of d_1 to d_3. The main idea is to *aggressively exploit the coded packet* $[X_2 + Y_1]$ *overheard by* d_3. That is, we can broadcast a coded packet $[X_2 + Y_1 + Z_3]$. If d_1 receives such packet, d_1 can *decode* the desired X_2 packet by subtracting Y_1 and Z_3 based on d_1's existing knowledge about Y_1 and Z_3. If d_2 receives such packet, d_2 can decode the desired Y_1 packet by subtracting X_2 and Z_3 based on d_2's existing knowledge about X_2 and Z_3. For d_3, d_3 has neither the knowledge about X_2 nor the knowledge about Y_1, which is the reason why MU-ARQ chooses not to mix the three packets together in the first place. However, we note that although d_3 does not know the individual packets X_2 and Y_1, d_3 did overhear a coded packet $[X_2 + Y_1]$. Therefore, If d_3 receives the new coded packet $[X_2 + Y_1 + Z_3]$, d_3 can still decode the desired Z_3 by *subtracting the previously heard coded packet* $[X_2 + Y_1]$ from the new received packet. In this way, we *recoup the benefits of overheard coded packets that are not immediately decodable.* Therefore, the coded packet $[X_2 + Y_1]$ not only serves multiple destinations simultaneously (the inter-flow coding savings) but also enables / creates new coding opportunities of *mixing 3 flows* for the later time slots. This is not possible for a passive inter-flow scheme like MU-ARQ that discards the overheard coded packets.

We have observed the impact of this inherent limitation of MU-ARQ in Figure 1. The throughput efficiency of MU-ARQ is *strictly lower* compared to our theoretically optimal ECR protocol, in particular for small channel success probability p. For example, with $p = 0.2$, the theoretically optimal throughput efficiency is 40% higher than MU-ARQ's (0.7 vs 0.5).

In summary, we have learned **Lesson 1: Discarding overheard coded packets that are not immediately decodable can lead to suboptimal performance. We thus need a coding scheme that *recoups* the coding benefits of coded packets that are not immediately decodable.**

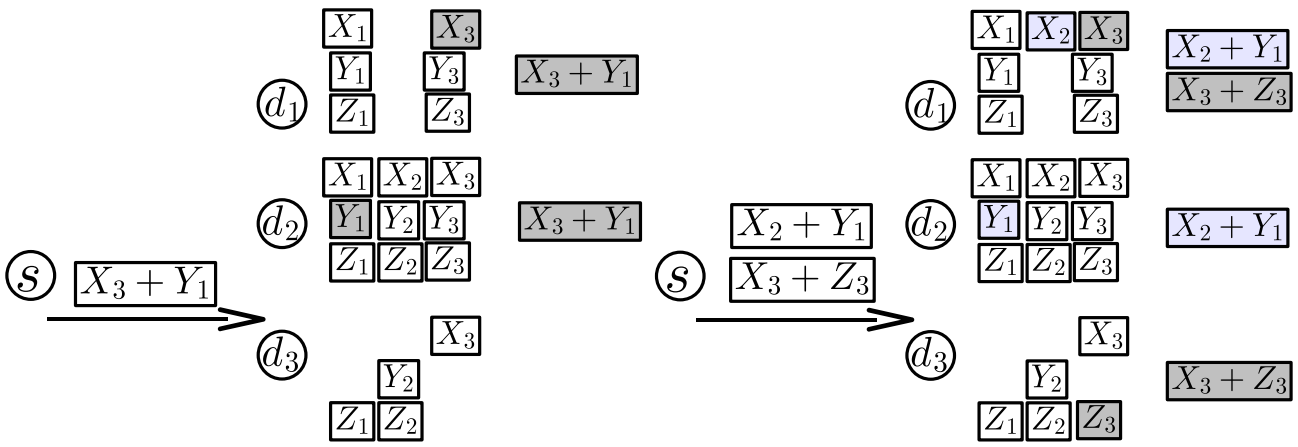
3.2 Practical limitations of MU-ARQ

In their analysis, the authors in [14] assume an infinitely large batch size that allows the AP to always make optimal decisions about which packets to code together.[2] In practice, with a finite batch size, a second important issue arises:

Issue 2: The performance of MU-ARQ with a finite batch size is sensitive to the decision of which coded packet to send in each time slot. Go back to our example in Figure 2. In Section 3.1, we assumed that the AP first sends $[X_3 + Z_3]$ and then $[X_2 + Y_1]$. In Fig. 2(b) we note that there is actually one other coding opportunity $[X_3 + Y_1]$ in addition to the previous choices $[X_3 + Z_3]$ and $[X_2 + Y_1]$. What if the AP decides to send $[X_3 + Y_1]$ first (instead of the previous choices)? Also suppose that both d_1 and d_2 receive $[X_3 + Y_1]$ and use it to decode X_3 and Y_1, respectively. After the first packet, the new overhearing pattern becomes Figure 4(a).

Now d_2 has received all three Y packets; d_1 only needs to receive one more packet X_2; d_3 only needs to receive one more packet Z_3; but X_2 and Z_3 *cannot be coded together* since X_2 is not overheard by d_3. As a result, we need at least two more time slots to complete transmission. The total number of necessary transmissions is thus

[2] In spite of that, we have seen that by discarding coded packets that are not immediately decodable, MU-ARQ cannot achieve optimal performance.



(a) MU-ARQ sending $[X_3 + Y_1]$ first.

(b) MU-ARQ sending $[X_3 + Z_3]$ and $[X_2 + Y_1]$ first.

Figure 4. The impact of different choices of coded packets.

$1 + 2 = 3$ when we send $[X_3 + Y_1]$ first. In contrast, suppose we use the original choices $[X_3 + Z_3]$ and $[X_2 + Y_1]$ and they are heard by $\{d_1, d_3\}$ and by $\{d_1, d_2\}$, respectively; see Figure 4(b) for the overhearing pattern table. In this case, *all three clients d_1 to d_3 can decode all the desired X, Y, and Z packets after two transmissions.* The decision of not sending $[X_3 + Y_1]$ first thus saves 1 transmission.

If we assume that the underlying broadcast channel is always noiseless, such a coding decision problem can be cast as an NP-hard integer programming problem.[3] In practice, the wireless channel is not perfect and each broadcast packet is likely to be heard randomly by some but not by all destinations. This makes the coding decision of MU-ARQ far from optimal even when using the optimal integer programming solver, and does not let MU-ARQ realize the full potential of inter-flow coding especially for the scenarios with high loss rates.[4] We have thus learned **Lesson 2: We need a coding scheme *insensitive* to the decision of which coded packet to send first.**

Finally, one more issue with MU-ARQ in practice is **Issue 3: The entangled interplay between recouping the overheard coded packets and the code design/scheduling.** It is worth noting, as is briefly mentioned in [14] without exploration, that it is possible to modify MU-ARQ to *recoup the overheard coded packets* in an attempt to alleviate Issue 1. However, any naive modification of MU-ARQ to recoup the coding opportunities discussed in Issue 1 will significantly exacerbate Issue 2, i.e., the coding decision problem (see the discussion of [20]). Thus, we have **Lesson 3: in designing a practical protocol, we need a *systematic* method to address Issues 1 and 2 simultaneously.**

4 ECR: Main ideas

Our proposed ECR protocol successfully solves the above issues of MU-ARQ/ER in an efficient and effective manner by introducing three novel design components: (i) Batch-based operations, (ii) A systematic phase-based coding decision policy, and (iii) The smooth combination of inter-flow coding and intra-flow coding. ECR not only enables practical implementation but also admits provably optimal theoretical performance, especially when

[3] It is worth noting that we deliberately construct our example such that there is no opportunity of mixing 3 flows together, which allows us to focus on the challenges of the coding/scheduling problem in the simplest case when there are only coding opportunities of mixing 2 flows. In practice, when there are opportunities of mixing 2 and 3 flows, the AP also needs to decide whether to send 2-flow-coded packets first or to send 3-flow-coded packets. The combinatorial problem becomes highly non-trivial.

[4] Indeed, in [21], ER with any of the two heuristics performs almost as well as when using an exhaustive search strategy.

there is a sufficiently large number of clients in the system. In this section, we discuss the main ideas in the design of ECR, and how they address the issues with MU-ARQ/ER. We then give a detailed description of the protocol in Section 5.

In ECR, the AP transmits packets using 802.11 broadcast and clients store all the overheard packets for a limited time (equal to the duration of a batch, see Section 4.1.) Clients periodically send back cumulative ACKs (similar to those used in ER) reporting to the AP which coded and uncoded packets they have received in the past.

4.1 The Batch-Based Approach

To realize the full potential of inter-flow coding, ECR (i) recoups the overheard coded packets that are not immediately decodable, and (ii) enables a new efficient code scheduling algorithm that is throughput-optimal under realistic channel models. *These two problems are inseparable.* On one hand, different ways of recouping benefits of the overheard coded packets lead to different new coding opportunities; this affects the corresponding scheduling policy. On the other hand, the scheduling/coding policy decides which packet will be sent and thus different policies result in different overheard packets in the system. Note that it is generally more beneficial to wait for more coding opportunities before starting inter-flow coding. However, as it is impractical to wait infinitely for new coding opportunities, we use a batch-based design that curtails the delay impact and also regulates the associated header size for each coded packet.

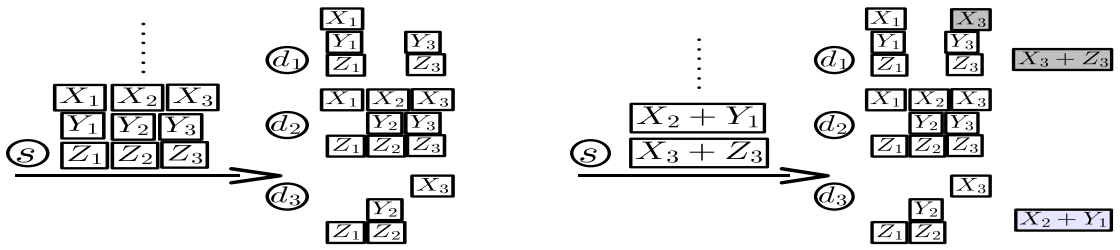
Batch-based operation: Each flow i has N_i packets to transmit in each batch. If there are M flows to be served by the AP, then an *inter-flow batch* contains $\sum_{i=1}^{M} N_i$ packets. The protocol will intelligently decide how to mix the $\sum_{i=1}^{M} N_i$ packets. Our goal is to finish the transmission of the N_i packets for each flow in the shortest amount of time. Note that simply performing random linear network coding on all $\sum_{i=1}^{M} N_i$ packets (i.e., the approach used in [19] for multihop wireless networks) is not efficient as all clients need to receive all $\sum_{i=1}^{M} N_i$ packets before decoding is possible, which takes an excessive amount of time.

