Efficient Network-Coding-Based Opportunistic Routing Through Cumulative Coded Acknowledgments

Dimitrios Koutsonikolas, *Member, IEEE, ACM*, Chih-Chun Wang, *Member, IEEE*, and Y. Charlie Hu, *Senior Member, IEEE, ACM*

***Abstract*—The use of random linear network coding (NC) has significantly simplified the design of opportunistic routing (OR) protocols by removing the need of coordination among forwarding nodes for avoiding duplicate transmissions. However, NC-based OR protocols face a new challenge: *How many coded packets should each forwarder transmit?* To avoid the overhead of feedback exchange, most practical existing NC-based OR protocols compute offline the expected number of transmissions for each forwarder using heuristics based on periodic measurements of the average link loss rates and the ETX metric. Although attractive due to their minimal coordination overhead, these approaches may suffer significant performance degradation in dynamic wireless environments with continuously changing levels of channel gains, interference, and background traffic. In this paper, we propose CCACK, a new efficient NC-based OR protocol. CCACK exploits a novel *C*umulative *C*oded *ACK*nowledgment scheme that allows nodes to acknowledge network-coded traffic to their upstream nodes in a simple way, oblivious to loss rates, and with negligible overhead. Through extensive simulations and testbed experiments, we show that CCACK greatly improves both throughput and fairness compared to MORE, a state-of-the-art NC-based OR protocol.**

***Index Terms*—Coded feedback, network coding, opportunistic routing, wireless mesh networks (WMNs).**

I. Introduction

WIRELESS mesh networks (WMNs) are increasingly being deployed for providing cheap, low-maintenance Internet access (e.g., [1]–[3]). A main challenge in WMNs is to deal with the poor link quality due to urban structures and interference, both internal (among flows in the WMN) and external (from other 802.11 networks). For example, 50% of the operational links in Roofnet [1] have loss rates higher than 30% [4]. Hence, routing protocol design is critical to the performance and reliability of WMNs.

Traditional routing protocols (e.g., [5]–[7]) for multihop wireless networks treat the wireless links as point-to-point links. First, a fixed path is selected from the source to the destination. Then, each hop along the chosen path simply sends data packets to the next hop via 802.11 unicast. *Opportunistic routing* (OR), as first demonstrated in the ExOR protocol [8], has recently emerged as a mechanism for improving unicast throughput in WMNs with lossy links. Instead of first determining the next hop and then sending the packet to it, a node with OR *broadcasts* the packet so that all neighbor nodes have the chance to hear it and assist in forwarding.

In practice, it is not beneficial if all nodes in the network participate in forwarding traffic for a single flow. Hence, existing OR protocols typically construct a *belt* of forwarding nodes (FNs) for each flow, and only members of the belt are allowed to forward packets.

OR provides significant throughput gains compared to traditional routing. However, it introduces a difficult challenge. Without any coordination, all members of the FN belt that hear a packet will attempt to forward it, creating spurious retransmissions, which waste bandwidth. To address this challenge, a coordination protocol needs to run among the nodes so that they can determine which one should forward each packet.

Recently, [9] showed that the use of *random intraflow network coding* (NC) can address this challenge in a very simple and efficient manner, with minimal coordination. With NC, the source sends random linear combinations of packets, and each router also randomly mixes packets it already has received before forwarding them. Random mixing at each router ensures that, with high probability, different nodes that may have heard the same packet can still transmit linearly independent coded packets.

NC has significantly simplified the design of OR protocols and led to substantial throughput gains [9] compared to non-coding-based protocols. However, the use of NC introduces a new challenge: *How many coded packets should each forwarder transmit?* This challenge, if not efficiently addressed, may prevent NC-based OR protocols from realizing the maximum possible gains.

A. Challenge in NC-Based OR Protocols

We illustrate the main challenge in NC-based OR protocols with the example shown in Fig. 1. This figure shows a typical scenario of an NC-based OR protocol. The source S has three downstream FNs A, B, and C. Assume for simplicity that S has three innovative packets X_1, X_2, and X_3 to send. Instead of transmitting the native packets, S transmits three coded packets $X_1 + X_2 + X_3, 3X_1 + X_2 + 2X_3$, and $X_1 + 2X_2 + 3X_3$ in sequence, which are denoted by the corresponding *coding vectors* (1, 1, 1), (3, 1, 2), and (1, 2, 3). Assume that the (1, 1, 1) coded

Manuscript received December 17, 2009; revised October 14, 2010; accepted January 16, 2011; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor S. Diggavi. Date of publication February 24, 2011; date of current version October 14, 2011. An earlier version of this paper was presented at the IEEE Conference on Computer Communications (INFOCOM), San Diego, CA, March 15–19, 2010. This work was supported in part by the National Science Foundation (NSF) under Grants CCF-0845968 and CNS 0905331.

The authors are with the School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN 47907 USA (e-mail: dkoutson@purdue.edu; chihw@purdue.edu; ychu@purdue.edu).

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

Digital Object Identifier 10.1109/TNET.2011.2111382

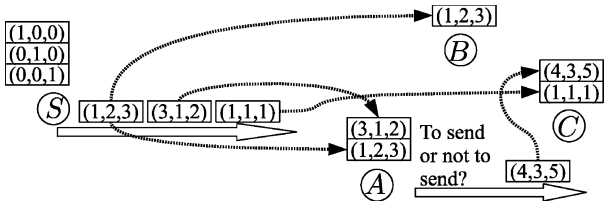


Fig. 1. Importance of knowing how many coded packets to transmit.

packet is received by C, and the (3, 1, 2) and (1, 2, 3) packets are received by A and by $\{A, B\}$, respectively. The downstream FNs A, B, and C have received a sufficient amount of innovative packets. Collectively, the three FNs can now act as the new source, and the original source S should stop transmission. However, it is a nontrivial task for S to know whether its downstream FNs have accumulated a sufficient amount of innovative packets.

The same challenge exists for the intermediate FN A. After transmitting a useful coded packet (4, 3, 5), which is received by FN C, A has to decide whether it should continue or stop sending coded packets. Furthermore, A has limited knowledge about the reception status of the three packets transmitted by S [e.g., A may not know that C has received (1, 1, 1) from S], which makes the decision of whether to stop transmission even harder for A than for the source S.

Note that overhearing, a commonly used way of acknowledging noncoded wireless traffic due to its zero overhead, does not suit network-coded traffic. For the same example in Fig. 1, when C has the opportunity to transmit, a network-coded packet (5, 4, 6) may be generated from the two innovative packets received by C. Even if A overhears this new (5, 4, 6) packet, A still does not know whether C received the (4, 3, 5) packet transmitted by A since it is not aware of the reception of the (1, 1, 1) packet by C.

One way to address the challenge is to combine individual packet overhearing, as in non-coding-based protocols, with a *credit system*, based on coded transmissions, and have the FNs perform detailed bookkeeping to guarantee credit conservation in the system. This approach is taken in MC^2 [10]. Although theoretically optimal [11], this approach is quite complex in practice. In addition, like every approach that relies on individual packet overhearing, it requires a reliable control plane. In typical WMN environments with high packet loss rates or contention [4], this approach can cause excessive signaling overhead and retransmissions, which can significantly limit the performance.

B. Loss-Rate-Based Approaches

Since theoretically optimal solutions are hard to implement in practice, existing NC-based OR protocols use heuristics based on link loss rates to address the challenge in a simple manner and to minimize the control overhead.

MORE [9], the first NC-based OR protocol, employs an offline approach that requires no coordination among FNs. In MORE, the source calculates and assigns a *transmission credit* to each FN using the ETX metric [12] computed from loss rate measurements. Receptions from upstream nodes are then used to trigger new transmissions at the FNs, with precomputed relative frequencies using the transmission credits. Since the ETX metric expresses the *expected* behavior, the approach used in MORE cannot guarantee that the destination will always receive enough packets due to the randomness of the wireless channel. Hence, the source in MORE keeps transmitting packets from the same batch until it receives an ACK from the destination, unnecessarily increasing interference.

Many other works that improve MORE also use offline measured loss rates as a basic component in their proposed solutions (e.g., [13]–[15]).

The drawback of all these approaches is that performance heavily depends on the accuracy and freshness of the loss-rate measurements. Loss-rate estimates are obtained through periodic probing and are propagated from all nodes to the source. Apparently, the higher the probing frequency, the higher the accuracy, but also the higher the overhead. As a recent study [16] showed, even low-rate control overhead in nonforwarding links can have a multiplicative throughput degradation on data-carrying links.

To reduce this overhead, the authors of MORE collect the loss rates and calculate the credits only in the beginning of each experiment. In practice, this suggests that loss-rate measurements should be performed rather infrequently. Unfortunately, recent WMN studies [17], [18] have shown that *although link metrics remain relatively stable for long intervals in a quiet network, they are very sensitive to background traffic*. For example, in [17], the authors observe that 100 ping packets (one per second) between two nodes in a 14-node testbed caused an increase of 200% or more to the ETT [19] metric of around 10% of the links.[1] Even worse, a 1-min TCP transfer between two nodes in the same network caused an increase of more than 300% to the ETT metric of 55% of the links.

In summary, these approaches suffer from difficulties in accurately estimating loss rates. Overestimated loss rates cause redundant transmissions, which waste wireless bandwidth. On the other hand, underestimated loss rates may have an even worse impact since nodes may not transmit enough packets to allow the destination to decode a batch. This motivates the need for a new approach *oblivious* to loss rates.

C. Our Approach—Cumulative Coded Acknowledgments

In this paper, we present a novel approach to NC-based OR and propose CCACK, a new efficient NC-based OR protocol. Unlike existing protocols, FNs in CCACK decide how many packets to transmit in an online fashion, and this decision is completely oblivious to link loss rates.[2] This is achieved through a novel *C*umulative *C*oded *ACK*nowledgment scheme that allows nodes to acknowledge network-coded traffic to their upstream nodes in a simple and efficient way with negligible overhead. Feedback in CCACK is not required strictly on a per-packet basis; this makes the protocol resilient to individual packet loss and significantly reduces its complexity.

[1]The ETT metric estimates the quality of a link taking into account both the loss rate (through the ETX metric) and the link bandwidth.

[2]By "oblivious to link loss rates," we mean here that loss rates are not taken into account in determining how many packets each FN should transmit. We still use MORE's loss-rate-based offline algorithm in CCACK to build the FN belt for a fair comparison between the two protocols. We note though that the coded feedback mechanism in CCACK is orthogonal to the belt construction.

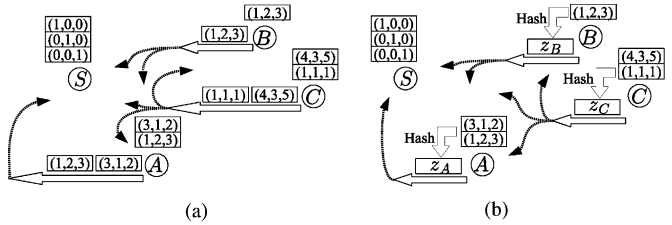


Fig. 2. Different types of feedback for network-coded traffic. (a) Uncoded feedback. (b) Coded feedback.

