CCACK: Efficient Network Coding Based Opportunistic Routing Through Cumulative Coded Acknowledgments

Dimitrios Koutsonikolas, Chih-Chun Wang, Y. Charlie Hu
School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN 47907
{dkoutson, chihw, ychu}@purdue.edu

***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 practically zero overhead. In addition, the cumulative coded acknowledgment scheme in CCACK enables an efficient credit-based, rate control algorithm. Our evaluation shows that, compared to MORE, a state-of-the-art NC-based OR protocol, CCACK improves both throughput and fairness, by up to 20x and 124%, respectively, with average improvements of 45% and 8.8%, respectively.

I. Introduction

Wireless mesh networks (WMNs) are increasingly being deployed for providing cheap, low maintenance Internet access (e.g. [1, 2, 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, 6, 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.

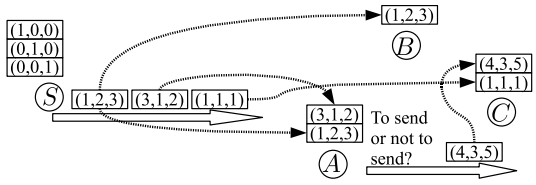


Fig. 1. The importance of knowing how many coded packets to transmit.

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 intra-flow 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. The challenge in NC-based OR protocols

We illustrate the main challenge in NC-based OR protocols with the example shown in Figure 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 $(1, 1, 1)$ coded 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 non-trivial 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 non-coded wireless traffic due to its zero overhead, does not suit network coded traffic. For the same example in Figure 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 forwarders 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 which 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 pre-computed 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, 14, 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 non-forwarding 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. 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.

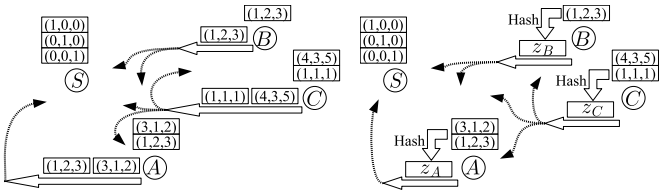
C. Our approach – Cumulative Coded Acknowledgments

In this paper, we propose CCACK, a new efficient NC-based OR protocol. Unlike MORE, FNs in CCACK decide how many packets to transmit in an online fashion, and this decision is completely oblivious to link loss rates.[1] 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 practically *zero 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, compared to [10].

Take the scenario in Figure 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 Figure 2(a).[2]

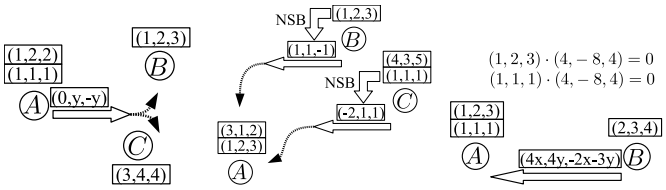
[1] By "oblivious to link loss rates" we mean here that loss rates are not taken into account in determining how many packets each FN should transmit. We still use MORE's loss rate based offline algorithm in CCACK to build the FN belt, for a fair comparison between the two protocols. We note though that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

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



(a) Uncoded Feedback (b) Coded Feedback

Fig. 2. Different types of feedback for network-coded traffic.



(a) NSB Feedback for unreliable multicast. (b) NSB Feedback for reliable unicast OR. (c) The false positive error of the NSB feedback.

Fig. 3. Null-Space-Based (NSB) feedback for unreliable multicast, for reliable unicast OR, and the corresponding false positive event.

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 bytes. To convey a space of dimension $\kappa \gg 1$ thus requires κ 32-byte vectors, which is too large to piggyback to normal forward traffic. The unreliability of the wireless channel further exacerbates the problem as the $\kappa \times 32$-byte 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 Figure 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. This thus drastically reduces the need of retransmitting feedback information over the unreliable wireless channel.

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].

We evaluate the performance of CCACK and compare it against MORE using extensive simulations. Our results show that CCACK improves both throughput and fairness over MORE, by 45% and 8.8%, respectively, on average. For some challenged flows which completely starve under MORE, CCACK increases throughput by up to 20x and fairness by up to 124%. In addition, the coding and memory overheads of CCACK are comparable to those of MORE.