4.2 Addressing Issue 1: The Novel Concept of Phases

A critical observation of recouping the benefits of overheard coded packets is that for any given packet, the number of overhearing clients is monotonically increasing. For example, suppose a packet X intended for d_1 is heard by d_2, and we retransmit X for the second time. This time X is received only by d_3. Even though d_2 does not receive X in the second time, d_2 still knows X from the first transmission. Therefore, X is now overheard by both d_2 and d_3. The number of clients overhearing X thus increases monotonically over time. Similar statements hold for inter-flow coded packets as well. That is, if a coded packet is created by mixing two flows, then the overheard coded packet can only create new coding opportunities for mixing > 2 flows.[5] In our example of Figure 3(b), the coded packet $[X_2 + Y_1]$ is created for mixing 2 flows. Once $[X_2 + Y_1]$ is overheard by d_3, it can be used for mixing 3 flows as illustrated in Figure 3(b). The number of flows it participates thus increases monotonically.

The above observation prompts the following phase-based design. We define a *k-flow packet* as a packet that mixes exactly k flows together, k ranging from 1 to M. Any k-flow packet thus serves k clients simultaneously. If a k-flow packet is heard by a client other than the intended k clients, it is not immediately decodable. As discussed previously, ECR will recoup this kind of packets and use it as side information when mixing over $> k$ flows in the future. Since overhearing a k-flow packet may only create a new opportunity for mixing h flows with some $h > k$, an optimal solution is to send out k-flow packets first, and hopefully this will create new opportunities for mixing h flows together where $h > k$. Based on this reasoning, we divide an inter-flow batch into M phases. Each batch

[5] Intuitively speaking, for each packet created by mixing two flows, the corresponding *ingredient packets* are native packets overheard by exactly one other destination. Once the ingredient native packet is overheard by at least one more destination, then such native packet is overheard by > 1 destinations, and can thus be used to generate packets that mix > 2 flows.



(a) The Phase 1 Operation

(b) The Phase 2 operation

Figure 5. Phased-based operation of the proposed algorithm.

goes through Phases 1 to M in sequence. During Phase k, the AP only sends out k-flow packets; overhearing these packets can create new coding opportunities of mixing $h > k$ flows in the future phases.

An example of phase-based operations: We again use a 3-client scenario as our running example, see Figure 5. Each inter-flow batch contains $N_1 + N_2 + N_3 = 9$ packets. In Phase 1 (see Figure 5(a)), the AP transmits packets mixing only 1 flow, i.e., no inter-flow coding is performed. As can be seen, some Phase 1 packets successfully reach the intended client while some do not. The latter are called the overheard packets, and in our example, X_2, X_3, Y_1, and Z_3 are overheard packets. These overheard Phase 1 packets (or equivalently the overheard 1-flow packets) will later be used in Phases 2 and 3. In our example, X_2, X_3, and Y_1 will be used in Phase 2 while Z_3 will be used in Phase 3.

After spending some time in Phase 1, AP switches to Phase 2 based on the feedback from the clients. In Phase 2, AP starts to send packets mixing exactly 2 flows, see Figure 5(b). In our example, based on the overhearing record, Phase 2 sends packets like $[X_3 + Z_3]$ and $[X_2 + Y_1]$ that mix the overheard Phase-1 packets. It is worth noting that $[X_3 + Z_3]$ mixes two packets $[X_3]$ and $[Z_3]$, both of which are overheard packets transmitted in Phase 1. Similarly, the Phase 2 packet $[X_2 + Y_1]$ is constructed from overheard packets in Phase 1.

Again, after spending some time in Phase 2, AP switches to Phase 3 based on the feedback from the clients. In Phase 3, AP sends packets mixing exactly 3 flows. One such example is the $[X_2 + Y_1 + Z_3]$ packet described in Figure 3(b). This new $[X_2 + Y_1 + Z_3]$ packet mixes 3 flows. The way we create it is by mixing two packets $[X_2 + Y_1]$ and $[Z_3]$. Again, $[Z_3]$ is an overheard packet transmitted in Phase 1. $[X_2 + Y_1]$ is an overheard packet transmitted in Phase 2, which is now recouped in the ECR protocol (in contrast with being discarded in ER). The Phase 3 packet $[X_2 + Y_1 + Z_3]$ is indeed constructed by overheard packets in the previous Phases k, $k < 3$. By monotonically progressing from Phases 1 to 3, it is guaranteed that we create the maximum amount of k-flow coding opportunities for each individual phase.

4.3 Addressing Issue 2: Intra-flow Coding Within A Single Phase

The phase-based operation addresses the question whether to send a packet mixing k flows first or to send a packet mixing h flows first for $k < h$ as we always give high priority to the k-flow packet with a smaller k. However, as discussed in the example of Figure 4, even when we are only sending packets mixing the same number of flows, it is still critical to decide which packet to be sent earlier. We observe that the problem of deciding which packet to send under a noisy broadcast channel model is essentially identical to the packet forwarding problem in the original *opportunistic routing* protocol ExOR [4]. Motivated from the success of using *random intra-flow coding* to efficiently solve the packet forwarding problem (the MORE protocol [5]), ECR uses random intra-flow coding to simplify the packet selection decision, without resorting to complicated integer programming solvers or simplified suboptimal heuristics.

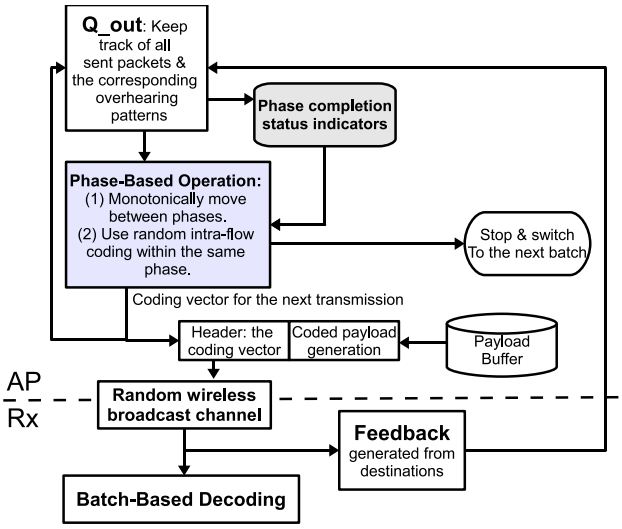


Figure 6. ECR flow chart.

5 Implementation

5.1 Protocol Overview

A flow chart of the protocol operation at the AP and the client is given in Figure 6.

The AP maintains M buffers B_i, $i = 1, \cdots, M$ (jointly shown as *Payload Buffer* in Figure 6); the i-th buffer stores the payloads of the N_i packets for the current batch. To fully exploit/recoup all possible coding opportunities, the AP also needs to keep track of all data packets in the air. For that purpose, the AP keeps a queue Q_{out} that stores the inter-flow coding vectors (not the payloads) of all the outgoing packets for the current batch. Each inter-flow coding vector $\mathbf{v}$ stores the coding coefficients for the $N_{\text{total}} \stackrel{\Delta}{=} \sum_{i=1}^{M} N_i$ packets of the M flows, i.e., it is a row vector of size N_{total}. We set the maximum allowable number of vectors in Q_{out} to be $10 \times \left(\sum_{i=1}^{M} N_i\right)$. Each coding vector is associated with an *overhearing bit map* and a *creation bit map*, each of size M bits, and a unique 2-byte sequence number. The overhearing bit map records which client has overheard this packet. Since each outgoing coding vector/packet is created by mixing a subset of M flows, the creation bit map of an inter-flow vector is used to indicate which flows were used to generate this vector. The unique sequence number for each inter-flow coding vector is to facilitate cumulative feedback as the client can simply send back a list of sequence numbers to acknowledge which packets it has received.

The Q_{out} buffer is used by the phase-based operation to make the coding decision, which starts from Phase 1 and ends in Phase M. When the phase-based operation is in Phase k, the AP generates a coded k-flow packet and broadcasts it whenever there is a transmission opportunity. More explicitly, the phase-based operation generates the inter-flow coding vector for the next to-be-transmitted packet. The coding vector is then used by a separate routine to construct the coded payload from the payload of native packets stored in the payload buffer.

As discussed in Section 4, the monotonic progression from Phase 1 to Phase M gives higher priority to coding over a smaller number of flows. packet has the potential of creating new coding opportunities of mixing h flows in a future Phase h with $h > k$. On the other hand, we also want to quickly move to higher phases so that we can code over a large number of flows and thus capitalize the largest amount of coding benefits. To that end, we need a new component: the **Phase Completion Status Indicators (PCSIs)**. As already mentioned, when the AP receives a feedback packet from a client, the AP will update the overhearing status of Q_{out}. A separate process then takes the updated Q_{out} as input to compute the PCSIs. The PCSIs are then used by the phase-based operation

to decide when to switch from Phase k to Phase $(k+1)$, or to decide whether we have finished the last Phase M and are ready to move to the next inter-flow batch. When we move to the next inter-flow batch, we clear the buffer of Q_{out} and replace the payload buffer with the payloads from the new inter-flow batch.

The clients store all overheard packets. Throughout the transmission, each client periodically sends back ACKs that tell the AP which coded and uncoded packets it has received in the past. Since each packet is associated with an inter-flow coding vector, one can view that the periodic feedback tells the AP which coding vectors have been received by which client.

Unlike the MU-ARQ and ER, ECR uses **batch decoding** to extract the benefits of network coding in a similar way as the MORE protocol. More explicitly, each client collects all the packets it has heard and performs decoding at the end of the current inter-flow batch. When a client receives a packet belonging to a new batch, it removes from its memory all the overhearing packets of the previous batch.

Next we describe in detail the main components of ECR.

5.2 Phase-Based Operation

In the beginning of each inter-flow batch, we add N_{total} *elementary basis vectors* to Q_{out}. That is, if we concatenate the elementary row vectors vertically, we have an $N_{\text{total}} \times N_{\text{total}}$ identity matrix after initialization. All the overhearing bit maps are set to all-zero. For an elementary row vector corresponding to the i-th client, its i-th bit of the creation bit map is set to one and all other creation bits are set to zero. The creation map thus indicates that this elementary coding vector contains the information from flow j. All the sequence numbers are set to 0. Note that the sequence number 0 is reserved only for the N_{total} row vectors in the initialization. All other coding vectors will have a unique sequence number that is not 0.

We use $[M]$ to denote the integer set $\{1, 2, \cdots, M\}$. The phase-based operation maintains $(2^M - 1)$ floating-point variables a_S indexed by S. Each subscript S is a non-empty subset of $[M]$. For example, when $M = 2$, we have $(2^M - 1) = 3$ different non-empty subsets[6] $\{1\}$, $\{2\}$, and $\{1, 2\}$. Therefore, we have three floating-point variables $a_{\{1\}}$, $a_{\{2\}}$, and $a_{\{1,2\}}$. We use $|S|$ to denote the number of elements in S. The phase-based operation also maintains a register K, which records the current phase index.

Initialization of the path-based operation: In the beginning of each inter-flow batch, let $a_S \leftarrow 0$ for all non-empty set $S \subseteq [M]$. Let $K \leftarrow 1$.

Normal operation in Phase K:

Step I — *Choosing the flows to be coded together:* Whenever there is a transmission opportunity and the AP is in Phase K, the AP will mix K flows together. More explicitly, for any given non-empty subset $S \subseteq [M]$ that has $K = |S|$ elements, the AP can mix together all flows $i \in S$ in Phase K. The first key question is thus how to choose a subset S of size K for the AP to code the corresponding flows together. In ECR, the module "Phase completion status indicators" will output a non-negative integer d_S for each subset S. The larger the d_S is for a given S, the more important it is to code the flows in S. To perform tie-breaking between different S, every time we can transmit a packet, we use the following subroutine to choose S:

1: Consider all non-empty subset $S \subseteq [M]$ satisfying both (i) the size is K, and (ii) the corresponding d_S generated from PCSIs is non-zero.
2: Among those S, choose the S^* with the largest a_S value.
3: Let $a_{S^*} \leftarrow a_{S^*} - \frac{1}{d_{S^*}}$.