Take the scenario in Fig. 1 as a continuing example. One naive approach to ensure that S (resp. A) knows when to stop transmission is through the use of *reception reports*, for which each node broadcasts *all the basis vectors* of the received linear space to its upstream nodes, as illustrated in Fig. 2(a).[3]

An obvious drawback of this approach is the size of the feedback messages. For practical network coding with symbol size $\mathrm{GF}(2^8)$ and batch size 32, each coding vector contains 32 B. To convey a space of dimension $\kappa \gg 1$ thus requires κ 32-B vectors, which is too large to piggyback to normal forward traffic. The unreliability of the wireless channel further exacerbates the problem as the $\kappa\times$32-B feedback messages need to be retransmitted several times until they are overheard by all the upstream nodes.

In contrast, in CCACK each node uses *a single coded feedback vector* to represent the entire space, which may consist of $\kappa \gg 1$ basis vectors. In the broadest sense, the three coded acknowledgment vectors z_A to z_C in Fig. 2(b) serve as a hash for their corresponding spaces. As will be explained in Section III, we have devised a simple mechanism that successfully *compresses* (most of) the space information into a single vector, say z_A for node A, while allowing upstream nodes to *extract* the original space from z_A without exchanging any additional control information. Each single vector z_A can be easily piggybacked to the forward data traffic. This compressed/coded acknowledgment is critical to the efficiency since, in CCACK, overhearing any of the data packets of A with piggybacked coded ACK will convey to the upstream nodes the entire space (or most of the space) of A.

In addition to efficiently solving the challenge of how many packets each FN should transmit, the cumulative coded acknowledgment scheme in CCACK enables us to develop an efficient rate control algorithm. In contrast, MORE has no explicit rate control mechanism, and its performance degrades as the number of flows in the network increases [9], [11], [13], [14].

D. Contributions

This paper makes the following contributions.

- We identify a main challenge in the newly emerged class of NC-based OR protocols: *How many coded packets should each forwarder transmit?* We discuss the inefficiencies of existing loss-based approaches in addressing this challenge and show, through our simulation and testbed evaluations, the severe impact such approaches can have on the performance of NC-based OR protocols.
- We propose CCACK, a new efficient NC-based OR protocol. Unlike existing protocols, FNs in CCACK decide how many packets to transmit in an online fashion, and this decision is completely oblivious to link loss rates. Central to the design of CCACK is a novel Cumulative Coded ACKnowledgment scheme that allows nodes to acknowledge network-coded traffic to their upstream nodes in a simple and efficient way, with negligible overhead. In addition to efficiently solving the challenge of how many packets each FN should transmit, the cumulative coded acknowledgment scheme in CCACK enables us to develop an efficient rate control algorithm.
- CCACK brings a shift to the design paradigm of NC-based OR protocols. Existing NC-based OR protocols have identified feedback overhead as a main cause for performance degradation in practical wireless routing protocols and used NC to eliminate the need for feedback exchange, resorting to offline loss-based heuristics. On the contrary, CCACK *encodes* feedback to exploit its benefits and avoid the drawbacks of offline heuristics and, at the same time, to hide its overhead.
- We present extensive simulations with a realistic physical model showing that CCACK offers significant throughput and fairness improvements over the state-of-the-art MORE by 27%–45% and 5.8%–8.8%, respectively, on average, with a varying number of flows. For some challenged flows that completely starve under MORE, CCACK increases throughput by up to 21$\times$ and fairness by up to 124%. We further quantify the header, memory, and coding overheads of CCACK and show that they are comparable to those of MORE, making CCACK easily deployable on commodity hardware.
- We present an application-layer implementation of CCACK and MORE and their evaluation on a 22-node 802.11 WMN testbed deployed in two academic buildings at Purdue University, West Lafayette, IN. Despite the small size of our testbed along with the limitations of our implementation that limit the potential gains, our testbed results show that CCACK improves both throughput and fairness over MORE by up to 3.2$\times$ and 83%, respectively, with average improvements of 11%–36% and 5.7%–8.3%, respectively, validating the benefits of our approach.

The remainder of this paper is organized as follows. In Section II, we introduce the basic principles of coded feedback through a simple existing coded feedback scheme. We identify two problems with this scheme that motivate the design of CCACK, presented in Section III. Section IV evaluates the performance of CCACK and MORE through extensive simulations, and Section V describes the implementation and evaluation of CCACK and MORE on a wireless testbed. Finally, Section VI concludes the paper.

II. Existing Coded Feedback Scheme

One candidate solution (which has been used in the past in a different context [20]), attractive due to its simplicity, is null-space-based (NSB) coded feedback, i.e., each node sends to each upstream node one vector randomly chosen among all vectors in the null space of the innovative vectors the node has received in the past. Take for example Fig. 3(a), which shows FNs A and B from Fig. 1. Let B_{in} denote the buffer containing

[3]We sometimes refer to the linear space spanned by the received vectors as the *knowledge space*.

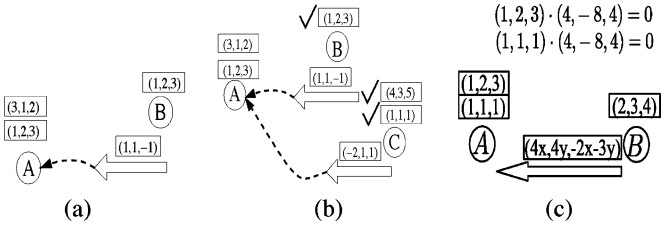


Fig. 3. Null-space-based (NSB) coded feedback. (a) Basic operation. (b) Collective space problem. (c) False-positive error.

the innovative coding vectors received by an FN [B_{in} contains two vectors (1, 2, 3) and (3, 1, 2) at node A and one vector (1, 2, 3) at node B in Fig. 3(a)]. Every time B broadcasts a coded data packet to its own downstream nodes, it also appends to the packet header an ACK vector z_B satisfying

$$z_B \cdot v = 0 \qquad \forall v \in B_{\text{in}}. \tag{1}$$

Namely, the inner product between z_B and v is zero. There may be multiple choices of z_B that satisfy this condition [e.g., in Fig. 3(a), z_B can be any vector of the form $(-2x-3y, x, y)$]. z_B is then chosen uniformly randomly among all valid vectors satisfying (1). Let $S_B = \langle v : v \in B_{\text{in}} \rangle$ denote the linear space spanned by vectors in B_{in}. One can easily show the following.

Lemma 1: With the above random construction of z_B, any vector $v' \in S_B$ must satisfy $z_B \cdot v' = 0$. Moreover, for any vector $v'' \notin S_B$, we have $\text{prob}(z_B \cdot v'' = 0) = \frac{1}{2^8}$ assuming the $\text{GF}(2^8)$ finite field is used.

The intuition behind Lemma 1 can be explained as follows. Suppose the rank of space S_B in Lemma 1 satisfies $\text{Rank}(S_B) < 32$. Moreover, we further restrict ourselves to choose a z vector that is strictly nonzero. Once such a z is chosen, there are totally 256^{31} different vectors u that satisfy $z \cdot u = 0$. Among them, $256^{\text{Rank}(S_B)}$ of them are in S_B and $256^{31} - 256^{\text{Rank}(S_B)}$ of them are not in S_B. Note that, out of the entire space (totally 256^{32} different vectors), there are $256^{32} - 256^{\text{Rank}(S_B)}$ v'' vectors that are not in S_B. Therefore, the probability that a randomly chosen $v'' \notin S_B$ satisfies $z \cdot v'' = 0$ is

$$\frac{256^{31} - 256^{\text{Rank}(S_B)}}{256^{32} - 256^{\text{Rank}(S_B)}} < \frac{1}{256}.$$

From the above lemma, when the upstream node A overhears a packet from B, it simply needs to compute the inner product of each of its own innovative vectors with z_B. Suppose that z_B is chosen as $(1, 1, -1)$. Since $(1, 2, 3) \cdot (1, 1, -1) = 0, A$ concludes that B has received packet $(1, 2, 3)$. On the other hand, since $(3, 1, 2) \cdot (1, 1, -1) = 2 \neq 0, A$ concludes that packet $(3, 1, 2)$ is an innovative packet for B, and hence it should send more coded packets to B.

A. Problems of the NSB Coded Feedback for Unicast OR

Although attractive due to its simplicity, the NSB feedback scheme suffers from two significant limitations when used in NC-based OR.

Problem 1: Collective Space Problem: Take Fig. 3(b) for example. Based on the NSB concept, B and C send $z_B = (1, 1, -1)$ and $z_C = (-2, 1, 1)$, respectively, which are orthogonal to their local innovative vectors. When A checks the inner product of the coded feedback and its own innovative packets, we have

$$z_B \cdot (3, 1, 2) = 2 \neq 0 \quad z_C \cdot (3, 1, 2) = -3 \neq 0.$$

Therefore *A thinks that the coding vector* (3, 1, 2) *is innovative to both its downstream nodes and thus continues transmission even when collectively B and C have had enough information already*. It will not stop transmission until one of its downstream nodes has a local knowledge space that covers the local knowledge space of A. This defeats the purpose of OR. This misjudgment is caused by the fact that the NSB coded feedback does not convey the collective space of all downstream nodes, but only the space relationship between the individual pairs (i.e., A versus B and A versus C).

Problem 2: Nonnegligible False-Positive Probability: Take Fig. 3(c) for example. Node A would like to send two packets to node B, and a coded packet has been received by B already. B sends an orthogonal vector z_B satisfying (1), which is randomly chosen to be any vector of the form $z_B = (4x, 4y, -2x - 3y)$. Suppose that B happens to choose $z_B = (4, -8, 4)$. *Since z_B is orthogonal to all the innovative vectors of A, A will wrongfully infer that the knowledge space of B covers the local knowledge space of A. A thus attempts no further transmission*. Although the probability of such a false-positive event is small, its impact to the system performance is significant. Any single hop that experiences this false-positive event will cause an upstream node to stop transmission prematurely. The communication chain is thus broken, and the destination may not be able to receive enough innovative packets to decode the current batch.

III. CCACK Design

In this section, we present the design of CCACK. We first describe how CCACK addresses the collective space problem. We then describe a novel cumulative coded feedback scheme, which reduces the false-positive probability to practically zero. Finally, we present a simple rate control algorithm, built upon this coded feedback scheme.

The source and the intermediate FNs in CCACK use intraflow random linear NC. We selected a batch size of $N = 32$ packets, and the random coefficients for each linear combination are selected from a Galois Field (GF) of size 2^8, the same as in [9], [10], and [15]. Similar to [20], nodes in CCACK embed an additional *ACK vector* in the header of each coded data packet of the forward traffic to acknowledge a subset of the packets (or coding vectors) they have received (heard) in the past from their upstream nodes. For the following, we use the terms *forward coding vectors* and *ACK coding vectors* to denote the coding coefficients used to encode the payload of the packets and the feedback vectors used to acknowledge the space, respectively. Basically, the forward coding vector and the payload is used by any downstream node that receives this packet. The ACK coding vector is used by any upstream node that receives this packet. For simplicity, assume for now that the ACK coding vector is constructed following the NSB principle, i.e., it is simply a vector in the null space of the innovative vectors owned by the node. In Section III-B, we will describe a new construction of

the ACK coding vector that solves the false-positive problem of the NSB feedback.