II. Existing Coded Feedback Scheme

Coded feedback has been used in the past in a different context; in [20], a null-space-based (NSB) coded feedback scheme is used to enhance reliability of an NC-based multicast protocol for multimedia applications in mobile ad hoc networks. In this section, we review this scheme and identify two problems when trying to apply it to reliable unicast OR: the collective space problem and the false positive problem. These two problems motivate the need for a new cumulative coded feedback scheme which is a major component of CCACK.

Take Figure 3(a) for example. A batch of 3 packets are coded together and nodes A to C need to decode all three packets. Let B_v denote the buffer containing the innovative coding vectors received by A (which contains two vectors $(1,2,2)$ and $(1,1,1)$ in Figure 3(a)).

Since A has received fewer than 3 innovative packets, it informs its neighbor nodes that it needs more packets by appending to each coded packet a vector z_A satisfying

$$z_A \cdot v = 0, \quad \forall v \in B_v \tag{1}$$

Namely, the inner product between z_A and $v \in B_v$ is zero. There may be multiple choices of z_A that satisfy (1) (e.g., in Figure 3(a), z_A can be any vector of the form $(0, y, -y)$). z_A is then chosen *uniformly randomly* among all valid vectors satisfying (1). Let $S_A = \langle v : v \in B_v \rangle$ denote the linear space spanned by vectors in B_v. One can easily show that:

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

From the above lemma, node B (resp. C) simply needs to compute the inner product of its own innovative vectors with z_A. In Fig. 3(a), suppose that z_A is chosen as $(0, 1, -1)$. Since $(0, 1, -1)\cdot(1, 2, 3) = -1 \neq 0$, node B must contain innovative packet for A. B can broadcast its innovative packet and once A receives it, A will be able to decode the entire batch.

A. Problems of the NSB Coded Feedback for Unicast OR

The goal of using coded feedback in the context of unreliable multicast is different from in the context of reliable unicast OR. In the former, coded feedback is used by a node to inform neighbors that *they have to send more packets*. In contrast, in the latter, we want to use coded feedback so that nodes can inform their upstream nodes that *they should not send any more packets*. This fundamental difference causes two major problems when trying to apply the above NSB coded feedback scheme to reliable unicast OR protocols, like MORE.

Problem 1: The collective space problem. Take Figure 1 for example. Nodes B and C would like to convey their space information to A so that A can stop packet transmission. 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 (see Figure 3(b)). The idea is that, upon the reception of z_B and z_C, A will know that the knowledge spaces of B and C have *collectively* covered the local knowledge space of A and thus will stop transmission.

Nonetheless, when A checks the inner product of the coded feedback and its own innovative packets, we have

$$z_B \cdot (3, 1, 2) = 2 \neq 0 \text{ and } z_C \cdot (3, 1, 2) = -3 \neq 0.$$

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 already have enough information*. This misjudgment is caused by that the NSB coded feedback does not convey the collective space of all downstream nodes but only the space relationship between the individual pairs (e.g., A vs. B and A vs. C). Therefore, if we apply the NSB coded feedback as in [20] to unicast OR, A 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.

Problem 2: Non-negligible false-positive probability. Take Figure 3(c) for example. A wants to send two packets to B and a network coded packet has been received by B already. To convey its local knowledge space back to A, 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 chooses $z_B = (4, -8, 4)$ and A receives such z_B. *Since z_B is orthogonal to all the innovative vectors of A, A will wrongfully conclude that the knowledge space of B covers the local knowledge space of A. A thus attempts no further transmission.* Although Lemma 1 guarantees that this false positive event happens only with probability $\frac{1}{2^8}$, its impact to the system performance is significant. The reason is that in a multi-hop transmission, 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 independent packets for decoding. Although one can fix this false-positive issue by retransmitting another z_B vector, the necessary timer management for the unreliable feedback channel and the additional interference caused by retransmission easily negate the benefits of sending coded feedback.

III. CCACK DESIGN

In this section, we present the design of CCACK. We begin with an overview of the protocol and then we describe its two main components: construction of a novel cumulative coded feedback scheme which addresses the two problems we discussed in Section II-A, and a rate control algorithm, built upon this coded feedback scheme.

A. CCACK Overview

The source and the intermediate FNs in CCACK use intra-flow 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, same as in [9, 10, 15].