[6]The number of variables grows exponentially with respect to M, and the suitable value of M depends on the modern microprocessor capabilities. The larger the M value, the higher the throughput. In our implementation, we only use $M = 4$ (using 15 variables). Our results show that mixing only 4 flows together in an efficient way already outperforms the state-of-the-art ER protocol, which mixes all flows together.

Intuition: Using the auxiliary a_S variables, the frequency of selecting a subset S will be proportional to the corresponding importance indicator d_S.

Step II — *Generate a new coding vector:* Once the S^* is determined, use the following subroutine to generate a new inter-flow coding vector $\mathbf{v}$, which is a row vector of dimension N_{total}. We need the following definition for the subroutine.

Definition 1 *An inter-flow coding vector* $\mathbf{v}$ *in* Q_{out} *is* compatible *to a non-empty subset* $S \subseteq [M]$ *if both the following two conditions are satisfied: (i) In the creation bit map of* $\mathbf{v}$*, all the bits outside* S *are zero. (ii) For all* $i \in S$*, it is either that the* i*-th bit of the creation map is 1, or that the* i*-th bit of the overhearing map is 1.*[7]

In other words, $\mathbf{v}$ is compatible to S if the subset S "covers" the creation bit map; and jointly the creation plus the overhearing maps "cover" the subset S. The subroutine that generates an outgoing coding vector $\mathbf{v}_{\text{out}}$ is as follows.

1: $\mathbf{v}_{\text{out}} \leftarrow 0$ is initialized as a row vector of length N_{total}.
2: **for** all inter-flow coding vectors $\mathbf{v}$ in Q_{out} that are *compatible* to the chosen subset S^* **do**
3: Select a coding coefficient $c_{\mathbf{v}}$ randomly from $\mathsf{GF}(2^4)$.
4: $\mathbf{v}_{\text{out}} \leftarrow \mathbf{v}_{\text{out}} + c_{\mathbf{v}}\mathbf{v}$.
5: **end for**

Namely, the final output $\mathbf{v}_{\text{out}}$ is the random summation of all the inter-flow coding vectors compatible to S^*. We use this $\mathbf{v}_{\text{out}}$ as the inter-flow coding vector for the to-be-transmitted packet. Store the new $\mathbf{v}_{\text{out}}$ back into Q_{out} (see the flow chart Fig. 6). Set the overhearing map to 0. For all $i \in [M]$, set the i-th bit of the creation map to 1 or 0 depending on whether $i \in S^*$ or not. Append to $\mathbf{v}_{\text{out}}$ a unique 2-byte sequence number.

Intuition: In the beginning of Phase 1, each S^* must contain only one element. Say $S^* = \{1\}$. Then we will select the inter-flow coding vectors in Q_{out} with the 1st bit of the creation map being 1. Note that during the initialization of Q_{out}, those initialization vectors corresponding flow 1 have the 1st bit of the creation map being 1. Therefore, during Phase 1, when $S^* = \{1\}$, the AP sends coded bits that only mix the packets from flow 1.

Continue our running example in Figure 5. At the end of Phase 1, the overhearing pattern is described in Fig. 5(a) (an identical example is described in Fig. 2(b) as well). Since the $[X_3]$ intended for d_1 is not heard by d_1 but overheard by d_2 and d_3, this packet will have the 1st creation bit being 1 and the 2nd + 3rd overheard bits being 1. Therefore, $[X_3]$ is compatible to the subset $S = \{1, 3\}$. Similarly, the $[Z_3]$ packet intended for d_3 is not heard by d_3 but overheard by both d_1 and d_2. Then $[Z_3]$ is compatible to $S = \{1, 3\}$ as well. As a result, when we choose $S^* = \{1, 3\}$ in Phase 2, the AP will mix $[X_3]$ with $[Z_3]$ as illustrated in Fig. 5(b).

At the end of Phase 2, the overhearing pattern is described in Figure 5(b) (a similar example is described in Figure 3(b) as well). Consider the coded $[X_2 + Y_1]$ packet that was created for serving both d_1 and d_2. Therefore, both the 1st and the 2nd bits of the creation map are 1. Since this coded packet is also overheard by d_3 (see Figure 5(b)), the 3rd bit of the overhearing map is 1. Therefore, this coded packet $[X_2 + Y_1]$ is compatible to the subset $S = \{1, 2, 3\}$. Similarly, a $[Z_3]$ packet intended for d_3 is not heard by d_3 but overheard by both d_1 and d_2. Then this packet packet is compatible to $S = \{1, 2, 3\}$ as well. As a result, when we choose $S^* = \{1, 2, 3\}$ in Phase 3, the AP will mix $[X_2 + Y_1]$ with $[Z_3]$ as illustrated in Figure 3(b). *The overheard coded packet* $[X_2 + Y_1]$ *is successfully recouped by the proposed algorithm.*

In addition to choosing compatible vectors to recoup the overheard coded packets, another key ingredient is the *random intra-flow coding component* in this algorithm. That is, instead of deciding which compatible packets to be mixed together, we *mix all compatible packets by random network coding*. Consider the running example in Figure 2. Suppose after sending out 9 uncoded packets, the overhearing patterns are described in Fig. 2(a). The

[7] It is possible that both the i-th bits of the creation map and the overhearing map are one.

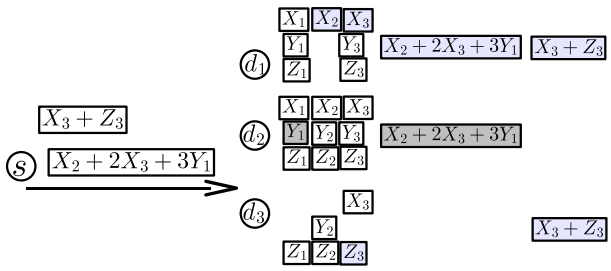


Figure 7. The benefits of using random intra-flow coding.

discussion of Issue 2 in Section 3.2 shows that the decision whether to send $[X_3 + Y_1]$ or $[X_2 + Y_1]$ first will affect the performance. We use random coding to solve this problem.

Suppose the AP, currently in Phase 2, chooses $S^* = \{1, 2\}$, i.e., the AP decides to mix some X and Y packets. Only three packets are compatible $S^* = \{1, 2\}$. That is, packet $[X_2]$, generated for d_1, is heard by d_2 and is thus compatible to S^*. Packet $[X_3]$, generated for d_1, is heard by both d_2 and d_3 and is thus compatible to S^*. Packet $[Y_1]$, generated for d_2, is heard by d_1 and is thus compatible to S^*. Instead of deciding whether to mix X_2 and Y_1 first, or to mix X_3 and Y_1 first, we simply mix all of them together by random linear network coding. Namely, we transmit $[c_1X_2+c_2X_3+c_3Y_1]$ for some randomly chosen coefficients c_i. Figure 7 illustrates this scenario when we choose $c_1 = 1$, $c_2 = 2$, and $c_3 = 3$. With the randomly generated $[X_2 + 2X_3 + 3Y_1]$ (noting the intra-flow mixing of X_2 and X_3), destination d_2 can still recover Y_1 by its existing knowledge about X_2 and X_3. On the other hand, d_1 can decoder neither X_2 nor X_3. Instead of individual packets, d_1 can only decode the mixture $[X_2 + 2X_3]$. When the next coded packet $[X_3 + Z_3]$ is sent, d_1 can decode X_3, which in turn can be used to decode X_2 from the mixture $[X_2 + 2X_3]$. Random intra-flow coding thus achieves the optimal performance without making the difficult decision whether to send $[X_2 + Y_1]$ first or $[X_3 + Y_1]$ first.

Step III — *Construct the coded payload and send the coded packet:* Based on the new inter-flow coding vector $\mathbf{v}_{\text{out}} = (v_1, \cdots, v_{N_{\text{total}}})$, we can assemble the coded payload by summing up the native payloads (stored in the payload buffer) with the corresponding coefficients $v_1, \cdots, v_{N_{\text{total}}}$. The final outgoing coded packet contains the 2-byte sequence number, the new inter-flow coding vector, the coded payload, and the creation bit map. The inclusion of the creation bit map serves two purposes: first, it informs a client whether the coded packet includes a packet intended for that client. This information is used in the decoding process, describe in Section 5.4. Second, we leverage the creation bit map to reduce the size of the packet header, in particular the part occupied by the inter-flow coding vector. When sending a k-flow packet, $k < M$, we only include in the packet header the segments of the inter-flow coding vector corresponding to the chosen k flows. The clients use the creation bit map to reconstruct the whole inter-flow coding vector by filling the segments corresponding to the remaining $(M - k)$ flows with 0's.

Remark: The coding vector will be used for decoding as in most batch-based network coding schemes. The 2-byte sequence number facilitates the feedback from the destination. Note that the sequence number is defined based as a sequence number within the same inter-flow batch, not within the same flow. Therefore, each client periodically sends back a list of sequence numbers acknowledging which coded/uncoded it has received without the need to distinguish to which flows the packets belong. Upon the receipt of the list of ACK'ed sequence numbers, the Q_{out} updates the corresponding overhearing map accordingly.

5.3 Phase Completion Status Indicators

An important component of ECR is the PCSI computation, which decides whether the AP can stop the current Phase k and move to Phase $(k + 1)$. Note that the overhearing maps of the vectors in Q_{out} are updated only

after receiving an ACK from the destinations. Therefore, we only need to run the following routine during the initialization of Q_{out} for each inter-flow batch (see Section 5.2), and when the AP receives a cumulative ACK from a client. The computation of the PCSIs is as follows:

There is one PCSI for each non-empty subset $S \subseteq [M]$, which is denoted by d_S. For any given S, the computation of d_S is carried out as follows.

1: $d_S \leftarrow 0$.
2: **for** all $i \in S$ **do**
3: Consider all vectors in Q_{out} that satisfy at least one of the following two conditions: **Cond. 1:** it is received by d_i (by checking the i-th overhearing bits); and **Cond. 2:** it is *compatible* to at least one $S' \subseteq [M]$ satisfying $|S'| > |S|$. [8]
4: Project all these vectors onto flow i.
5: Compute the rank of the projected vectors. Let r_1 denote the computed rank and store it for later use.
6: Consider all vectors in Q_{out} that satisfy at least one of the following three conditions: **Cond. 1**, **Cond. 2** as defined in Line 5.3, or **Cond. 3:** it is *compatible* to S.
7: Project all these vectors onto flow i.
8: Compute the rank of the projected vectors. Let r_2 denote the computed rank.
9: $d_S \leftarrow d_S + (r_2 - r_1)$.
10: **end for**

Intuition: The PCSIs answer when we can switch to the next phase. Suppose we are in Phase k and try to mix k flows in S ($|S| = k$). If we know that the "benefit" of mixing S flows is no different than that of mixing S' flows for some $S' \subseteq [M]$ satisfying $|S'| > |S|$, then it means that mixing S flows is redundant, since the same benefit can be achieved by mixing S' flows. Since mixing S' flows can serve $|S'|$ destinations simultaneously (recall that $|S'| > |S| = k$) rather than serving k flows as by mixing S flows, we definitely want to start mixing S' flows rather than wasting more time on mixing S flows only. The rank r_2 computed by PCSIs represents the benefits of "staying in the current Phase k" while the rank r_1 represents the benefits of "moving out of the current Phase k and focusing on the higher-phase S' flows." When the difference is zero, it means that it is time to move to the next phase, or equivalently to abandon mixing S packets in the future. Note that d_S is the sum of $r_2 - r_1$ for all $i \in S$. The reason is that mixing S flows potentially benefits all $i \in S$. Therefore, we compute the benefit $r_2 - r_1$ for each individual flow i and sum them up for the computation of d_S.