All packets in CCACK are broadcast so that they can be heard by all neighbor nodes. Each node maintains a buffer $B_{\rm in}$ of N entries where it stores all the innovative packets of the current batch. If a node receives a packet from an upstream node, it checks whether the packet is innovative by comparing the coding vector to those of the existing packets in $B_{\rm in}$. If innovative, the newly received packet is stored in $B_{\rm in}$, similarly to MORE and other existing NC-based OR protocols.[4]

Unlike the intermediate nodes, which piggyback the ACK vectors to data packets, the destination periodically broadcasts coded feedback to its upstream nodes in the form of ACK vectors (without any payload). This is necessary since the destination sends no data packets. Once it receives N innovative packets for a batch, it decodes the batch to obtain the N original packets. It then sends an end-to-end ACK back to the source along the shortest ETX path in a reliable manner.[5]

A. Solving the Collective Space Problem

As we saw in Section II-A, the collective space problem is due to the fact that each NSB coded acknowledgment can only covey the space information/relationship between one pair of nodes (from the downstream node back to the upstream node). Hence, the first step toward addressing the collective space problem is to *collectively* consider the NSB acknowledgments from all the downstream nodes.

However, this is generally not enough. In Fig. 3(b), we see that the vector (4, 3, 5) owned by C is a linear combination of the two vectors owned by A. When A receives $z_C = (-2, 1, 1)$ from C, it computes $z_C \cdot (3, 1, 2) \neq 0$ and decides that its vector (3, 1, 2) has not been heard by any downstream node. Thus, it continues transmitting packets. Note that the same problem would arise if (4, 3, 5) had been sent by an upstream node of A and had been received by both A and C. In that case, A would have dropped it since it is linearly dependent to its own two vectors (1, 2, 3) and (3, 1, 2), and the picture would be the exactly the same as in Fig. 3(b). Hence, to address the collective space problem, nodes need to *remember all the packets that have been in the air*, not only the innovative ones.

To achieve this, nodes in CCACK maintain two additional vector buffers per flow: $B_{\rm rx}$ and $B_{\rm tx}$. The size $B_{\rm rx}$ and $B_{\rm tx}$ can be larger than N since these buffers only store 32-B coding vectors and not whole packets. In our implementation, we used a size equal to $5 \times N$. The forward coding vectors in $B_{\rm rx}$ and $B_{\rm tx}$ together constitute all the vectors that have recently been in the air and have been heard by the node of interest. More specifically, nodes store the coding vectors of *all* packets they receive from upstream nodes in $B_{\rm rx}$ and the coding vectors of the packets they broadcast in $B_{\rm tx}$. Each such vector can be marked as H (heard by a downstream node) or ¬H (not heard). A vector is marked as ¬H when it initially is inserted in either of the two buffers since the node has no information at that time whether any downstream node has heard the packet or not. Nodes gradually mark the coding vectors in $B_{\rm rx}$ and $B_{\rm tx}$ as H as they receive more ACK coding vectors from downstream nodes.

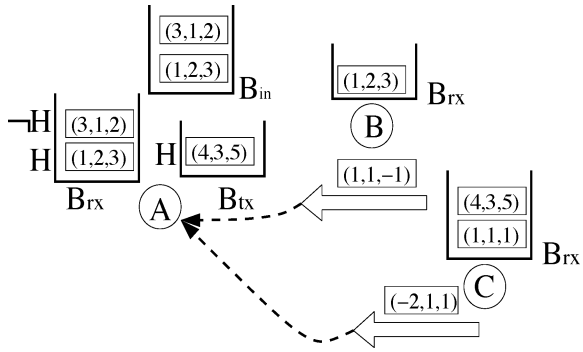


Fig. 4. Solving the collective space problem in CCACK.

In contrast to the NSB coded feedback scheme in [20], nodes in CCACK construct the ACK coding vectors using *all the received forward coding vectors stored in* $B_{\rm rx}$ and not only the innovative vectors stored in $B_{\rm in}$. Also, when a node A overhears a packet from a downstream node, it uses the ACK coding vector of that packet to decide whether any of the coding vectors in $B_{\rm rx} \cup B_{\rm tx}$, instead of $B_{\rm in}$, have been heard by the downstream node.

We first explain the benefits of using separate buffers $B_{\rm rx}$ and $B_{\rm tx}$ in terms of solving the collective space problem through our running example from Fig. 3(b) in Fig. 4. For node A, $B_{\rm rx}$ contains the received coding vectors (1, 2, 3) and (3, 1, 2), while $B_{\rm tx}$ contains the transmitted vector (4, 3, 5). By checking inner products with z_B and z_C, A knows that the $(1, 2, 3) \in B_{\rm rx}$ and $(4, 3, 5) \in B_{\rm tx}$ have been *heard* by downstream nodes and accordingly marks them as H. Since the rank of the H vectors in $B_{\rm rx} \cup B_{\rm tx}$ is the same as the rank of the innovative vectors in $B_{\rm in}$, A stops transmission.

To analyze more rigorously how the usage of two separate buffers $B_{\rm rx}$ and $B_{\rm tx}$ can greatly alleviate the collective space problem, we consider the simplified representative scenario in Fig. 1. A is the closest neighbor of S and can help relay packets to its downstream nodes B and C. To focus on the collective space problem faced by A, we assume only S and A have transmitted packets thus far. The downstream nodes B and C only listen to the transmissions of S and A, and then compute ACK packets accordingly via the NSB feedback. We further assume that the coding finite field $\mathrm{GF}(q)$ is sufficiently large so that the false-positive probability of the NSB feedback is zero and we can isolate the collective space problem.

Since source S has all the packets, its knowledge space S_S is simply the entire vector space of the current batch of N packets. We then divide the transmitted packets into four groups:

Group 1: those packets transmitted by S and heard by A, but not by any of B and C;

Group 2: those packets transmitted by S and heard by one of $\{B, C\}$, but not by A;

Group 3: those packets transmitted by S and heard by A and by one of $\{B, C\}$;

Group 4: those packets transmitted by A and heard by one of $\{B, C\}$.

[4]Note that with minor modifications, the performance of the protocol could be slightly improved if nodes also store any innovative packet they potentially receive from downstream nodes. We have incorporated this feature in our implementation, but omit it here to simplify the discussion.

[5]In our current implementation, similar to MORE, the source moves to batch $i + 1$ only when it receives the end-to-end ACK from the destination for batch i. As [15] showed, a better approach is for the source to move to batch $i + 1$ immediately after it stops transmitting packets for batch i. In the future, we plan to incorporate this feature in CCACK.

We first observe that by using two separate buffers $B_{\rm rx}$ and $B_{\rm tx}$, A can use the collective NSB feedback from the downstream nodes B and C to decide whether the packets in its $B_{\rm rx}$ or $B_{\rm tx}$ buffer belong to Group 3 or 4. As a result, *a collective space problem happens when and only when the linear span of Group-2, Group-3, and Group-4 packets (i.e., packets heard by B, C), denoted by* $\mathsf{span}(G2, G3, G4)$, *covers the linear span of Group-1 and Group-3 packets (packets heard by A)* $\mathsf{span}(G1, G3)$, *while* $\mathsf{span}(G3, G4)$ *(i.e., those packets heard by B, C and which A is aware of through NSB feedback) does not cover* $\mathsf{span}(G1, G3)$. Therefore, with the help of two separate buffers, the collective space problem is completely solved when all packets heard by B or C are also heard by A (i.e., when Group 2 is empty). In practice, we often encounter the case when we do have some Group-2 packets. We show the following. 1) In addition to the above case (in which Group 2 is empty), there are many other cases when the collective space problem is indeed solved by using separate buffers. 2) Even for the cases when the collective space is not completely solved by separate buffers, the penalty we pay is generally very small.

Consider the moment when $\mathsf{span}(G2, G3, G4) \supseteq \mathsf{span}(G_1, G_3)$ is satisfied for the first time, i.e., when A should stop transmission since the collective space of B and C covers A. We consider the following two scenarios.

Scenario 1: $\mathrm{Rank}(\mathsf{span}(G_1, G_3)) + \mathrm{Rank}(\mathsf{span}(G_2)) \leq N$. Due to the use of random linear network coding (RLNC) at source S, Group-2 packets are *as linearly independent as possible* from Group-1 and Group-3 packets. We can thus prove that among Groups 2–4, only Group-3 and Group-4 packets can serve the roles of covering Group-1 and Group-3 packets. Therefore, $\mathsf{span}(G2, G3, G4) \supseteq \mathsf{span}(G_1, G_3)$ implies $\mathsf{span}(G3, G4) \supseteq \mathsf{span}(G_1, G_3)$. A thus knows that it should stop transmission. The collective space problem is completely solved.

Scenario 2: $\mathrm{Rank}(\mathsf{span}(G_1, G_3)) + \mathrm{Rank}(\mathsf{span}(G_2)) = N + \delta_{\rm Rank}$ for some $\delta_{\rm Rank} > 0$. Following a similar RLNC-based argument as used in Scenario 1, one can show that the overlap between $\mathsf{span}(G_2)$ and $\mathsf{span}(G_1, G_3)$ is at most of rank $\delta_{\rm Rank}$. As a result, Group-2 packets can help cover $\mathsf{span}(G_1, G_3)$ for at most rank $\delta_{\rm Rank}$. Therefore

$$\mathrm{Rank}(\mathsf{span}(G3, G4)) \geq \mathrm{Rank}(\mathsf{span}(G_1, G_3)) - \delta_{\rm Rank}.$$

This means that even though A "feels that" the collective space of B and C does not cover itself (while it actually does) in the present moment, after another $\delta_{\rm Rank}$ successful transmission from A to $\{B, C\}$, A will be sure that the collective space of B and C covers itself. The penalty of the collective space problem is thus only $\delta_{\rm Rank}$ additional transmissions.

Note that since we use CCACK to stop the transmission of S as well, it is likely that the S stops transmission the very first moment that the collective space of A, B, and C covers that of S. Since RLNC is used, it means that S stops transmission whenever $\mathrm{Rank}(\mathsf{span}(G_1, G_3)) + \mathrm{Rank}(\mathsf{span}(G_2)) = N$ (recalling that Group-2 packets are independent from Groups 1 and 3). Therefore, we mostly encounter Scenario 1 (when $\delta_{\rm Rank} = 0$). In some rare cases when CCACK is not able to stop the transmission of S right away, we will see $\delta_{\rm Rank} > 0$. However, in general, the corresponding $\delta_{\rm Rank}$ is expected to be very small, and hence the penalty we pay in Scenario 2 is very small. The simulation results in Section IV-B confirm our intuition that Scenario 2 happens rarely (or happens, but with a small $\delta_{\rm Rank}$) by comparing the performance of CCACK to a perfect ACK scheme that is free of any collective-space problem.