Nodes in CCACK maintain per flow state, which includes: a packet buffer B_v, two coding vector buffers B_u and B_w, the *current batch* of the flow, and a *credit counter* (Section III-D). With the exception of B_u and B_w, all the other information is also maintained in MORE. Similar to MORE, this information is soft-state and it is flushed if no packet for a flow is received for 5 minutes.

The source and the FNs broadcast randomly mixed packets and store the coding vectors of these packets in B_w. Whenever a node overhears a packet, the node first checks whether the packet is innovative by comparing the coding vector to those of the existing packets in B_v. If innovative, the newly received packet is stored in B_v. Regardless being innovative or not, the node also checks whether the newly received packet is from an upstream node. If yes, then it stores the forward coding vector in B_u.

Similar to [20], nodes in CCACK embed an additional *ACK vector* in the header of each coded data packet of the forward traffic to report a subset of the packets (or coding vectors) they have received in the past from their upstream nodes. The construction of the ACK vector is described in Section III-C. 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.

Each forward coding vector in B_u and B_w can be marked as H(heard) or $\neg\mathsf{H}$(not heard). A coding vector is marked as $\neg\mathsf{H}$ when 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 mark vectors as H, using the inner product of these vectors and the ACK coding vectors they receive from downstream nodes, as explained in Section III-C.

The destination periodically broadcasts coded feedback to its upstream nodes in the form of ACK vectors (without any payload). This is necessary to inform its upstream nodes whether they should temporarily stop transmitting, 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.[3]

B. Solving the collective-space problem

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_u*, and not only the innovative vectors stored in B_v. Also, when an upstream 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 $B_u \cup B_w$, instead of B_v, have been heard by the downstream node.

[3]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.

Nodes keep checking the rank of the B_u and B_w vectors marked as H. When this rank becomes equal to the rank of innovative packets in B_v for a node A, A stops transmitting either temporarily, until it receives another innovative packet, or permanently if the rank of the B_v vectors is already equal to N. In both cases, the downstream nodes have received a sufficient number of packets that cover the innovative packets of A from the knowledge space perspective.

Focusing on B_u and B_w vectors instead of B_v, this new structure solves the collective-space problem of the NSB coded feedback. Continue our example in Figure 3(b). For node A, B_u contains the received coding vectors $(1, 2, 3)$ and $(3, 1, 2)$ while B_w contains the transmitted vector $(4, 3, 5)$. Suppose we reuse the NSB coded feedback for nodes B and C. Then by checking inner products with z_B and z_C, A knows that the $(1, 2, 3) \in B_u$ and $(4, 3, 5) \in B_w$ have been received. Since the rank of $(1, 2, 3)$ and $(4, 3, 5)$ is the same as the rank of B_v vectors, A stops transmission.

C. 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 $\left(\frac{1}{2^8}\right)^M$ for any integer $M \geq 1$.

Each node maintains M different $N \times N$ *hash matrices* H_1 to H_M where $N = 32$ is the batch size and each entry of the matrix is randomly chosen from $GF(2^8)$. All nodes in the network are aware of the H_1 to H_M matrices of the other nodes. This is achievable by using the ID of a node as a seed to generate the H_1 to H_M matrices. We assume that all vectors are row vectors and we use the transpose u^{T} to represent a column vector.

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

Nodes construct the ACK vectors using the following algorithm:

§ CONSTRUCT THE ACK VECTOR

1: Start from a $0 \times N$ matrix Δ.
2: **while** The number of rows of $\Delta \leq N - 1 - M$ **do**
3: Choose the u with the smallest $usage\_count$ from B_u. If more than one such u exist, choose one randomly.
4: **for** $j = 1$ to M **do**
5: **if** the $1 \times N$ row vector uH_j is linearly independent to the row space of Δ **then**
6: Add uH_j to Δ.
7: Perform row-based Gaussian elimination to keep Δ in a row-echelon form.[4]
8: **end if**
9: **end for**
10: Increment the $usage\_count$ of u by 1.
11: **end while**
12: Choose randomly the coding coefficients c_1 to c_N such that the following matrix equation is satisfied:

$$\Delta(c_1, \cdots, c_N)^{\mathrm{T}} = (0, \cdots, 0)^{\mathrm{T}}.$$

Remark 1: We also require that the randomly chosen coefficients c_1 to c_N are not all zero.
Remark 2: By Line 3 there will be at least 1 degree of freedom when solving the above equations. Since Δ is in the row-echelon form, it is easy to choose c_1 to c_N.
13: Use the vector $(c_1, \cdots, c_N)$ as the ACK vector.