Deciding whether to move to the next phase: After the computation of d_S for all non-empty subsets $S \subseteq [M]$, the AP needs to decide whether to move to the next batch or not. Suppose the current batch is k for some $1 \leq k \leq M-1$. We check all $S \subseteq [M]$ with $|S| = k$. If all those S have $d_S = 0$, then it means staying in Phase k is redundant and we move to Phase $(k+1)$. Otherwise, stay in Phase k. When $k = M$, there is no next phase to move to. We stay in Phase M until all destinations have sent feedback acknowledging that the decoding of the inter-flow batch is successful. Once all destinations have acknowledged the current inter-flow batch, we move to the new batch.

5.3.1 Reducing Complexity

The computation of $(2^M - 1)$ PCSIs every time the AP receives a cumulative ACK from a client is the computationally heaviest component of ECR. However, the computation can be made more efficient by noticing that there are a lot of repeated computation. For example, any vector in Q_{out} that satisfies **Conds. 1** and **2** in Line 5.3 for some i and S_1 must also satisfy **Conds. 1** and **2** for the same i and all other S_2 satisfying $|S_2| = |S_1|$. Therefore, the computation of r_1 for the pair (i, S_1) must be identical to that for the pair (i, S_2). After optimizing the

[8] When considering $S = [M]$, since there exists no $S' \subseteq [M]$ satisfying $|S'| > |S| = M$, **Cond. 2** becomes a null condition. No coding vector can satisfy **Cond. 2**. Therefore Line 5.3 collapses to "considering all vectors satisfying **Cond. 1**."

operations of the PCSI to remove all duplicated computation, the complexity of computing the PCSI can be made comparable to that of ER.

Optimized PCSI Computation Algorithm. The new, optimized PCSI computation algorithm is as follows.

1: **INPUT:** s receives a list of packet sequence numbers from a destination d_i.
2: Update the i-th bit in the overhearing map of all the vectors corresponding to the list of input sequence numbers.
3: **OUTPUT:** A list of joint coding vectors for which the corresponding overhearing maps change.
For example, suppose a vector v has overhearing map $(1, 1, 0)$ and destination d_1 ACKs this vector one more time. Since the overhearing map remains the same after the update, we do not output v. For comparison, suppose a vector u has overhearing map $(0, 0, 1)$ and destination d_1 ACKs this vector. Then the overhearing map changes from $(0, 0, 1)$ to $(1, 0, 1)$. u will thus be included in the output.

Based on the list of coding vectors generated from the previous section, we update the following matrices. The following routine needs to be run for all d_1 to d_4 and for all the vectors in the generated list.

Before describing the subroutine, we need to define "sequential downstream subsets of S."

Definition: Given a subset $S \subset [M]$, the sequential downstream subsets of S is a sequence of sets such that the first set is S, and the last set is $\emptyset$. All the sets in the middle are a subset of S. And the sequence follows that if S_1 appears before S_2, then we must have $|S_1| \geq |S_2|$.

For example, if $S = \{1, 3, 4\}$, then the sequential downstream subsets of S is a sequence of eight sets: $\{1, 3, 4\}$, $\{1, 3\}$, $\{1, 4\}$, $\{3, 4\}$, $\{1\}$, $\{3\}$, $\{4\}$, and $\emptyset$.

Note that the choice of the sequential downstream subsets is not unique. For example, when $S = \{1, 3, 4\}$, then the following sequence of sets is also the sequential downstream subsets of S: $\{1, 3, 4\}$, $\{1, 4\}$, $\{3, 4\}$, $\{1, 3\}$, $\{1\}$, $\{4\}$, $\{3\}$, and $\emptyset$. Our subroutine holds for any choice of the sequential downstream subsets.

The subroutine:

1: For each session, say session 1, for each $S \subset [M]$, maintain two matrices $R_{S,1}$ and $R_{S,2}$. There are thus totally 2×2^M matrices maintained for session 1. Similar to what we discussed in the original scheme, $R_{S,2}$ is always larger than $R_{S,1}$. The rank of them are the r_2 and r_1 in the previous description.
Each matrix $R_{S,1}$ (or $R_{S,2}$) is also associated with a flag $b_{S,1}$ (or $R_{S,2}$). There are thus totally 2×2^M flags maintained for session 1.
2: **INPUT:** A joint coding vector v from the generated list. A target destination, say d_1. Since we only focus on the projection of v on d_1, we thus assume that v is the projected vector in the following discussion.
3: Set all the flags to 0.
4: Suppose the overhearing map (the overhearing pattern) of v is S_{hearing} and the creation map is S_{creation}.
5: **if** S contains d_1 (since we focus on d_1) **then**
6: $S_0 \leftarrow [M]$.
7: **else**
8: $S_0 \leftarrow S_{\text{hearing}} \cup S_{\text{creation}}$.
9: **end if**
10: Consider the sequential downstream sets of S_0. For each S in the list of sequential downstream sets of S_0, we perform the following operations. (The sequential downstream sets decide which S to consider first.)
11: **for** each S in the sequential downstream sets of S_0 **do**
12: **if** $b_{S,1} = 0$ **then**
13: **if** $S = S_0$ and $1 \in S_{\text{hearing}}$ **then**
14: Add the (projected) vector v into $R_{S_0,1}$ through Gaussian elimination.
15: **if** v is linear dependent to the existing $R_{S_0,1}$ **then**
16: Set $b_{S_0,2} \leftarrow 1$.
17: Set $b_{S_2,1} \leftarrow 1$ and $b_{S_2,2} \leftarrow 1$ for all S_2 being a strict subset of S_0.

18: **end if**
19: **else if** $S \neq S_0$ **then**
20: Consider the subsets of the form: $S' = S \cup \{j\}$ such that $j \notin S$ but $j \in S_0$.
21: Let $v_{S'}$ denote the vector that are added last to the $R_{S',2}$ *after* finishing Gaussian elimination.
22: Among all $v_{S'}$, let $v^*_{S'}$ denote the one with the largest number of zeros in the left portion of the coordinates.
23: Add the $v^*_{S'}$ into $R_{S,1}$ through Gaussian elimination.
24: **if** $v^*_{S'}$ is linear dependent to the existing $R_{S,1}$ **then**
25: Set $b_{S,2} \leftarrow 1$.
26: Set $b_{S_2,1} \leftarrow 1$ and $b_{S_2,2} \leftarrow 1$ for all S_2 being a strict subset of S.
27: **end if**
28: **end if**
29: **if** $b_{S,2} = 0$ **then**
30: **if** $S = S_0$ and $1 \notin S_{\text{hearing}}$ **then**
31: Add the (projected) vector v into $R_{S_0,1}$ through Gaussian elimination.
32: **if** v is linear dependent to the existing $R_{S_0,1}$ **then**
33: Set $b_{S_0,2} \leftarrow 1$.
34: Set $b_{S_2,1} \leftarrow 1$ and $b_{S_2,2} \leftarrow 1$ for all S_2 being a strict subset of S_0.
35: **end if**
36: **else**
37: Let v_S denote the vector that are previously added to the $R_{S,1}$ *after* finishing Gaussian elimination.
38: Add the v_S into $R_{S,2}$ through Gaussian elimination.
39: **if** v_S is linear dependent to the existing $R_{S,2}$ **then**
40: Set $b_{S_2,1} \leftarrow 1$ and $b_{S_2,2} \leftarrow 1$ for all S_2 being a strict subset of S.
41: **end if**
42: **end if**
43: **end if**
44: **end if**
45: **end for**

High-Level Discussion: The sequential downstream subsets define the sequence of sets to be considered. Depending on whether d_1 has received the vector and on the set $S_{\text{hearing}} \cup S_{\text{creation}}$, we decide where to start the update by choosing S_0. After we have processed S_0, we update the downstream subset S through a chain effect (adding only a Gaussian-elimination-processed vector). If any the new vector is linearly dependent to one subset S, then we shut down all the further update on the subsets $S_2 \subseteq S$ by setting the corresponding flags.

The chain effect says that $R_{S,2}$ depends on $R_{S,1}$ and that $R_{S,1}$ depends on one of the $R_{S',2}$ with $S' = S \cup \{j\}$.

In Section 8, we measure the execution time of the optimized algorithm on our testbed and show that it remains sufficiently low.

5.4 Batch Decoding at the Client

Each receiver uses the same batch-based decoding algorithm. For simplification, we consider only receiver d_1. Rx i for $i > 1$ can be obtained by symmetry.

Initialization at d_1: For each new inter-flow batch, initialize two empty queues: $Q_{\text{rx\_vec}}$ and $Q_{\text{rx\_seq}}$. Each entry in $Q_{\text{rx\_vec}}$ contains an N_{total}-dimensional inter-flow coding vector (row vector) and a coded payload. Each entry in $Q_{\text{rx\_seq}}$ is a 2-byte sequence number.

Shuffle the columns of $Q_{\text{rx\_vec}}$: An important part of the initialization step is to shuffle the columns of $Q_{\text{rx\_vec}}$. The vectors in $Q_{\text{rx\_vec}}$ form a $|Q_{\text{rx\_vec}}| \times N_{\text{total}}$ matrix. We deliberately shuffle the columns such that the columns

corresponding to flow 1 are the last N_1 columns. Ex: for rx 3, we shuffle the columns of flow 3 to the last N_3 columns. All our discussion is based on the shuffled columns.

When Rx 1 **receives a packet:** (1) Put the corresponding sequence number in $Q_{\text{rx\_seq}}$, which will be used when sending out feedback.

(2) Let $\mathbf{v}$ denote the inter-flow coding vector of the received packet. We test the linearly independence of $\mathbf{v}$ with respect to the $Q_{\text{rx\_vec}}$ matrix by Gaussian elimination. If the vector $\mathbf{v}$ is linearly dependent to the $Q_{\text{rx\_vec}}$ matrix, discard the received packet. If $\mathbf{v}$ is linearly independent to the $Q_{\text{rx\_vec}}$ matrix, then we add both $\mathbf{v}$ and the coded payload to $Q_{\text{rx\_vec}}$. That is, perform Gaussian elimination on both the inter-flow coding vector and on the coded payload. After Gaussian elimination, the $|Q_{\text{rx\_vec}}| \times N_{\text{total}}$ matrix is always upper triangular. Note that this is a two-step process. We first test the independence only for the coding vector. Only when the coding vector is linearly independent, then we process the payload, which reduces the computation complexity.