B. Solving the False-Positive Problem

We now describe our new ACK design that drastically reduces the false-positive probability from $\frac{1}{2^8}$ to $(\frac{1}{2^8})^M$ for any integer $M \geq 1$. From Lemma 1, when a node C sends one vector in the null space of its innovative vectors, the false-positive probability at an upstream node A is $\leq \frac{1}{2^8}$. Obviously, if node A sends $M > 1$ vectors in the null space of its innovative vectors, all in the same packet, then the overall false-positive event happens when all M orthogonality tests return false positive, and the overall false-positive probability is $(\frac{1}{2^8})^M$. Hence, the main idea in CCACK is to append only *one* ACK vector to each data packet, similar to NSB, but to construct it in such a way that it is *almost equivalent* to appending M vectors independently distributed over the null space. We achieve this effect by using M hash matrices $H^{(1)}$ to $H^{(M)}$. Since the vector/matrix operations are heavily used in our description, we further assume that all vectors are row vectors, and we use the transpose $u^{\rm T}$ to represent a column vector (constructed from the row vector u).

More explicitly, assume that a node C wants to acknowledge L vectors, $u_1, \ldots, u_L$, from its $B_{\rm rx}$ to its upstream node A, and let S denote the linear space spanned by the L vectors. With NSB, C would send a vector $z = (c_1, \ldots, c_N)$ satisfying $z \cdot u_i = 0, i = 1, \ldots, L$. With the hash matrices $H^{(1)}$ to $H^{(M)}$, we can rotate the chosen z vector M times by multiplying it with each of the M hash matrices, respectively. We require that the vectors created from the M rotations, $z^{(j)} = zH^{(j)}, j = 1, \ldots, M$ remain orthogonal to $u_1, \ldots, u_L$, i.e., we have $L \times M$ linear equations $u_i H^{(j)} z^{\rm T} = 0, i = 1, \ldots, L, j = 1, \ldots, M$, and we choose a single vector z that satisfies these equations. Such z now represents M vectors $z^{(j)}, j = 1, \ldots, M$, and all of them are orthogonal to $u_i, i = 1, \ldots, L$. Sending one such z is thus as if we have used the NSB feedback scheme for M times.

To make the above intuition rigorous, we compute globally M different $N \times N$ *hash matrices* $H^{(1)}$ to $H^{(M)}$, where $N = 32$ is the batch size. Each $H^{(i)}$ matrix is a diagonal matrix with the diagonal elements denoted by $h_1^{(i)}$ to $h_N^{(i)}$. We require that the choice of the set of M hash matrices satisfies that: 1) $h_j^{(i)} \neq 0$ for all $i = 1, \ldots, M$ and $j = 1 \ldots, N$; and 2) for any M distinct indices $j_1, \ldots, j_M$ in $\{1, \ldots, N\}$, the following square matrix has full rank:

$$\mathrm{Rank}\left(\begin{bmatrix} h_{j_1}^{(1)} & h_{j_1}^{(2)} & \cdots & h_{j_1}^{(M)} \\ h_{j_2}^{(1)} & h_{j_2}^{(2)} & \cdots & h_{j_2}^{(M)} \\ \vdots & \vdots & \cdots & \vdots \\ h_{j_M}^{(1)} & h_{j_M}^{(2)} & \cdots & h_{j_M}^{(M)} \end{bmatrix}\right) = M. \quad (2)$$

The intuition behind is that like any other hash-based mechanisms, we would like the hash matrices to be *as random as possible*.[6] Equation (2) explicitly quantifies the "desired level of

[6]One obvious construction method is to choose all M hash matrices randomly, which turns out to work quite well in our simulation and testbed implementation.

randomness" in terms of the corresponding linear independence between the column vectors $(h_j^{(1)}, \ldots, h_j^{(M)})$. Such a quantitative level of randomness is critical for our theoretic exploration.

Since the hash matrices are designed globally and can be reused for all batches, we only need to compute them once, and they can be reused by any CCACK protocols. In our CCACK protocol design, we start from $M = 4$ randomly chosen hash matrices $H^{(1)}$ to $H^{(M)}$ and repeatedly perturb the hash matrices until they satisfy (2). Using the precomputed hash matrices, the detailed algorithm for constructing of an ACK vector is as follows:

§ CONSTRUCT THE ACK VECTOR

1: Start from a $0 \times N$ matrix Φ.
2: **repeat**
3: Choose the u with the smallest $\mathrm{usage\_count}$ from B_{rx}. If more than one such u exist, choose one randomly.
4: **if** u is linearly independent to the row space of Φ **then**
5: Add[7] u to Φ.
6: **end if**
7: Increment the $\mathrm{usage\_count}$ of u by 1.
8: **until** the number of rows of Φ equals $\frac{N}{M} - 1$ or until all u in B_{rx} have been selected.
Remark 1: These vectors in Φ are the vectors being acknowledged.
9: For each i, the matrix product $\Phi H^{(i)}$ is a $(\frac{N}{M} - 1) \times N$ matrix. Construct a $(N - M) \times N$ matrix Φ' by vertically concatenating all $\Phi H^{(i)}$ matrices for $i = 1, \ldots, M$.
10: Choose randomly the coding coefficients c_1 to c_N such that the following matrix equation is satisfied:

$$\Phi'(c_1, \ldots, c_N)^{\mathrm{T}} = (0, \ldots, 0)^{\mathrm{T}}. \tag{3}$$

Remark 2: By Lines 8 and 9, (3) contains $(N - M)$ linear equations and N variables. We thus have at least M degrees of freedom when solving the above equations. Therefore, we can convert Φ' to its equivalent row-echelon form, which makes it easy to choose randomly the desired c_1 to c_N.
Remark 3: When randomly assigning the values of c_1 to c_N, we further require that out of the N coefficients c_1 to c_N, at least M of them are nonzero. This can always be achieved easily by choosing the M free variables in the row-echelon form to be nonzero.
11: Use the vector $(c_1, \ldots, c_N)$ as the ACK vector.

To improve the efficiency of our feedback mechanism, we associate a $\mathrm{usage\_count}$ with every vector in B_{rx} (Line 3 of the algorithm). When a vector is first placed in B_{rx}, its $\mathrm{usage\_count}$ is set to 0. Every time this vector is selected in the feedback construction algorithm, its $\mathrm{usage\_count}$ is incremented by 1. The ACK vector is always constructed using those vectors in B_{rx} with the lowest counts. This will reduce the probability that the same vectors are repeatedly acknowledged many times.

[7]We can convert Φ to its row-echelon form in order to speed up the computation when adding a new vector u.

We then observe that we can only acknowledge $\frac{N}{M} - 1$ vectors in B_{rx} (Line 9). The reason is that if we acknowledge $\geq \frac{N}{M}$ coding vectors in B_{rx}, then the resulting Φ matrix has $\geq \frac{N}{M}$ rows and the corresponding Φ' matrix has $\geq N$ rows. As a result, the number of equations in (3) is larger than the number of variables in (3). Generally, it is thus impossible to find a nonzero vector $(c_1, \ldots, c_N)$ satisfying (3). Note that the global parameter M thus represents a tradeoff between the number of u vectors one can acknowledge $(\frac{N}{M} - 1)$ and the false-positive probability $(2^{-8})^M$. Since *any false-alarm event for any packet over any link will trigger the landsliding cost of breaking the communication chain*, we observe in our experiments that any $M \leq 3$ will severely jeopardize the reliability of the CCACK. In our implementation we thus choose $M = 4$, which gives a false-positive probability of 2.32×10^{-10} that is necessary for the effectiveness of CCACK.

When a node A overhears a packet with an ACK vector z from a downstream node, it examines the coding vectors in its $B_{\mathrm{rx}} \cup B_{\mathrm{tx}}$ by checking the orthogonality to the M rotated versions of z. More explicitly, a vector $w \in B_{\mathrm{rx}}$ (or B_{tx}) is marked H whenever w passes *all the following M different "H_tests"* (one for each $H^{(j)}$):

$$\forall j = 1, \ldots, M \qquad wH^{(j)}z^{\mathrm{T}} = 0. \tag{4}$$

Hence, although the downstream node sent only one ACK vector z, the effect is the same as if it sent M orthogonal vectors $H^{(1)} \cdot z^{\mathrm{T}}$ to $H^{(M)} \cdot z^{\mathrm{T}}$. We now formally quantify the false-positive probability (passing all M tests simultaneously) with this new coded feedback scheme.

Proposition 1: Suppose a node C would like to acknowledge $(\frac{N}{M} - 1)$ independent vectors u_1 to $u_{\frac{N}{M}-1}$ by constructing an ACK vector z according to the proposed feedback construction algorithm. Let S denote the linear subspace spanned by u_1 to $u_{\frac{N}{M}-1}$. If we choose a vector w uniformly randomly from $\mathrm{GF}(q)^N$ and apply the M orthogonality tests in (4), we then have

$$\mathsf{prob}(w \text{ passes all } M \text{ tests} | w \in S) = 1$$
$$\mathsf{prob}(w \text{ passes all } M \text{ tests} | w \notin S) \leq \left(\frac{1}{q}\right)^M.$$

Proof: If w is in S, then $w = \sum_i \alpha_i u_i$ is the linear combination of the selected u vectors. Since by construction $u_i H^{(j)} z^{\mathrm{T}} = 0$ for all selected u_i, we have $wH^{(j)}z^{\mathrm{T}} = \sum_i \alpha_i u_i H^{(j)} z^{\mathrm{T}} = 0$. Such w will pass all M tests.

To quantify the false-positive probability, we notice that for any chosen z vector, a w vector can pass all M tests if and only if w is orthogonal to all M column vectors $H^{(1)}z^{\mathrm{T}}$ to $H^{(M)}z^{\mathrm{T}}$. If we horizontally concatenate the column vectors $H^{(1)}z^{\mathrm{T}}$ to $H^{(M)}z^{\mathrm{T}}$, we can form an $N \times M$ matrix. If this $N \times M$ matrix is of full column rank M, then there are exactly q^{N-M} such w vectors that can be in the null space of $\{H^{(1)}z^{\mathrm{T}}, \ldots, H^{(M)}z^{\mathrm{T}}\}$. As a result, the false-positive probability becomes

$$\mathsf{prob}(w \text{ passes all } M \text{ tests, but } w \text{ is not in } S | w \notin S) = \frac{q^{N-M} - q^{\mathrm{Rank}(S)}}{q^N - q^{\mathrm{Rank}(S)}} \leq \left(\frac{1}{q}\right)^M.$$

For the following, we thus only need to show that the $N \times M$ matrix $(H^{(1)}z^{\mathrm{T}}, \ldots, H^{(M)}z^{\mathrm{T}})$ is of full column rank M.