[4]Since Δ is always of row-echelon form, it is easy to check whether the new vector is linearly independent to the row space of Δ.

When a node A overhears a packet with an ACK vector z from a downstream node, it uses again the inner product to check all its vectors in B_u and B_w and determine whether any of them has been heard by the downstream node. More explicitly, a vector $u \in B_u$ (or B_w) is marked H if and only if u passes *all the following M different "H_tests"* (one for each H_j):

$$\forall j = 1, \cdots, M, \quad uH_j z^{\mathrm{T}} = 0, \tag{2}$$

where H_1 to H_M are the hash matrices of the downstream node of interest.

Remark: In our practical implementation, instead of choosing completely random hash matrices H_1 to H_M (each with N^2 random elements), we simply choose H_1 to H_M as "random diagonal matrices", with the N diagonal elements for each H_j randomly chosen from 1 to 255 (excluding zero) and all other elements being zero. This simplification improves the efficiency as the matrix multiplication uH_j can be performed in linear instead of N^2 time.

We now quantify the false positive probability (passing all M tests simultaneously) with this new coded feedback scheme.

Proposition 1: Consider an upstream/downstream node pair A_U and A_D, and A_U receives an ACK vector z_0 from A_D. The hash matrices H_1 to H_M of node A_D are chosen uniformly randomly. For any w vector in $B_u \cup B_w$ of the upstream node A_U, if such w is in the space of the u vectors selected by the downstream node A_D, then it is guaranteed that such w vector will pass all M tests in (2). If such w is not in the space of the selected u vectors, then the false-positive probability (passing all M tests) is $\left(\frac{1}{2^8}\right)^M$.

Sketch of the proof: Let S_B denote the linear space spanned from the u vectors selected by A_D. It is straightforward to show that any $w \in S_B$ must have $wH_j z_0^{\mathrm{T}} = 0$. A more interesting case is when $w \notin S_B$.

Conditioning on the non-zero z_0 vector, for any j the $H_j z_0^{\mathrm{T}}$ vector must be randomly distributed over the null space of S_B vectors since H_j is chosen uniformly randomly. Therefore, the M vectors $H_1 z_0^{\mathrm{T}}$ to $H_M z_0^{\mathrm{T}}$ are equivalent to sending randomly chosen NSB coded feedback for M times. By Lemma 1, the overall false positive probability becomes $\left(\frac{1}{2^8}\right)^M$. A detailed proof can be found in [21].

In our implementation, we used $M = 4$, which gives a false positive probability of 2.33×10^{-10}.[5]

D. 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"[6] as:

$$\Delta Q^f = \dim(B_v^f) - \dim(B_H^f) \tag{3}$$

where B_H^f is the set of vectors in $B_u^f \cup B_w^f$ marked as H, and $\dim(S)$ denotes the number of linearly independent packets in the set S. Note that $\dim(B_H^f) \leq \dim(B_v^f)$. 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-B, when $\Delta Q^f = 0$, i.e., $\dim(B_v^f) = \dim(B_H^f)$, 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 ΔQ_{rel}^f for each flow f as:

$$\Delta Q_{rel}^f = \frac{\Delta Q^f}{\Delta Q^f + \Delta Q_N} \tag{4}$$

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

$$\Delta Q_N = 0.5 \times \Delta Q_N + 0.5 \times \Delta Q_{n_j}^{tot} \tag{5}$$

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 $\alpha \times \Delta Q_{rel}^f + \beta$. 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 $\alpha \times \Delta Q_{rel}^f + \beta$ 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 $\Delta Q_{rel}^f = 1$, then $\alpha \times \Delta Q_{rel}^f + \beta = 1$ and the credit counter will always remain equal to 1, effectively allowing the node to always transmit.

[5] A naive way of achieving the same level of false-positive probability is to use a larger finite field size $\mathrm{GF}(2^{32})$. In such a case, a table look-up method for GF multiplications has to have $2^{32} \times 2^{32}$ 4-byte entries, which takes prohibitively 4 million terabytes to store.