Decode: Consider the rows for which the only non-zero entries corresponding to flow 1. Since we shuffle the columns of flow 1 to the last N_1 columns and since $Q_{\text{rx\_vec}}$ is an upper triangular matrix, those rows must be the last few rows. When the number of those rows is N_1, we perform decoding. That is, we use the last N_1 rows and the corresponding coded payload to decode as if we are performing only intra-flow decoding.

Intuition: The critical step is shuffling the flow 1 columns to the last N_1 columns. In this way, when we perform Gaussian elimination, it will automatically eliminate the "interference" caused by other flows. (The only non-zero entries in the last few rows are all in the columns of flow 1.) As a result, when the number of rows containing only flow 1 information (the last few rows) reaches N_1, we can decode the flow 1 packets from the coded payload in the same way as we perform decoding for intra-flow coding protocols.

5.5 Client Feedback

Clients in ECR send periodic cumulative feedback to inform the AP of their reception status. We distinguish two forms of feedback: before and after decoding a batch. Before decoding a batch, a client sends a reception report similar to the ones used in ER reporting to the AP the most recent $\sum_{k=1}^{M} N_k$ sequence numbers it received/overheard for the current batch in the form of a bitmap (plus a start sequence number). When a client decodes a batch, it sends a special ACK to inform the AP. The AP treats this special ACK as a bit map of sequence numbers that acknowledges *all* packets that have been sent during the batch. Since this special ACK may be lost, from that moment and until the AP moves to the next batch (i.e., until *all* clients decode their current batch), a client in FINISH mode (i.e., having already decoded its current batch) resends periodically the same special ACK to the AP.

6 Correctness Guarantee and Performance Analysis

The ECR protocol is designed with rigorous mathematical foundation. In this section, we outline the corresponding proofs of the correctness and the performance analysis.

6.1 Correctness

Assume the success probability of each link is strictly larger than zero. To show the correctness of the ECR algorithm, we need to prove the following:

- ECR will move from Phase k to Phase $(k+1)$ in a finite amount of time for any $1 \leq k \leq M-1$.
- Once the AP is in Phase M, all destinations will send the final acknowledgment of the entire inter-flow batch within a finite amount of time.

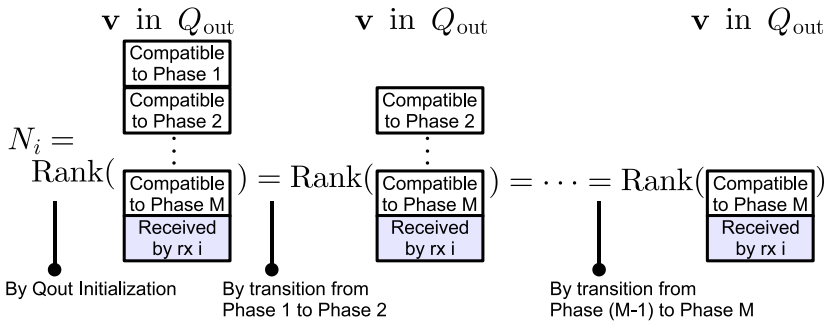


Figure 8. Illustration of the intuition of Phase Completion Status Indicators

That is, ECR proceeds normally and will not get stuck in any phases. Note that the above two statements holds for arbitrary $\mathsf{GF}(2^b)$ (even for binary field $\mathsf{GF}(2)$). That is a critical part as in practical schemes, we cannot rely on the "with high probability $(1-\epsilon)$ for sufficiently large $\mathsf{GF}(2^b)$" statements commonly used in random linear network coding [10].

Proving that AP always moves between phases is straightforward. We notice that whether the AP switches phases depends on $d_S = r_2 - r_1$. For any given S, the created coding vector $\mathbf{v}$ will have all the creation bits in S set to 1. Therefore, if there is any destination j outside S overhears the transmitted $\mathbf{v}$, then such vector will be considered compatible to $S' = S \cup \{j\}$. As a result, the rank r_1 will increase with a non-zero probability. In the end, the rank r_1 approaches r_2 in a finite amount of time. d_S becomes zero in a finite amount of time and the AP moves to the next phase.

The proof of the completion of the inter-flow batch in Phase M is more complicated, which is built around the following propositions.

Proposition 1 *For any time instant of the AP, suppose a packet is generated according to a flow set S. (Currently, the AP is in Phase $|S|$.) Consider one receiver $i_0 \in S$ and fix that i_0. If the i_0-th destination receives that packet, then after the Gaussian elimination performed at the i_0-th destination, the new vector will have zero elements for all columns corresponding to flow-j, for all $j \neq i_0$. Note that after Gaussian elimination, the elements in the columns corresponding to flow-i_0 may or may not be zero.*

Intuition: This proposition says that if for any destination d_{i_0}, $i_0 \in S$, all the "interference" from flow j, $j \neq i_0$ can be canceled at destination d_{i_0}. This similar to the "immediate decodability concept" of COPE. That is, for the COPE protocol, upon the reception of one S-flow packet intended for $i_0 \in S$, d_{i_0} can cancel all the interference and decode the desired packets. The difference for ECR is that now d_{i_0} can cancel all the interference and obtain *one more linearly independent intra-flow coded packet.*

For any destination d_i, we notice that $Q_{\text{rx\_vec}}$ is an upper triangular matrix. Let r_{proj} denote the rank of the submatrix of $Q_{\text{rx\_vec}}$ induced by the columns of flow i. Let r_{lower} denote the number of the last few rows for which all the non-zero entries are in columns of flow i. We have the following theorem.

Proposition 2 *For any time instant, $r_{proj} = r_{lower}$.*

Intuition: This proposition further solidifies the essence of Proposition 1. That is, the "rank" of the projected space on flow i can be fully extracted by Gaussian elimination when focusing on the last few rows for decoding.

Proving that once the AP is in Phase M, ECR moves to the next batch within a finite amount of time. For the following, we call the projection of a space to flow i as the "flow-i space." If the phase completion status

indicator $d_S = 0$, it means that the flow-i space spanned by the vectors compatible to S is fully covered by the flow-i space spanned by vectors that are compatible by S' or are received by destination i. As a result, when we move from Phase k to $(k+1)$, it is guaranteed that the flow-i space of the vectors that are compatible to Phase $(k+1)$ or higher or received by destination i has the same rank as the flow-i space of the vectors compatible to Phase k or higher or received by i, see Fig. 8. On the other hand, all elementary vectors that are stored in Q_{out} during the initialization of Q_{out} are compatible to $\{i\}$. Therefore the flow-i space of vectors compatible to the Phase 1 selection $S = \{i\}$ has full rank N_i, see Fig. 8. As a result, when the AP moves to the final Phase-M, the flow-i space of the vectors that are compatible to $[M]$ or received by i, must have the full rank N_i (Fig. 8). Hence, after spending a finite amount of time in Phase M, the destination d_i will have received vectors such that the corresponding flow-i space has full rank N_i. By Proposition 2, each destination d_i thus can fully cancel the interference of other flow j, $j \neq i$, and recover all N_i native packets. Destination d_i will thus acknowledge the entire inter-flow batch in a finite amount of time. The proof is thus complete.

*Remark:*It is worth noting that the design of ECR is started by taking into account all the practical limitations that have been puzzling the existing protocol designs (see Sections 2.1 and 3.2). Therefore, in contrast with the existing top-down approach (from the theoretic MU-ARQ principle [14] to the practical ER protocol [21]), our bottom-up approach (addressing practical constraints first) ensures that the resulting ECR protocol depends only on very simple feedback-based operations without complicated time-out mechanisms. The most intriguing feature of ECR is that with carefully designed, feedback-based computation at the AP, this bottom-up approach *is also theoretically optimal.* That is, under the same theoretic setting as used in MU-ARQ [14], our practice-oriented *ECR strictly outperforms the overly idealistic MU-ARQ principle and achieves the best poosible throughput for any idealistic/practical schemes one can envision*, as we prove in the following.

6.2 Performance Analysis

For the performance analysis of ECR, we denote underlying finite field of network coding by $\mathsf{GF}(2^b)$. We use the same channel model as when we describe the existing results in Eqs. (1) and 2 of Section 2. Unlike the MU-ARQ, we use a more realistic assumption of periodic feedback (sending feedback every $F > 1$ time slots using a separate channel). By generalizing the *physically-degraded-channel-based* proofs in [8], we obtain the following proposition that upper bounds the best possible η for any transmission scheme one can envision:

Proposition 3 *For any given values of M and F, the throughput efficiency of any scheme must satisfy:*

$$\eta \leq \frac{M}{\sum_{k=1}^{M} \frac{1}{1-(1-p)^k}}. \tag{3}$$

Our ECR protocol achieves the optimal throughput efficiency in the following sense:

Proposition 4 *For any given values of M and F, the throughput efficiency of ECR satisfies*

$$\lim_{N\to\infty} \lim_{2^b\to\infty} \eta = \frac{M}{\sum_{k=1}^{M} \frac{1}{1-(1-p)^k}}. \tag{4}$$

Namely, for sufficiently large N and 2^b, ECR is throughput optimal. The superior performance of ECR for finite N and finite 2^b is verified by simulation and testbed implementation in Sections 7 and 8.

7 Simulation Study

In this section, we compare the performance of ECR against that of ER and 802.11 using the Glomosim simulator [24].

7.1 Methodology

We place an AP in the center of the simulation area and up to 20 clients uniformly on a circle around the AP. To evaluate the performance of the protocols under different loss scenarios, the clients are placed close to the AP and we generate link loss rates in a controlled manner, by artificially dropping packets at each client following a Bernoulli model. Following the evaluation methodology in [21], we consider both homogeneous and heterogeneous loss rates.

Following the evaluation methodology in [21], we consider both homogeneous and heterogeneous loss rates. In the homogeneous case, the loss rates of all links are the same, varied between 10% and 90% in different simulations. In the heterogeneous case, the loss rate for each link is randomly selected between 0 and an upper bound and the upper bound varies from 10% to 90%.

In every simulation, the AP transmits 1500-byte packets for 100 sec. The results shown in the following sections are averages over 10 runs. Note that the standard deviations are very low in all cases except for the heterogeneous loss scenarios in Section 7.3. Hence, in the interest of space and clarity, we do not show them in the rest of the results.

We noticed the performance of ER heavily depends on three parameters: retransmission queue threshold, retransmission queue timeout, and period of cumulative ACKs. Further, the optimal set of values for the three parameters depends on the number of clients. For example, if the AP supports a large number of clients (more than 20), then a retransmission queue threshold of 25 packets may be too small, limiting the coding opportunities. As another example, when the number of clients increases, the number of cumulative ACKs increases, resulting in increased overhead and contention with data packets. On the other hand, reducing the frequency of ACKs, to keep the overhead constant, may reduce the AP's knowledge about the clients' content, which in turn can again reduce the coding opportunities. [21] uses a threshold of 25 packets and a timeout of 250ms and does not discuss the third parameter.