By our construction, z has at least M nonzero coordinates. Denote the indices of the nonzero coordinates by j_1–j_M, and the corresponding coordinates of z by z_{j_1}–z_{j_M}. Now, we focus on the j_1–j_M rows of the $N \times M$ matrix, which is

$$\begin{bmatrix} h_{j_1}^{(1)} z_{j_1} & h_{j_1}^{(2)} z_{j_1} & \cdots & h_{j_1}^{(M)} z_{j_1} \\ h_{j_2}^{(1)} z_{j_2} & h_{j_2}^{(2)} z_{j_2} & \cdots & h_{j_2}^{(M)} z_{j_2} \\ \vdots & \vdots & \cdots & \vdots \\ h_{j_M}^{(1)} z_{j_M} & h_{j_M}^{(2)} z_{j_M} & \cdots & h_{j_M}^{(M)} z_{j_M} \end{bmatrix}.$$

The matrix is the product of the diagonal matrix consisting of z_{j_1}–z_{j_M} and the hash matrix in (2). Since all z_{j_1}–z_{j_M} are nonzero, and by our construction of hash matrices in (2), both matrices are of full rank, hence the above $M \times M$ submatrix must be of full rank. Therefore, the $N \times M$ matrix $(H^{(1)}z^{\mathrm{T}}, \ldots, H^{(M)}z^{\mathrm{T}})$ is of full column rank. The proof is complete. ∎

It is worth noting that a naive way of avoiding false-positive events is to increase the underlying finite field size $\mathrm{GF}(2^b)$, which is not viable for WMNs. One reason is that to achieve the level of false-positive probability needed in our CCACK scheme ($M = 4$), we need $b = 32$, which uses 4 B to represent a single coding symbol. The size of each forward coding vector and each coded feedback vector thus grows from 32×1 to 32×4 B, which substantially increases the overhead. An even greater challenge is that each addition and multiplication coding operation now operates on $\mathrm{GF}(2^{32})$. A table lookup method has to have $2^{32} \times 2^{32}$ 4-B entries, which takes prohibitively 64 million TB to store. Even if we use log-exp lookup tables that only require $\mathrm{O}(2^b)$ space, the total space is still prohibitively large for $b = 32$ (16 GB). Since table lookup is impossible, one thus has to use online polynomial-based computation each time a coding operation needs to be performed, which is beyond today's microprocessor capability.

C. Rate Control

The cumulative coded feedback scheme in CCACK helps nodes to determine when they should stop transmitting packets for a given batch, but it does not tell anything about *how fast* nodes should transmit before they stop. Unlike in MORE, in CCACK we cannot use receptions from upstream to trigger new transmissions since the goal is exactly to stop the upstream nodes, when the downstream nodes have sufficiently enough packets. In addition, we want to apply rate control to the source as well and not only to the FNs.

The rate control algorithm in CCACK uses a simple credit scheme, which is oblivious to loss rates, but aware of the existence of other flows in the neighborhood, and leverages CCACK's cumulative coded acknowledgments.

For each flow f at a node, we define the "differential backlog"[8] as

$$\Delta Q^f = \dim(B_{\mathrm{in}}^f) - \dim(B_H^f) \tag{5}$$

[8]Our solution is inspired by the theoretical backpressure based rate control algorithms [21]. The difference is that, instead of queue lengths, we use innovative coded packets to define a *cumulative differential backlog* for flow f at every node with respect to all its downstream nodes for that flow.

where B_H^f is the set of vectors in $B_{\mathrm{rx}}^f \cup B_{\mathrm{tx}}^f$ marked as H, and $\dim(S)$ denotes the number of linearly independent packets in the set S. Note that $\dim(B_H^f) \leq \dim(B_{\mathrm{in}}^f)$. ΔQ^f is the difference between the number of innovative packets at a given node and the cumulative number of innovative packets at its downstream FNs for flow f. As we saw in Section III-A, when $\Delta Q^f = 0$, i.e., $\dim(B_{\mathrm{in}}^f) = \dim(B_H^f)$, the node stops transmitting packets for flow f. Note that for the destination of flow f, $\Delta Q^f = 0$.

We also define the relative differential backlog $\Delta Q_{\mathrm{rel}}^f$ for each flow f as

$$\Delta Q_{\mathrm{rel}}^f = \frac{\Delta Q^f}{\Delta Q^f + \Delta Q_N} \tag{6}$$

where ΔQ_N is the total differential backlog of all the neighbor nodes for all flows, calculated as follows. Every time t_i a node n_j broadcasts a coded data packet, it includes in the packet header its current total differential backlog $\Delta Q_{n_j}^{\mathrm{tot}}(t_i)$ of all flows crossing that node. All nodes that hear this packet update their ΔQ_N as follows:

$$\Delta Q_N = \Delta Q_N + \Delta Q_{n_j}^{\mathrm{tot}}(t_i) - \Delta Q_{n_j}^{\mathrm{tot}}(t_{i-1}) \tag{7}$$

where $\Delta Q_{n_j}^{\mathrm{tot}}(t_{i-1})$ is the last advertised $\Delta Q_{n_j}^{\mathrm{tot}}$ by node n_j. A node removes the contribution of its neighbor node n_j to the total ΔQ_N by updating $\Delta Q_N = \Delta Q_N - \Delta Q_{n_j}^{\mathrm{tot}}(t_{i-1})$ if it does not hear a new packet from n_j before a timeout.

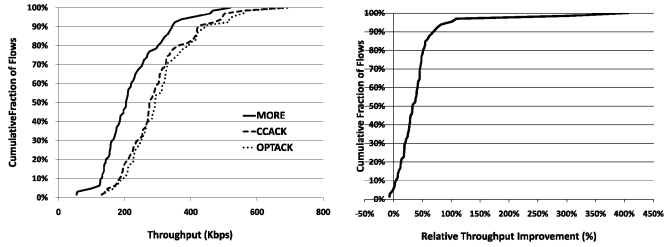
Every node in CCACK (including the source and the destination) maintains a credit counter for each flow. Every time there is a transmission opportunity for a node A, one flow f is selected in a round-robin fashion, among those flows with $\Delta Q^f > 0$, and the credit counter of that flow is incremented by $\alpha \times \Delta Q_{\mathrm{rel}}^f + \beta$. If the counter is positive, the node transmits one coded packet for flow f and decrements the counter by one, otherwise it selects the next flow. The credit increment $\alpha \times \Delta Q_{\mathrm{rel}}^f + \beta$ is larger for flows with large "backpressure," thus packets of such flows will be transmitted more frequently. For our implementation, we selected $\alpha = 5/6$ and $\beta = 1/6$. If $\Delta Q_{\mathrm{rel}}^f = 1$, then $\alpha \times \Delta Q_{\mathrm{rel}}^f + \beta = 1$ and the credit counter will always remain equal to 1, effectively allowing the node to always transmit.

IV. Evaluation

A. Methodology

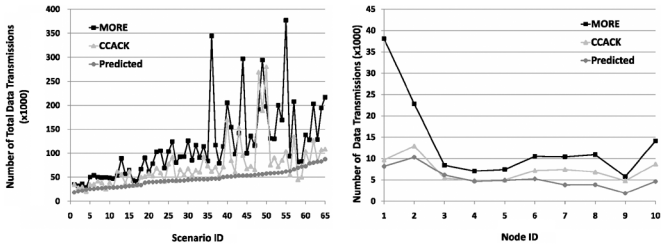
We evaluated the performance of CCACK and compared it against MORE using extensive simulations. We used the Glomosim simulator [22], a widely used wireless network simulator with a detailed and accurate physical signal propagation model. Glomosim simulations take into account the packet header overhead introduced by each layer of the networking stack and also the additional overhead introduced by MORE or CCACK. For the implementation of MORE, we followed the details in [9].

We simulated a network of 50 static nodes placed randomly in a 1000×1000 m^2 area. The average radio propagation range was 250 m, the average sensing range was 460 m, and the channel capacity was 2 Mb/s. The *TwoRay* propagation model was used and combined with the Rayleigh fading model to make the simulations realistic. Because of fading, transmission



(a) (b)

Fig. 5. Throughput comparison between CCACK and MORE: single flow. (a) CDF of throughputs achieved with MORE and CCACK. (b) CDF of relative throughput improvement of CCACK over MORE.



(a) (b)

Fig. 6. Total number of data transmissions with MORE and CCACK and predicted number of transmissions based on MORE's credit calculation algorithm, with a single flow. (a) Total number of data transmissions per scenario. (b) Total number of data transmissions per node for one scenario.

and sensing range are not fixed, but vary significantly around their average values.

We simulated each protocol in nine different randomly generated topologies, i.e., placement of the 50 nodes. We varied the number of concurrent flows from 1 up to 4. For a given number of flows, we repeated the simulation 10 times for each topology, each time randomly selecting a different set of source–destination pairs, i.e., we had a total of 90 different scenarios for a given number of flows. In each scenario, every source sent a 12-MB file consisting of 1500-B packets.

Following the methodology in [8] and [9], we implemented an ETX measurement module in Glomosim that was run for 10 min prior to the file transfer for each scenario to compute pairwise delivery probabilities. There was no overhead due to loss rate measurements during the file transfer.

It is generally known that the full benefit of OR over traditional routing is exposed when the destination is several hops away from the source [9]. In those cases, OR reduces the overhead of retransmissions incurred by high loss rates and increased self-interference. Hence, for the single-flow experiment, among the 90 flows we simulated, we show the results of the 65 flows for which the destination was not within the transmission range of the source (with ETX shortest paths of 3–9 hops). For the evaluation with multiple flows, we kept scenarios with flows of shorter paths when those flows interfered with other flows. On the other hand, we do not show the results for scenarios where the multiple flows were out of interference range of each other since those scenarios are equivalent to the single-flow case. We were left with 68 scenarios with two flows and 69 scenarios with three and four flows.

B. Single Flow

We begin our evaluation with a single flow. Fig. 5(a) plots the cumulative distribution function (cdf) of the throughputs of the 65 flows with MORE and CCACK. We observe that CCACK outperforms MORE: The median throughput with CCACK and MORE is 276 and 205 kb/s, respectively.

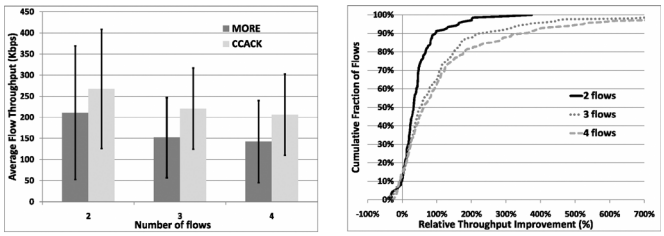
To evaluate the performance loss in CCACK due to the facts that in some (rare) occasions CCACK may not solve the collective space problem and that the false-positive probability of CCACK is small $\frac{1}{256^4}$ but not zero, we implemented a third scheme called OPTACK. OPTACK uses the *optimal* ACK scheme: When a node A hears a packet from a downstream node C, we allow node A to directly read from node C's B_{in} buffer all the basis vectors C has received so far. Hence, nodes in OPTACK maintain the exact collective space of the downstream nodes and can compare it to their own space and make the right decision whether to stop or not, without any false negatives or false positives. For a fair comparison to CCACK, we wanted to keep the overhead due to ACK the same for both protocols. Hence, nodes in OPTACK still embed only one 32-B ACK vector in each data packet, which is ignored by the upstream nodes.[9] In OPTACK, nodes update their knowledge of the collective space of their downstream nodes only when they overhear data transmissions. Therefore, the operation of OPTACK is similar to that of CCACK, but is completely free from any collective space problem and any false-positive/negative problem. Fig. 5(a) shows that OPTACK only slightly outperforms CCACK. The median throughput OPTACK is 292 kb/s, only 6% higher than with CCACK. This result shows that CCACK effectively solves (most of) the collective space and the false-positive problems with low complexity and very little performance loss.