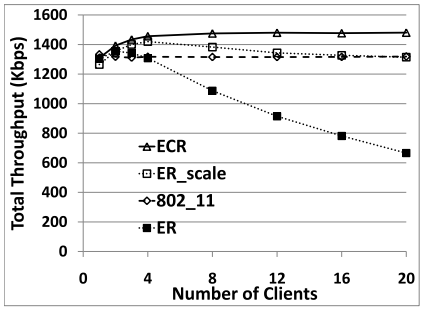
Finally, note that in [21], the authors used for their simulation evaluation a simplified simulator that did not model timing dynamics. Hence, they did not evaluate the actual ER implementation, but a simplified version of the protocol, where a sender sends a constant-size batch of packets at a time and then (after instant feedback from the clients) retransmits lost packets until all of them are received by the destined clients. With this simplified (but not practical) version, the authors were not able to study the interdependence of the three parameters of the actual implementation.

To deal with this complex interdependence among the three parameters, we decided to jointly scale all three parameters proportionally to the number of clients K. We tried three different scaling factors, K, $K/2$, and $K/4$. We also tried different values for the base ACK period (i.e., the ACK period for $K = 2$ clients) for each scaling factor. We do not present these results here due to space limitation. For our final evaluation and comparison with ECR, we use a base ACK period (of 40ms, and a scaling factor of $K/2$, which gave the best performance among different configurations.

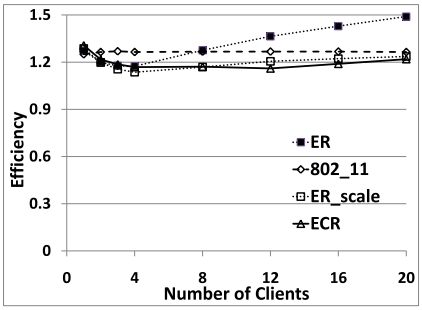
We repeated the same procedure to choose the ACK period for ECR, which is the only parameter that requires scaling in our protocol. We finally chose the same scaling factor for ECR's ACK period, but with a base period of 20 ms. Finally, we use a batch of 48 packets for each flow and a Galois Field $\mathsf{GF}(2^4)$ (half byte).

Evaluation Metrics. We use the following metrics:

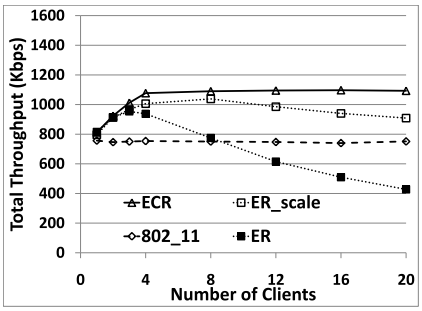
Aggregate Throughput: The throughput per client is defined as the total number of unique packets (excluding duplicates) received for 802.11 and ER, or the total number of decoded packets for ECR, multiplied by the packet



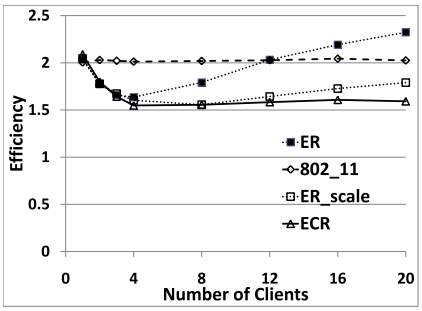
(a) Throughput comparison, 20% loss.



(b) Efficiency comparison, 20% loss.



(c) Throughput comparison, 50% loss.



(d) Efficiency comparison, 50% loss.

Figure 9. Throughput and efficiency comparison for different numbers of clients under homogeneous losses.

size, and divided by the total time required to collect/decode those packets. The aggregate throughput is the sum of the throughputs of all clients.

Efficiency: The ratio of the total number of packets sent by the AP (including retransmissions) (original data packets, and MAC layer retransmissions for 802.11, original data packets, single retransmissions and coded retransmissions for ER, intra-flow and inter-flow coded packets for ECR), over the total number of data packets effectively received for each protocol (e.g. decoded in ER or ECR) by all the clients. The value of this metric is ≥ 1, with larger values indicating lower efficiency.

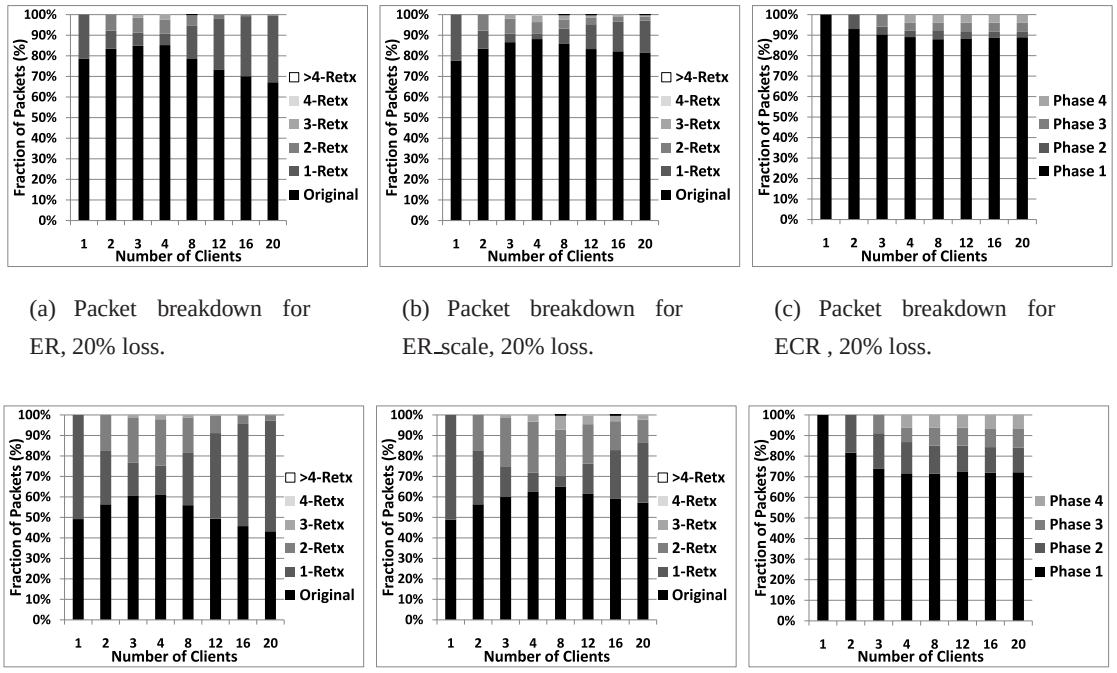
7.2 Homogeneous losses

7.2.1 Varying the number of clients

Performance comparison. Figures 9(a)-9(d) plot the throughput and efficiency of ECR, ER, and 802.11 with a varying number of clients for a loss rate of 20% and 50%. For ER, we plot two versions, the original one and the version where we scale all the parameters with the number of clients, denoted as ER_scale. We make the following observations:

First, we observe that the performance of ER drops dramatically with the number of clients. 802.11 outperforms ER in terms of both throughput and efficiency with more than 4 clients under a 20% loss rate and with more than 8 clients under a 50% loss rate. Note the complete ER protocol was evaluated in [21] with only up to 6 clients.

Second, ER_scale substantially outperforms ER in terms of both metrics and always outperforms 802.11. How-



(a) Packet breakdown for ER, 20% loss.

(b) Packet breakdown for ER_scale, 20% loss.

(c) Packet breakdown for ECR , 20% loss.

(d) Packet breakdown for ER, 50% loss.

(e) Packet breakdown for ER_scale, 50% loss.

(f) Packet breakdown for ECR , 50% loss.

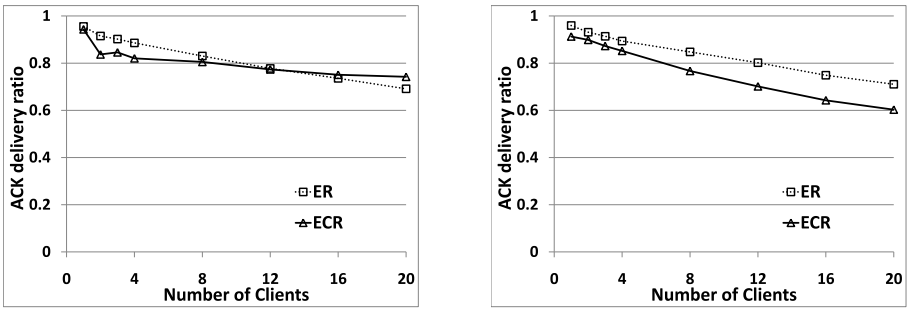
Figure 10. Packet breakdown for different coding schemes under homogeneous losses. For ER and ER_scale, i-Retx corresponds to transmissions mixing together i packets (for i different clients); 1-Retx corresponds to packets retransmitted uncoded. For ECR, Phase i corresponds to inter-flow coded packets of i different clients; Phase 1 include only intra-flow coded packets for a single flow.

ever, its performance also degrades with the number of clients.

Third, ECR outperforms ER_scale in terms of both throughput and efficiency (with the exception of a small number of clients where ER_scale's efficiency is about 3% better). The throughput gain of ECR over ER_scale is as high as 13% under a 20% loss rate and as high as 20% under a 50% loss rate, with 20 clients. Compared to original ER, ECR's throughput gain is much higher, up to 122% and 155% under 20% and 50% loss rates, respectively. Finally, compared to 802.11, ECR's throughput gain is as high as 13% under a 20% loss rate (same as against ER_scale) and as high as 48% under a 50% loss rate. Moreover, the protocol scales well with the number of clients in spite of only allowing to encode packets from groups of 4 flows.

Regarding the efficiency metric, note that with a loss rate L, 802.11 (which retransmits each packet uncoded) needs on average $\frac{1}{1-L}$ transmissions to deliver a packet, i.e., its efficiency is $\frac{1}{1-L}$. This is indeed the case in Figures 9(b), 9(d) – 802.11's efficiency varies from 1.22-1.27 with a 20% loss rate and from 2.00-2.03 with a 50% loss rate. In contrast, by exploiting network coding, ECR achieves a much better efficiency, in particular under high loss rates; ECR's efficiency is lower than 1.59 with a 50% loss rate even with 20 clients. ER_scale's efficiency is also better than 802.11's, but worse than ECR's and, as mentioned before, it degrades with the number of clients, especially under high losses.

Packet breakdown. To understand where the gains of ECR come from, we plot in Figures 10(a)-10(f) a breakdown of coded and uncoded transmissions for ER, ER_scale, and ECR, with a varying number of clients under 20% and 50% loss rates.



(a) 20% loss rate. (b) 50% loss rate.

Figure 11. ACK packet delivery ratio with varying number of clients under homogeneous losses.

In Figures 10(a), 10(d), we observe that as the number of clients increases beyond 4, (i) the number of retransmissions for ER increases; in particular, under a 50% loss rate, the retransmitted packets are more than the original ones for 12 or more clients. (ii) the percentage of coded retransmissions decreases; in the most extreme case, under a 20% loss rate, the fraction of coded retransmissions out of the total number of transmissions is less than 1%, i.e., almost packets are sent out uncoded. This explains the reduced efficiency for ER and its low throughput.

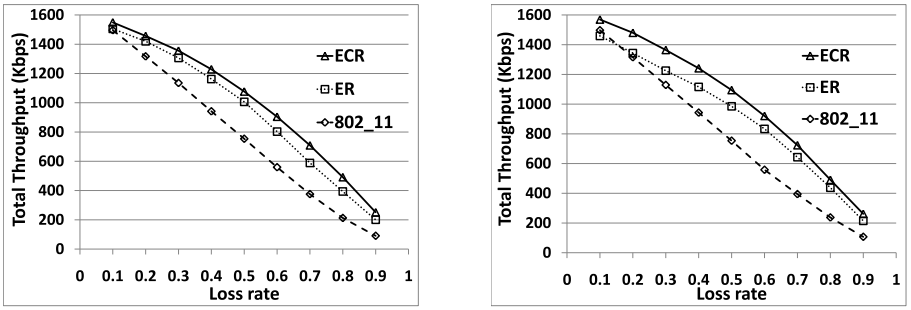
Figures 10(b), 10(e) show that scaling helps ER; the number of retransmitted packets with ER_scale is always lower than the corresponding number with ER in Figures 10(a), 10(d). Also, the fraction of coded packets with ER_scale is always larger than with ER. In spite of this improvement, we observe that the fraction of retransmissions for ER_scale still increases and the coding gain still decreases with the number of clients.

In contrast, Figures 10(c), 10(f) show that the fraction of packets in each phase with ECR remains unchanged with the number of clients. This shows that ECR scales well with the number of clients and explains its superior performance over ER_scale.

Why does the performance of ER_scale drop with the number of clients? Figures 11(a), 11(b) plot the delivery ratio of ACK packets (i.e., the number of ACK packets received at the AP divided by the total number of ACK packets sent by all the clients) for ER_scale and ECR with varying number of clients for loss rates of 20% and 50%. We observe that, in spite of scaling the ACK period with the number of clients and adding jitter before sending the ACKs, the delivery ratio of ACK packets still decreases with the number of clients. The delivery ratio is similar for the two protocols under 20% loss rate and higher for ER_scale under 50% loss rate. Nonetheless, the performance of ER_scale degrades with the number of clients while the performance of ECR remains constant.

However, the impact of ACK losses is much higher in ER (or ER_scale) than in ECR, for the following reasons. ECR uses a batch for each flow and a systematic phase-based transmission and coding strategy that does not depend on any timing dynamics. In a given phase i, the AP transmits linear combinations of a given set of packets and it always mixes together packets from i different flows. Thus an ACK loss may delay the transition to the next phase, but all the extra packets transmitted in the current phase will still combine packets from i flows, thus wasting little bandwidth.

In contrast, ACK losses in ER may have the following consequences: (i) they result in timeout expiration for those packets pending acknowledgment and a burst of retransmitted packets instead of new packets. (ii) they distort the RTT calculation of the next set of original packets; newly transmitted packets will typically have a longer timeout, i.e., they will have to wait longer before they are eligible for retransmission. (iii) they reduce the AP's knowledge of what packets have been overheard by the clients that sent the lost ACKs. (ii) and (iii) combined imply that many retransmitted packets are sent uncoded since the AP either finds no packets from other flows to mix together (due to (ii)) or it does not know whether any packets can be coded together (due to (iii)). This results in reduced efficiency and throughput for ER (and ER_scale) as the number of flows increases.



(a) 4 clients (b) 12 clients.

Figure 12. Throughput comparison with varying loss rates under homogeneous losses.

7.2.2 Varying the loss rate

We now evaluate the performance by varying the loss rate. Since ER_scale performs much better than the original ER without scaling, we will use this version in the remaining of this section and in the next section and we will call it ER for simplicity. Figures 12(a), 12(b) plot the aggregate throughput with ECR, ER, and 802_11, with 4 and 12 clients, respectively, when the loss rate varies from 10% to 90%. We observe that ECR outperforms ER and 802_11 for all loss rates and the improvement is higher with higher loss rates, which also agrees with the theoretical results. With 4 clients (Figure 12(a)), the throughput gain of ECR over ER varies from 3% (with 10% loss rate) up to 26% (with 90% loss rate) and the gain over 802.11 varies from 3.5% up to 178%. With 12 clients (Figure 12(b)), the throughput gain of ECR over ER varies from 7.5% (with 10% loss rate) up to 22% (with 90% loss rate) and the gain over 802.11 varies from 5% up to 143%.

7.3 Heterogeneous losses

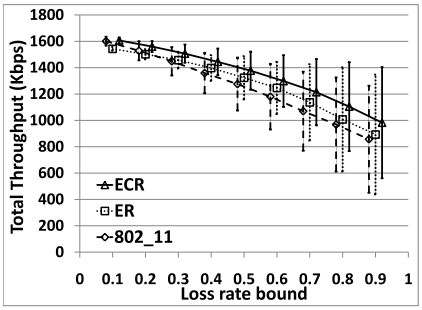
We now consider heterogeneous loss rates. This scenario is closer to the reality, where different clients have different loss rates either because they are located in different distances from the AP or due to multipath fading or because they experience different numbers of collisions (e.g., when other clients or neighboring APs act as hidden terminals).

Figures 13(a), 13(b) plot the aggregate throughput with ECR, ER, and 802_11, with 4 and 12 clients, respectively, when the loss rate bound varies from 10% to 90%. Again, ECR outperforms ER and 802_11 for all loss rate bounds and the improvement is higher with higher loss rate bounds and higher number of clients, i.e., with higher heterogeneity. With 4 clients (Figure 13(a)), the throughput gain of ECR over ER varies from 4% (with 10% loss rate) up to 10% (with 90% loss rate) and the gain over 802.11 varies from 0.5% up to 15%. With 12 clients (Figure 13(b)), the throughput gain of ECR over ER varies from 7% (with 10% loss rate) up to 51% (with 90% loss rate) and the gain over 802.11 varies from 2% up to 46%.

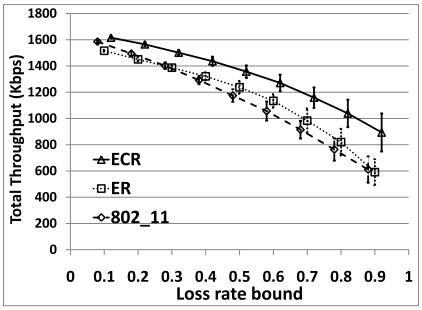
8 Testbed evaluation

In this section, we present experimental results comparing ECR and ER on an 802.11 testbed.

NC-based wireless protocols (e.g., [5, 11, 21]) are typically implemented as a shim between the IP and the MAC layer, i.e., at layer 2.5. Here, for ease of debugging, deployment, and evaluation, we implemented ECR at the application layer, using broadcast sockets. For a fair comparison, we also implemented ER at the application layer, following all the details in [21]. Our implementation handles only synthetic traffic, i.e. data packets are generated within the ER or ECR application running at the AP, similar to the implementation in [26], in which



(a) 4 clients



(b) 12 clients.

Figure 13. Throughput comparison with varying loss rate bounds under heterogeneous losses. For clarity, the points of the ECR and 802.11 curves are offseted horizontally.

packets are generated within Click. The application layer implementation of ECR and ER prevented us from a fair comparison of ER and ECR against 802.11, which is implemented in the driver/firmware of the wireless card.[9]

8.1 Experimental results

We set up a small testbed consisting of 9 low-end PCs, running Mandrake Linux 10.1 (kernel 2.6.11-6). Each is equipped with an Atheros 5212 based 802.11a/b/g wireless card operating in 802.11b ad hoc mode, using the open-source *madwifi* driver [17]. Each card is attached to a 2dBi rubber duck omnidirectional antenna with a low loss pigtail. The transmission power is set to 18dBm and RTC/CTS is disabled, as is the default setting.

In our evaluation, we used one machine as an AP and up to 8 machines as clients. To evaluate the performance of the two protocols under various loss scenarios, we generated loss rates between the AP and the clients in a controlled manner, similar to [21] and to our simulation methodology. For each scenario (i.e., a given number of clients and a given loss rate), the AP sends a 1.1MB file to each client, using 1460-byte packets. We repeat each scenario 10 times and we report the average throughput over the 10 runs.

We first wanted to verify that the complexity of the PCSI computation algorithm (which is executed every time the AP receives an ACK) (Section 5.3) remains reasonably low after removing all duplicated computations and does not become the bottleneck in the protocol's operation. Figure 14 plots the Cumulative Distribution Function (CDF) of the ACK process time in the case of $M = 4$ clients, with varying loss rates. The median ACK process time is equal to 203 μs, 139 μs, and 84 μs, for loss rates equal to 20%, 50%, and 80%, respectively. With a 1500-byte packet, these numbers limit the effective throughput to about 59Mbps, 86Mbps, and 142Mbps, respectively, which is higher than the maximum effective bit rate of 802.11 a/g WLANs (54 Mbps).

One observation made from Figure 14 is that the ACK processing time drops as the loss rate increases. The reason behind this behavior is the following: under high loss rates, each ACK acknowledges only a small number of packets, and hence the PCSI computation is repeated for a small number of times and the total ACK process time is low. On the other hand, under a low loss rates, the first ACKs for each batch acknowledge a large number of packets, which results in a large ACK process time (note that the 90-th percentile is considerably higher with a 20% loss rate compared to 50% or 80% loss rate). However, after the first ACKs, most of the packets at the AP are already ACKed and subsequent ACKs do not cause large changes to the overhearing bit map, resulting in a very

[9]The authors in [21] compared the performance of their ER implementation against a version of ER with coding disabled, which they used as an approximation of 802.11. We followed their methodology as a proof of concept, and confirmed that the gains of our ER implementation over ER with coding disabled are similar to the ones reported in [21].

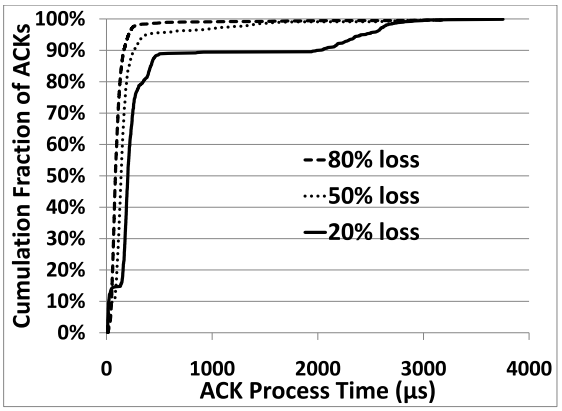


Figure 14. CDF of the ECR ACK processing time at the AP with varying loss rates.

Table 1. Testbed evaluation: Total throughput (in Kbps) with ER/ECR with varying numbers of clients and varying loss rates.

ER/ECR	Loss rate		
# of clients	20%	50%	80%
3	652.4/648.8	454.6/484.4	176.8/205
4	696.6/698	486.6/513.6	168/208.6
8	698.4/718.8	492.2/531.6	172.2/207.8

low process time (the 10-th percentile with a 20% loss rate is much lower than with 50% or 80% loss rate). As a result, the median process time does not grow too large even under low loss rates.