Fig. 5(b) plots the cdf of the relative throughput improvement of CCACK over MORE for all 65 flows, defined as $\frac{T^f_{\text{CCACK}} - T^f_{\text{MORE}}}{T^f_{\text{MORE}}} \times 100\%$, where T^f_{CCACK} and T^f_{MORE} are the throughput of flow f with CCACK and MORE, respectively. We observe that CCACK achieves a higher throughput than MORE for 95% of the flows. The median gain of CCACK over MORE is 34%. However, for some challenged flows with the destination several hops away from the source, the throughput with CCACK is much higher than with MORE; the 90th percentile of the gain is 71%.

Where Does the Gain for CCACK Come From?: Fig. 6(a) plots the total number of data transmissions with CCACK and MORE in each of the 65 scenarios, as well as the predicted number of transmissions in each scenario using MORE's offline ETX-based credit calculation algorithm. The 65 scenarios are sorted with respect to the predicted number of transmissions.

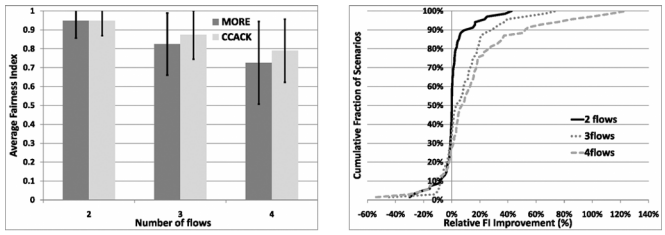
We observe that nodes with MORE perform a higher number of transmissions than the predicted number in all 65 scenarios. The actual number is often more than twice the predicted number, and in some scenarios up to 6–7× the predicted number. This shows that the credit calculation algorithm based on offline ETX measurements mispredicts the required number of transmissions even in the absence of background traffic. The cause is self-interference that changes the loss rates, which in most cases become higher than in a quiet network, where

[9]Apparently, a scheme like OPTACK cannot be implemented in practice; it is feasible only in a simulator.



(a) (b)

Fig. 7. Throughput comparison between CCACK and MORE: multiple flows. (a) Average per-flow throughputs (bars) and standard deviations (lines). (b) CDF of relative throughput improvement of CCACK over MORE.



(a) (b)

Fig. 8. Fairness comparison between CCACK and MORE: multiple flows. (a) Average per-scenario FIs (bars) and standard deviations (lines). (b) CDF of relative FI improvement of CCACK over MORE.

only probing traffic exists [18]. Moreover, the source in MORE keeps transmitting packets until it receives an ACK from the destination. With long paths, this may result in a large number of unnecessary transmissions as the ACK travels toward the source.

In contrast, the number of data transmissions with CCACK is much lower than with MORE in all but two scenarios. In most scenarios it is close to the predicted number, and in some cases it is even lower. This shows the effectiveness of the coded feedback mechanism in CCACK, combined with the online rate control mechanism of Section III-C.

Fig. 6(b) shows an example (one scenario) of how data transmissions are distributed over the FNs. Nodes are sorted with respect to their ETX distance to the destination, i.e., node 1 is the source and node 10 is the FN closest to the destination. With MORE, the source and the FN closest to the source perform many more transmissions than the remaining FNs, (2.5–7.6× and 1.4–4.6×, respectively). In contrast, CCACK ensures that these nodes stop transmitting when the remaining downstream FNs have received enough innovative packets. Overall, with CCACK, all 10 nodes perform fewer transmissions than with MORE. The savings range from 17% (for node 9) up to 74% (for the source).

C. Multiple Flows

We now evaluate CCACK and MORE with multiple concurrent flows. Here, in addition to throughput, we compare the two protocols in terms of fairness, using *Jain's fairness index* (FI) [23]. Jain's FI is defined as $(\sum x_i)^2/(n \times \sum x_i^2)$, where x_i is the throughput of flow i and n is the total number of flows. The value of Jain's FI is between 0 and 1, with values closer to 1 indicating better fairness.

Throughput Comparison: Fig. 7(a) and (b) compares throughput with CCACK and MORE with two, three, and four flows. Fig. 7(a) plots the average per-flow throughput with the two protocols as a function of the number of flows. We observe that CCACK outperforms MORE by 27% on average in the two-flow case, and by 45% on average in the three-flow and four-flow cases. Note that the gain of CCACK is higher with a larger number of flows, when the congestion level becomes higher, causing substantial changes to the link loss rates. By quickly and accurately stopping transmissions for a given flow at nodes whose downstream nodes have collectively received a sufficient number of packets, a large amount of bandwidth is saved, which can be used by the nodes or their neighbors for transmitting packets for other flows. In contrast, the gain of MORE over traditional routing in [9] drops as the number of concurrent flows increases.

Fig. 7(b) plots the cdf of per-flow relative throughput improvement with CCACK over MORE, as defined in Section IV-B, with two, three, and four flows. CCACK improves per-flow throughputs for more than 85% of the flows in all three cases (with two, three, and four flows). The median improvement is 33%, 55%, and 62%, respectively, with two, three, and four concurrent flows. Similar to the single-flow experiments, some starving flows with MORE show a several-fold improvement with CCACK: The 90th percentiles are 98%, 242%, and 364% in the two-, three-, and four-flow cases, respectively.[10]

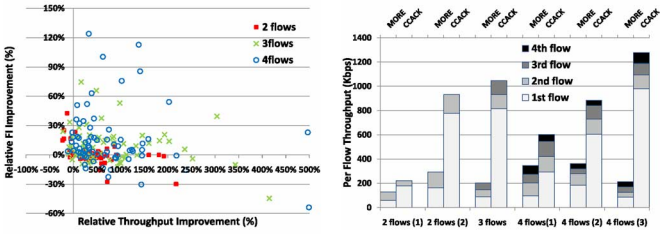
Fairness Comparison: Fig. 8(a) and (b) compares fairness with CCACK and MORE in the cases of two, three, and four concurrent flows. Fig. 8(a) plots the average FI with the two protocols. We observe that the average FI is the same with the two protocols in the two-flow case, but is higher with CCACK in the three-flow and four-flow cases by 5.8% and 8.8%, respectively.

Fig. 8(b) plots the cdf of per-scenario relative FI improvement with CCACK over MORE, defined similarly to the relative throughput improvement in Section IV-B, with two, three, and four flows. We observe that CCACK improves fairness in more scenarios as the number of flows in the network increases—in 40% of the two-flow scenarios, 65% of the three-flow scenarios, and 72% of the four-flow scenarios. Similar to the throughput results, the improvement is very large for some scenarios: up to 74% with three flows, and up to 124% with four flows. This shows again that CCACK improves throughput for some challenged flows, which completely starve with MORE.

Throughput versus Fairness: We now investigate more closely the relationship between throughput and fairness. Fig. 9(a) shows the scatterplots of the relative total throughput improvement per scenario versus the relative FI improvement per scenario in the two-, three-, and four-flow experiments.

We observe that CCACK improves at least one of the two metrics in all but two scenarios [two points in the third quadrant of Fig. 9(a)]. There are a few points in the second quadrant for all three cases; these are scenarios, where CCACK improves fairness, at the cost of a small total throughput decrease. The majority of the points for the two-flow case are gathered in the first and fourth quadrants, i.e., CCACK either improves throughput at the cost of a (typically) small decrease in fairness, or it improves both metrics. The majority of the points are gathered in

[10]Note that the heavy tails of the three-flow and four-flow curves are not shown in Fig. 7(b) for better clarity.



(a) (b)

Fig. 9. Investigating the relationship between throughput and fairness. (a) Scatterplot of relative throughput improvement versus relative FI improvement with two, three, and four flows. (b) Per-flow throughputs with MORE and CCACK for the six scenarios with the largest FI decrease under CCACK.

TABLE I
CODING OVERHEAD IN CCACK IN TERMS OF $\mathrm{GF}(2^8)$ MULTIPLICATIONS. OPERATIONS MARKED WITH (*) ARE COMMON IN MORE AND CCACK

Operation	Avg.	Std. Dev.
Packet Transmission		
Coded pkt construction (src/FNs)*	48000/27240	0/13128
ACK vector construction	11584	5369
Total (src/FNs)	59584/38824	5369/10021
Packet Reception		
Independence check*	326	156
H_tests	428	316
Rank of H pkts in $B_{rx} \cup B_{tx}$	292	169
Total	1046	416

the first quadrant for the three-flow and four-flow cases. This shows that as the number of flows increases, CCACK improves both throughput and fairness in most scenarios.

We now take focus on a few points in the fourth quadrant in Fig. 9(a), corresponding to scenarios where FI is reduced by more than 20% with CCACK. There are two two-flow, one three-flow, and three four-flow scenarios (points). Note that all six of these points correspond to large throughput improvements, from 72% up to 499%. One may wonder if these improvements are achieved at the cost of compromising the fairness, i.e., throughput of only one flow increases significantly, causing starvation to the remaining flows.

Fig. 9(b) shows that this is not the case. This figure plots the individual per-flow throughputs with MORE and CCACK for these six scenarios. We observe that in all but two cases, CCACK improves throughput of *all* flows involved. The reduction in the FI actually comes from the fact that throughput improvement is much higher for some flows than for some others, and not as a result of starvation of some flows. Take the last scenario [*4 flows (3)*] as an example. CCACK improves throughput of the first flow by 11× (from 85 to 978 kb/s), but also improves throughputs of the other three flows by 183%, 108%, and 113%.

A closer look at the topology of that scenario revealed an interesting situation. We found that the first flow was a 1-hop flow, whose FN belt overlapped with the FN belt of the fourth 9-hop flow, near the source of the fourth flow. With MORE, it took a long time for the destination of the fourth flow to decode each batch. During that time, the source as well as every other FN kept transmitting packets for that batch. Since routers in MORE serve flows in a round-robin fashion, they kept switching between the first and the fourth flow. In other words, a short flow was starved because of a long flow, achieving a throughput of only 85 kb/s, although there was no need for the routers to forward packets of the long flow. In contrast, with CCACK, the coded acknowledgment scheme quickly caused the source of the fourth flow to stop transmitting packets once the remaining FNs had enough packets. Hence, the FNs near the source were able to forward packets only for the 1-hop flow, increasing its throughput to 978 kb/s.

D. CCACK's Overhead

Finally, we estimate CCACK's overhead compared to MORE. Similar to [9], we discuss three types of overhead: coding, memory, and packet header overhead.

Coding Overhead: Unavoidably, CCACK's coding overhead is higher than MORE's since routers have to perform additional operations both when transmitting and when receiving a packet. However, all the additional CCACK operations are performed on N-byte *vectors* instead of the whole K-byte *payload*. Therefore, in practical settings (e.g., with $N = 32$ and $K = 1500$), the coding overhead of CCACK is expected to be comparable to that of MORE.

To verify this, we measured the per-packet cost of the various operations performed upon a packet transmission/reception averaged over all packets transmitted/received at all nodes in the 90 simulation scenarios of Section IV-B. Table I provides the average values and the standard deviations. The costs are given in terms of $\mathrm{GF}(2^8)$ multiplications, which are the most expensive operations involved in coding/decoding [9].

Construction of an ACK vector in CCACK requires on average 11 584 multiplications. The total coding cost in transmitting a packet (i.e., constructing a coded packet and an ACK vector) in CCACK is only 24% higher than MORE's, assuming the worst-case cost for packet encoding (48 000 multiplications). If we use instead the average packet encoding cost at FNs (27 240 multiplications), the total cost of transmitting a packet in CCACK is only 38 824 multiplications, i.e., lower than MORE's encoding cost at the source.[11]

When receiving a packet, the cost of checking for independence (also in MORE) requires on average only 326 multiplications. The additional operations of performing the $\mathrm{H\_tests}$ (if the received packet comes from downstream) and maintaining the rank of the H packets in $B_{\mathrm{rx}} \cup B_{\mathrm{tx}}$ (if a received packet from downstream passes all M $\mathrm{H\_tests}$) require on average only 428 and 292 multiplications, respectively, i.e., their costs are comparable to the independence check cost. The total cost of packet reception operations in CCACK is only 1.7% of the total packet transmission cost. Hence, the bottleneck operation in CCACK is preparing a packet for transmission at an FN with 32 innovative packets in B_{in}.

In [9], the authors found that the bottleneck operation in MORE (packet encoding at the source) takes on average 270 μs on a low-end Celeron 800 MHz, limiting the maximum achievable throughput with MORE to 44 Mb/s with a 1500-B packet. In CCACK, the cost of the bottleneck operation is 24% higher, so we can expect a maximum achievable throughput of 35 Mb/s. Note that this value is still higher than the effective bit rate of current 802.11a/g WMNs [24].

Memory Overhead: Same as in MORE, routers in CCACK maintain an innovative packet buffer B_{in} for each flow, and also

[11]Note that the source in CCACK does not have to construct an ACK vector, and hence the cost at the source is the same as in MORE.

a 64-kB lookup table for reducing the cost of the $\mathrm{GF}(2^8)$ multiplications [9]. With a packet size of 1500 B, the size of B_{in} is 48 kB. The extra overhead in CCACK comes from the two additional buffers B_{rx} and B_{tx}, which store, however, only 32-B vectors and not whole packets. In our implementation, the total size of B_{rx} and B_{tx} is $2 \times 5 \times 32 \times 32 = 10$ kB, which is relatively small compared to the size of MORE's structures.

Header Overhead: The N-byte ACK vector and the total differential backlog $\Delta Q_{n_j}^{\mathrm{tot}}$ are the two fields we add to the MORE header. The differential backlog per flow is bounded by the batch size N. With $N = 32$, 2 B are enough to support up to 2048 flows, and the total size of the two fields is equal to 34 B. However, in CCACK, we do not include in the packet header the transmission credits for the FNs, which are required in MORE. This can potentially make CCACK's header smaller than MORE's depending on the number of FNs.

V. Implementation and Testbed Evaluation

In this section, we describe an implementation of CCACK on a WMN testbed and present experimental results comparing CCACK and MORE.

A. Testbed Description

Our testbed, Mesh@Purdue (MAP) [25], currently consists of 22 mesh routers (small form factor desktops) deployed on two floors of two academic buildings at Purdue University. Each router has an Atheros 5212-based 802.11a/b/g wireless radio operating in 802.11b *ad hoc* mode and attached to a 2-dBi rubber duck omnidirectional antenna. The mesh routers run Mandrake Linux 10.1 (kernel 2.6.8–12), and the open-source *madwifi* driver [26] is used to enable the wireless cards. IP addresses are statically assigned. The testbed deployment environment is not wireless-friendly, having floor-to-ceiling office walls, as well as laboratories with structures that limit the propagation of wireless signals and create multipath fading.

B. Implementation Details

NC-based wireless protocols (e.g., [9] and [27]) 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 CCACK at the application layer, using broadcast sockets. For a fair comparison, we also implemented MORE at the application layer, following all the details in [9]. We note that such an implementation unavoidably results in some performance degradation for both protocols, compared to an implementation closer to the MAC layer, from crossing the kernel-user boundary. Actually, the degradation is larger for CCACK, as we explain later in this section.

Our implementation handles only synthetic traffic, i.e., data packets are generated within the MORE or CCACK application, similarly to the implementation in [28], in which packets are generated within Click. The layer-2.5 header of MORE or CCACK is part of the application-layer packet payload. The source initially generates N random payloads for the current batch and mixes them every time it wants to transmit a packet. It then appends the MORE or CCACK header and delivers the resulting packet to the IP layer, which in turn delivers the packet to the MAC for transmission. Packets are broadcast at the MAC layer, and every neighbor node can hear them. When a node receives a packet, it extracts and processes the protocol-specific header from the payload. If the node is an FN (i.e., it finds its ID in the FN list in the header), it also uses the coding vector (also included in the header) to check for linear independence. If the received packet is innovative, the rest of the payload is stored for future mixing (if the node is an FN) or for decoding (if the node is a receiver).

1) Removing the Dependence on the MAC Layer: In an ideal implementation at layer 2.5, a node running either MORE or CCACK transmits a packet when: 1) *the 802.11 MAC allows* and 2) *the credit counter is positive*. In our application-layer implementation, we cannot get any feedback from the MAC, and hence, we had to modify the transmission policy for the two protocols.

In our implementation of MORE, the application instead delivers packets to the IP when only the second condition holds and there is enough space in the socket buffer. From the IP layer, the packets are delivered to the wireless driver stored at the card's queue for transmission at a later time. Similar to a layer-2.5 implementation, the credit counter is incremented every time a packet is received from an upstream node, and decremented after every transmission.

Unlike in MORE, the credit counter in CCACK is incremented every time the MAC layer signals a transmission opportunity. Since the application cannot know when there is a transmission opportunity without access to the MAC layer, we approximate the number of transmission opportunities via the following heuristic. A node increments its credit counter every time it hears a data packet transmission from another node by a fraction of $1/K$ of the actual increment determined by the rate control algorithm, where K is the number of nodes in the node's neighborhood. The intuition behind this is that with a fair MAC layer, every node in a neighborhood would roughly get an equal number of transmission opportunities. To avoid possible deadlock situations, where every node in a neighborhood is waiting for another node to transmit, we also use a timeout equal to one data packet transmission time, after which a node always increments its credit counter.

2) Dealing With Queue Sizes: With a layer-2.5 implementation [9] of an NC-based protocol, a precoded packet is always available awaiting for transmission. If another innovative packet is received before the precoded packet is transmitted, the precoded packet is updated by multiplying the newly received packet with a random number and adding it to the precoded packet. This approach ensures that every transmitted packet includes information from all the received innovative packets, including the most recent ones.

In contrast, in our implementation, we have no control over a packet once it leaves the application layer, and we cannot update the coded packets buffered at the socket buffer or awaiting for transmission at the card's queue if a new innovative packet is received. This inefficiency can have a significant impact on the performance of the two protocols. If a packet is queued either at the IP or at the MAC layer for a long time, it may not contain information from all the received packets so far. Even worse, the downstream nodes may have already received enough packets from the current batch, in which case the enqueued packets should not be transmitted at all. To avoid this problem, we limit the socket buffer size to one packet and the card's queue length

to three packets in order to minimize the time from the moment a packet is created at the application layer till the moment the packet is actually transmitted.

3) Dealing With End-to-End ACKs: In both protocols, a destination sends an end-to-end ACK back to the source every time it decodes a batch. It is critical for the performance of the protocols that these ACKs are propagated to the source in a fast and reliable way since, otherwise, the source cannot move to the next batch. To provide reliability, we *unicast* the ACKs at the MAC layer, and we implemented an additional ACK-retransmission scheme at the application layer for both protocols. For fast ACK propagation, we prioritize ACKs over data transmissions by leveraging the TOS bits ("TOS field") of the IP header and the priority features in Linux routing [29].

In addition to the two protocols, we also implemented an ETX measurement module, the same as the one we used in our simulations. The source code for the two protocols and the ETX module together is over 7800 lines of C code.

C. Experimental Setup

In the implementation of the two protocols, we used the same parameters as in our simulation study in Section IV. In all the experiments, the bit rate of the wireless cards was set to 2 Mb/s and the transmission power to 16 dBm. We disabled RTS/CTS for unicast frames as most operational networks do. With these settings, the length of the shortest ETX paths between different nodes is 1–5 hops in length, and the loss rates of the links vary from 0% to 91%, with an average value of 36%.

We experimented with 20 single-flow scenarios (i.e., randomly selected source–destination pairs), 10 two-flow scenarios, and six three-flow scenarios. For each scenario, we first ran the ETX module to collect the pairwise loss rates and ETX metric for each link of our testbed, and then we ran the two protocols, MORE and CCACK, in sequence. With both protocols, the source sent a 2.3-MB file consisting of 1460-B packets.

As we have explained in Section IV-A, the gain of CCACK over MORE is more pronounced with flows over long paths, where the destination is several hops away from the source. Unfortunately, the size of our testbed limited our choices in flow selection. Hence, in the single-flow experiments, we also included flows where the destination was 2 hops away from the source. Similarly, the small size of the testbed resulted in a large fraction of the nodes being within sensing range of each other. This prevented us from increasing the total number of flows beyond three since the medium became congested, resulting in very poor performance for both protocols.[12] These two limitations, along with the implementation limitations we discussed in Section V-B1, limited the gains of CCACK over MORE compared to the simulations results in Section IV.

D. Experimental Results

Fig. 10(a) and (b) compare throughput with CCACK and MORE with one, two, and threeflows. In Fig. 10(a), we observe that CCACK outperforms MORE by 36% on average in the one-flow scenarios, by 11% in the two-flow scenarios, and by 15% on average in the three-flow scenarios. Fig. 6(b) plots the

[12]As explained in [9], intraflow NC-based protocols cannot increase the capacity of the network, and they can only improve throughput as long as the total load remains below the network capacity.