We now move on to compare the performance of the two protocols. Table 1 presents the average throughput with ER and ECR with 3, 4, and 8 clients, and for three different loss rates: 20%, 50%, and 80%. We observe that the two protocols perform similarly with a 20% loss rate (less than 3% difference) and ECR outperforms ER with 50% and 80% loss rates. The throughput gain of ECR is generally higher with the number of clients and the loss rate, ranging from 6.5-24%. As a specific example, the gain of ECR over ER is 0.2%, 5.5%, and 23.7%, with a 20%, 50%, and 80% loss rate, respectively in Table 1; the simulation gains for the same scenario from Figure 12(a) are 2.5%, 6.9%, and 24.9%, respectively, i.e., very close to the testbed gains.

9 Related work

In this section, we summarize the related work.

Theoretic Studies. The AP network corresponds to the classic "broadcast channel" problem in the information theory society, which has been an active research subject in the past four decades. Some example related works in this extremely rich literature include the 2-user feedback capacity exploration for the erasure channel [8, 23], the Gaussian channel [18], and the discrete memoryless channels [7]. The results in this work can also be viewed as a generalization of the *index coding* problem [3] from the noiseless channels to the random wireless broadcast erasure channels.

Other network coding based retransmission schemes. Recently, XORR [6], was proposed to address several issues that affect the performance of ER, though complementary to the coding strategy. In particular, it consid-

ered the impact of different link data rates on the coding decision and also incorporated a network coding aware opportunistic scheduler to the AP, to provide fairness among clients of different link qualities. However, XORR's coding strategy is far from optimal. Similar to MU-ARQ/ER, XORR also drops overheard coded packets that are not immediately decodable. Also, the protocol only considers the head-of-line packet of each flow when making coding decisions, which also limits the coding gain. In contrast, in this paper, we focused on the fundamental problem of designing an optimal coding strategy which is also amenable to a practical implementation, and we left the problem of adapting ECR to consider heterogeneous link data rates as part of our future work. Note that although fairness was not a direct goal of our design, ECR still provides fairness among each group of M clients.

More recently, [22] proposed *Medusa*, a proxy based solution to improve media streaming performance over WLANs. The design of *Medusa* includes an ER-like retransmission scheme, among other mechanisms. In contrast to ER, the threshold based scheduling algorithm of ER is no longer used; the protocol only uses ER's "sort-by-time" simple heuristic to determine which packets to code together.

Other retransmission schemes. A different approach to improving performance of WLANs is presented in [16], [2]. In these works, a client close to the AP acts as a relay for a client far from the AP, either relaying all the packets [2] or assisting only with retransmissions [16]. This approach requires collaboration among clients and are orthogonal to the approach ECRor ER's approach, where clients simply store each other's overheard packets but only talk to the AP.

Network coding in multihop wireless networks. Network coding has been extensively used to improve performance of multihop wireless networks over the past few years. The pioneering work in [1] showed that allowing relay nodes to encode and decode traffic rather than to simply forward, can achieve the multicast capacity. COPE was the first practical inter-flow network coding scheme for unicast in multihop wireless networks. It applies network coding to initial data transmissions and relies on 802.11 retransmissions for recovering from losses. COPE also sacrifices optimality for simplicity, using a simple "sort-by-time" coding strategy and dropping packets that are not immediately decodable, similar to ER.[10] The work in [20] identified two problems with COPE (poor performance in lossy environments and reduced coding gains from not recouping coding opportunities for packets not immediately decodable) and proposed CLONE, a suite of heuristic coding algorithms to address these problems. [20] also showed that the problem of determining the optimal coding strategy is NP-hard, even when only binary coding is allowed (i.e., only packets from two flows can be coded together). The authors concluded that out of all the proposed heuristics, only one (for binary coding) can be implemented on today's hardware. In contrast, ECR can not only achieve optimal theoretical performance, but also enables a practical implementation.

Intra-flow network coding has also drawn significant attention (e.g., [5, 15, 13]). MORE [5] was the first practical intra-flow network coding protocol, showing that the use of random linear network coding can greatly simplify the design of opportunistic routing protocols, since nodes no longer need to know exactly what packets have been forwarded by each neighbor. The design of MORE motivated the use of intra-flow network coding in ECR for solving a different problem: by using intra-flow network coding, the AP no longer needs to determine the order of coded packet transmissions within each phase. [9] theoretically computes the expected number of transmissions using ARQ, FEC. and network coding in the case of tree-based reliable multicast, and shows that network coding is the most efficient among the three schemes. In contrast, our work focuses on single-hop unicast.

Finally, there have also been some attempts to combine the two types of coding [19], [25]. I^{2}MIX combines the two types of coding by simply performing random linear network coding on all $\sum_{i=1}^{M} N_i$ packets (assuming M flows and a batch of N_i packets for flow i). As we have explained in Section 4.1, this approach is not efficient as all next hops need to receive all $\sum_{i=1}^{M} N_i$ packets before decoding is possible, which takes an excessive amount of time. In $C\&M$ [25], a node first creates a batch of inter-flow coded packets by mixing packets belonging to different flows and then sends out linear combinations of the created coded packets until all the next hops are able to decode their intended native packets. The paper discusses no details on the inter-flow coding strategy used.

[10] Actually, the design of ER was inspired by COPE, as mentioned in [21].

10 Conclusions

In this paper, we presented ECR, a novel network coding based retransmission protocol for WLANs. The design of ECR is accompanied by a theoretical underpinning yet enables practical implementation on off-the-shelf 802.11 hardware. We showed that, when the batch size N is sufficiently large, ECR attains the *provably optimal* inter-flow coding gain that is *strictly better* than that of the MU-ARQ principle, especially when the number of clients is from moderate to large. In addition, by inherently exploiting the simplicity advantage of intra-flow coding, the operation of ECR is straightforward and does not involve solving an NP-hard problem or resorting to suboptimal heuristics. Our performance evaluation, through extensive simulations and a testbed implementation, shows that the performance of ECR is very robust and consistently outperforms ER in a variety of scenarios.

In our future work we plan to address a few practical limitations of ECR that can further boost its performance, e.g., devise intelligent client grouping algorithms to group clients with similar loss rates together in order to avoid the "crying baby" problem, incorporate bitrate adaptation into ECR, and devise online batch size selection algorithms to allow the smooth operation of the protocol under higher layer protocols that set delay requirements (e.g., streaming protocols or TCP).

Acknowledgment

This work was supported in part by NSF grants CCF-0845968 and CNS-0905331.

References

[1] Rudolf Ahlswede, Ning Cai, Shuo-Yen Robert Li, and Reymond W. Yeung. Network information flow. *IEEE Transactions on Information Theory*, 46(4), July 2000.

[2] Paramvir Bahl, Ranveer Chandra, Patrick P. C. Lee, Vishal Misra, Jitendra Padhye, Dan Rubenstein, and Yan Yu. Opportunistic use of client repeaters to improve performance of wlans. In *Proc. of ACM CoNEXT*, 2008.

[3] Z. Bar-Yossev, Y. Birk, T.S. Jayram, and T. Kol. Index coding with side information. In *Proc. of IEEE FOCS*, 2006.

[4] Sanjit Biswas and Robert Morris. ExOR: Opportunistic multi-hop routing for wireless networks. In *Proc of ACM SIGCOMM*, 2005.

[5] Szymon Chachulski, Michael Jennings, Sachin Katti, and Dina Katabi. Trading structure for randomness in wireless opportunistic routing. In *ACM SIGCOMM*, 2007.

[6] Fang chun Kuo, Kun Tan, Xiangyang Li, Jiansong Zhang, and Xiaoming Fu. XOR Rescue: Exploiting Network Coding in Lossy Wireless Networks. In *Proc. of IEEE SECON*, 2009.

[7] A. El Gamal. The feedback capacity of degraded broadcast channels. *Trans. IT*, 25(2), 1978.

[8] L. Georgiadis and L. Tassiulas. Broadcast erasure channel with feedback — capacity and algorithms. In *Proc. of NetCod*, 2009.

[9] Majid Ghaderi, Don Towsley, and Jim Kurose. Reliability Gain of Network Coding in Lossy Wireless Networks . In *Proc. of IEEE INFOCOM*, 2008.

[10] T. Ho, M. Médard, R. Koetter, D.R. Karger, M. Effros, J. Shi, and B. Leong. A random linear network coding approach to multicast. *IEEE Trans. Inform. Theory*, 52(10):4413–4430, October 2006.

[11] Sachin Katti, Shyamnath Gollakota, and Dina Katabi. Embracing wireless interference: Analog network coding. In *Proc. of ACM SIGCOMM*, 2007.

[12] Sachin Katti, Hariharan Rahul, Wenjun Hu, Dina Katabi, Muriel Medard, and Jon Crowcroft. Xors in the air: Practical wireless network coding. In *Proc. of ACM SIGCOMM*, August 2006.

[13] Dimitrios Koutsonikolas, Chih-Chun Wang, and Y. Charlie Hu. CCACK: Efficient Network Coding Based Opportunistic Routing Through Cumulative Coded Acknowledgments. In *Proc. of IEEE INFOCOM*, 2010.

[14] Peter Larsoon and Niklas Johansson. Multi-User ARQ. In *Proc. of IEEE VTC-Spring*, 2006.

[15] Yunfeng Lin, Baochun Li, and Ben Liang. CodeOR: Opportunistic routing in wireless mesh networks with segmented network coding. In *Proc. of IEEE ICNP*, 2008.

[16] Mei-Hsuan Lu, Peter Steenkiste, and Tsuhan Chen. Design, implementation and evaluation of an efficient opportunistic retransmission protocol. In *Proc. of ACM Mobicom*, 2009.

[17] madwifi. `http://madwifi.org`.

[18] L.H. Ozarow and S.K. Leung-Yan-Cheong. An achievable region and outer bound for the Gaussian broadcast channel with feedback. *Trans. IT*, 30(4), 1984.

[19] Chuan Qin, Yi Xian, Chase Gray, Naveen Santhapuri, and Srihari Nelakuditi. I^2MIX: Integration of Intra-flow and Inter-flow Wireless Network Coding. In *Proc. of IEEE International Workshop on Wireless Network Coding (WiNC)*, 2008.

[20] Shravan Rayanchu, Sayandeep Sen, Jianming Wu, Suman Banerjee, and Sudipta Sengupta. Loss-Aware Network Coding for Unicast Wireless Sessions: Design, Implementation, and Performance Evaluation. In *Proc. of ACM SIGMETRICS*, 2008.

[21] Eric Rozner, Anand Padmanabha Iyer, Yogita Mehta, Lili Qiu, and Mansoor Jafry. Er: Efficient retransmission scheme for wireless lans. In *Proc. of CoNEXT*, 2007.

[22] Sayandeep Sen, Neel Kamal Madabhushi, and Suman Banerjee. Scalable wifi media delivery through adaptive broadcasts. In *Proc. of USENIX NSDI*, 2010.

[23] F. Xue and X. Yang. Network coding and packet-erasure broadcast channel. In *Proc. of IEEE SECON*, 2008.

[24] Xiang Zeng, Rajive Bagrodia, and Mario Gerla. Glomosim: A library for parallel simulation of large-scale wireless networks. In *Proc. of PADS Workshop*, May 1998.

[25] Xiaoyan Zhu, Hao Yue, Yuguang Fang, and Yumin Wang. A batched network coding scheme for wireless networks. *ACM Wireless Networks*, 15, 2009.

[26] More source code. http://people.csail.mit.edu/szym/more.