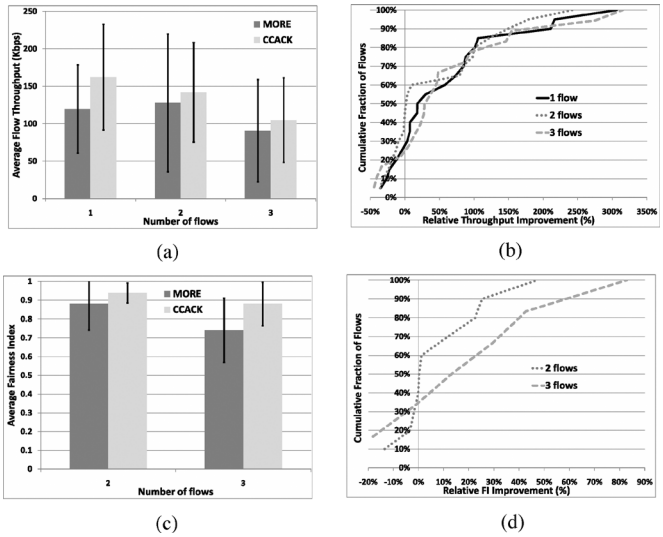


Fig. 10. Testbed evaluation. (a) Average per-flow throughputs (bars) and standard deviations (lines). (b) CDF of relative throughput improvement of CCACK over MORE. (c) Average per-scenario FIs (bars) and standard deviations (lines). (d) CDF of relative FI improvement of CCACK over MORE.

cdf of per-flow relative throughput improvement with CCACK over MORE, as defined in Section IV-B, with one, two, and three flows. CCACK improves per-flow throughputs for 72% of the flows in the one-flow scenarios, 55% of the flows in the two-flow scenarios, and 75% of the flows in the three-flow scenarios. The median improvement is 18%, 3%, and 28%, respectively, in the one-, two-, and three-flow scenarios. These gains are lower than the ones observed in the simulation results in Section IV due to the limitations we discussed in Section V-C. In spite of these limitations though, our results still demonstrate the benefit of CCACK over MORE in the case of challenged flows. We observe that about 20% of the flows in one-flow and two-flow scenarios, and 17% of the flows in the three-flow scenarios show a several-fold throughput improvement with CCACK, up to 3×, 2.4×, and 3.2×, respectively.

Fig. 10(c) and (d) compares fairness with CCACK and MORE in the cases of two and three concurrent flows. Fig. 10(c) plots the average FI with the two protocols. We observe that the average FI is higher with CCACK in both the two-flow and three-flow case by 5.7% and 18.9%, respectively. These values are actually higher than the simulation results. Due to the small size of the testbed, the network gets more easily congested, even with two flows, and CCACK's backpressure-inspired credit mechanism is very effective in allocating the medium's bandwidth fairly among contending flows. Fig. 10(d) plots the cdf of per-scenario relative FI improvement with CCACK over MORE. CCACK improves fairness in 60% of the two-flow scenarios and 65% of the three-flow scenarios, and the gains can be as high as 83% in some challenged scenarios.

VI. Conclusion

The use of random linear NC has significantly simplified the design of opportunistic routing (OR) protocols by removing the need of coordination among forwarding nodes for avoiding duplicate transmissions. However, NC-based OR protocols face a new challenge: *How many coded packets should each forwarder transmit?*

In this paper, we presented a novel approach to overcoming this challenge through the design of CCACK, a new efficient NC-based OR protocol. Instead of avoiding feedback exchange, CCACK encodes feedback messages in addition to encoding data packets. A novel Cumulative Coded ACKnowledgment scheme allows nodes in CCACK to acknowledge network coded traffic to their upstream nodes in a simple and efficient way, oblivious to loss rates and with negligible overhead. The cumulative coded acknowledgment scheme in CCACK also enables an efficient credit-based, rate control algorithm. Our experiments on a 22-node 802.11 WMN testbed show that compared to MORE, a state-of-the-art NC-based OR protocol, CCACK improves both throughput and fairness, by up to 3.2× and 83%, respectively, with average improvements of 11%–36% and 5.7%–8.3%, respectively, for a varying number of concurrent flows. Our extensive simulations show that the gains are actually much higher in large networks, with longer routing paths between sources and destinations.

REFERENCES

[1] "MIT roofnet," [Online]. Available: http://www.pdos.lcs.mit.edu/roofnet

[2] "Bay area wireless users group," [Online]. Available: http://www.bawug.org

[3] "Seattle wireless," [Online]. Available: http://www.seattlewireless.net

[4] D. Aguayo, J. Bicket, S. Biswas, G. Judd, and R. Morris, "Link-level measurements from an 802.11b mesh network," in *Proc. ACM SIGCOMM*, Aug. 2004, pp. 121–132.

[5] D. B. Johnson and D. A. Maltz, *Dynamic Source Routing in Ad Hoc Wireless Networks*. Norwell, MA: Kluwer, 1996.

[6] C. E. Perkins and E. M. Royer, "Ad hoc on-demand distance vector routing," in *Proc. IEEE WMCSA*, Feb. 1999, pp. 90–100.

[7] J. Bicket, D. Aguayo, S. Biswas, and R. Morris, "Architecture and evaluation of an unplanned 802.11b mesh network," in *Proc. ACM MobiCom*, 2005, pp. 31–42.

[8] S. Biswas and R. Morris, "ExOR: Opportunistic multi-hop routing for wireless networks," in *Proc. ACM SIGCOMM*, 2005, pp. 133–144.

[9] S. Chachulski, M. Jennings, S. Katti, and D. Katabi, "Trading structure for randomness in wireless opportunistic routing," in *Proc. ACM SIGCOMM*, 2007, pp. 169–180.

[10] C. Gkantsidis, W. Hu, P. Key, B. Radunovic, S. Gheorghiu, and P. Rodriguez, "Multipath code casting for wireless mesh networks," in *Proc. ACM CoNEXT*, 2007, Article no. 10.

[11] B. Radunovic, C. Gkantsidis, P. Key, and P. Rodriguez, "An optimization framework for opportunistic multipath routing in wireless mesh networks," in *Proc. IEEE INFOCOM*, 2008, pp. 2252–2260.

[12] D. S. J. D. Couto, D. Aguayo, J. C. Bicket, and R. Morris, "A high-throughput path metric for multi-hop wireless routing," in *Proc. ACM MobiCom*, 2003, pp. 134–146.

[13] X. Zhang and B. Li, "Optimized multipath network coding in lossy wireless networks," in *Proc. IEEE ICDCS*, 2008, pp. 243–250.

[14] X. Zhang and B. Li, "Dice: A game theoretic framework for wireless multipath network coding," in *Proc. ACM MobiHoc*, 2008, pp. 293–302.

[15] Y. Lin, B. Li, and B. Liang, "CodeOR: Opportunistic routing in wireless mesh networks with segmented network coding," in *Proc. IEEE ICNP*, 2008, pp. 13–22.

[16] J. Camp, V. Mancuso, O. Gurewitz, and E. Knightly, "A measurement study of multiplicative overhead effects in wireless networks," in *Proc. IEEE INFOCOM*, 2008, pp. 76–80.

[17] S. M. Das, H. Pucha, K. Papagiannaki, and Y. C. Hu, "Studying wireless routing link dynamics," in *Proc. ACM SIGCOMM/USENIX IMC*, 2007, pp. 327–332.

[18] Y. Li, L. Qiu, Y. Zhang, R. Mahajan, Z. Zhong, G. Deshpande, and E. Rozner, "Effects of interference on throughput of wireless mesh networks: Pathologies and a preliminary solution," presented at the HotNets–VI, 2007.

[19] R. Draves, J. Padhye, and B. Zill, "Routing in multi-radio, multi-hop wireless mesh networks," in *Proc. ACM MobiCom*, Sep. 2004, pp. 114–128.

[20] J. Park, M. Gerla, D. S. Lun, Y. Yi, and M. Medard, "Codecast: A network-coding-based ad hoc multicast protocol," *Wireless Commun.*, vol. 13, no. 5, pp. 76–81, Oct. 2006.

[21] L. Tassioulas and A. Ephremides, "Stability properties of constrained queueing systems and scheduling for maximum throughput in multihop radio networks," *IEEE Trans. Autom. Control*, vol. 37, no. 12, pp. 1936–1948, Dec. 1992.

[22] X. Zeng, R. Bagrodia, and M. Gerla, "Glomosim: A library for parallel simulation of large-scale wireless networks," in *Proc. PADS Workshop*, May 1998, pp. 154–161.

[23] R. K. Jain, D.-M. W. Chiu, and W. R. Hawe, "A quantitative measure of fairness and discrimination for resource allocation in shared computer systems Digital Equipment Corporation, Hudson, MA, Tech. Rep. DEC-TR-301, Sep. 1984.

[24] A. Kamerman and G. Aben, "Net throughput with IEEE 802.11 wireless LANs," in *Proc. IEEE WCNC*, 2000, vol. 2, pp. 747–752.

[25] "Mesh@Purdue," Purdue Univ., West Lafayette, IN, 2008 [Online]. Available: http://www.engineering.purdue.edu/mesh

[26] "MadWifi.net," Linux-Consulting, Reno, NV, 2000 [Online]. Available: http://madwifi.org

[27] S. Katti, H. Rahul, W. Hu, D. Katabi, M. Medard, and J. Crowcroft, "XORs in the air: Practical wireless network coding," in *Proc. ACM SIGCOMM*, Aug. 2006, pp. 243–254.

[28] "MORE source code," 2009 [Online]. Available: http://people.csail.mit.edu/szym/more

[29] B. Hubert, T. Graf, G. Maxwell, R. van Mook, M. van Oosterhout, P. B. Schroeder, J. Spaans, and P. Larroy, "Linux advanced routing and traffic control," 2004 [Online]. Available: http://lartc.org//lartc.html/

Dimitrios Koutsonikolas (M'09) received the Ph.D. degree in electrical and computer engineering from Purdue University, West Lafayette, IN, in 2010.

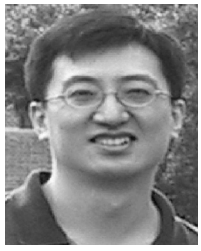
He is currently a Post-Doctoral Research Associate with Purdue University. His research interests are broadly on experimental wireless networking and mobile computing.

Dr. Koutsonikolas is a member of the Association for Computing Machinery (ACM) and USENIX. He won first place in the ACM Student Research Competition (SRC) at MobiCom 2009, and the third place in the same contest at MobiCom 2008.

Chih-Chun Wang (M'05) received the B.E. degree from National Taiwan University, Taiwan, in 1999, and the M.S. and Ph.D. degrees in electrical engineering from Princeton University, Princeton, NJ, in 2002 and 2005, respectively.

He joined the School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN, in 2006 as an Assistant Professor. His current research interests are in the graph-theoretic and algorithmic analysis of iterative decoding and of network coding. His other research interests fall in the general areas of optimal control, information theory, detection theory, coding theory, iterative decoding algorithms, and network coding.

Dr. Wang received the NSF CAREER Award in 2009.



Y. Charlie Hu (S'90–M'93–SM'07) received the Ph.D. degree in computer science from Harvard University, Cambridge, MA, in 1997.

He is an Associate Professor of electrical and computer engineering and computer science (by courtesy) with Purdue University, West Lafayette, IN. From 1997 to 2001, he was a Research Scientist with Rice University, Houston, TX. He has published over 120 papers in his research interests, which include operating systems, distributed systems, computer networking, and wireless networking.

Dr. Hu is a Senior Member of the Association for Computing Machinery (ACM). He received the NSF CAREER Award in 2003.