[6] Our solution is inspired by the theoretical backpressure based rate control algorithms [22]. 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.

IV. Evaluation

A. Methodology

We evaluated the performance of CCACK and compared it against MORE using extensive simulations. We used the Glomosim simulator [23], a widely used wireless network simulator with a detailed physical signal propagation model.

We simulated a network of 50 static nodes placed randomly in a $1000m \times 1000m$ area. The average radio propagation range was 250m, the average sensing range was 460m, and the channel capacity was 2Mbps. The *TwoRay* propagation model was used and combined with the Rayleigh fading model to make the simulations realistic. Because of fading, transmission and sensing range are not fixed but vary significantly around their average values.

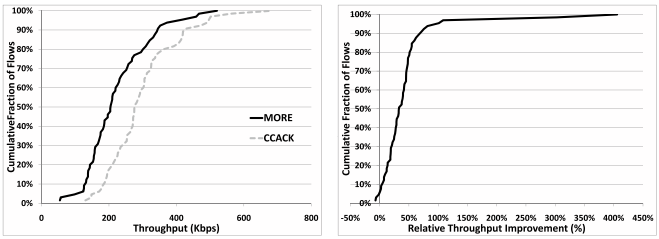
We simulated each protocol in 9 different randomly generated topologies, i.e., placement of the 50 nodes. We varied the number of concurrent flows from 1 up to 4. For a given number of flows, we repeated the simulation 10 times for each topology, selecting randomly each time a different set of source-destination pairs, i.e., we had a total of 90 different scenarios for a given number of flows. In each scenario, every source sent a 12MB file, consisting of 1500-byte packets.

Following the methodology in [9, 8], we implemented an ETX measurement module in Glomosim which was run for 10 minutes 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 2 flows, and 69 scenarios with 3 and 4 flows.

B. Single flow

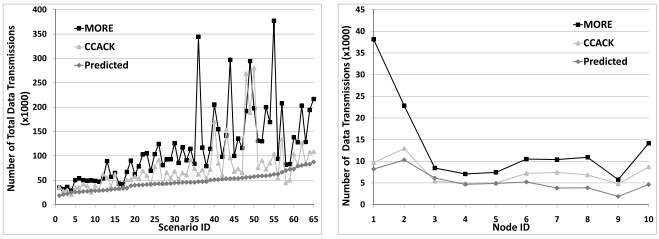
We begin our evaluation with a single flow. Figure 4(a) plots the Cumulative Distribution Function (CDF) of the throughputs of the 65 flows with MORE and CCACK. We observe that CCACK outperforms MORE; the median throughput with CCACK and MORE is 276Kbps and 205Kbps, respectively.



(a) CDF of throughputs achieved with MORE and CCACK.

(b) CDF of relative throughput improvement of CCACK over MORE.

Fig. 4. Throughput comparison between CCACK and MORE – single flow.



(a) Total number of data transmissions per scenario.

(b) Total number of data transmissions per node for one scenario.

Fig. 5. 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.

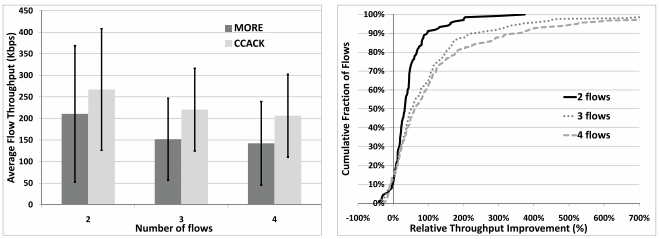
Figure 4(b) plots the CDF of the relative throughput improvement of CCACK over MORE for all 65 flows, defined as $\frac{T^f_{CCACK} - T^f_{MORE}}{T^f_{MORE}} \times 100\%$, where T^f_{CCACK}, T^f_{MORE} are the throughput of flow f with CCACK and MORE, respectively. We observe that CCACK achieves a higher 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 7-9 hops away from the source, the throughput with CCACK is 2-5x higher than with MORE.

Where does the gain for CCACK come from? Figure 5(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 6-7x 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. 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 towards the source.

In contrast, the number of data transmissions with CCACK is much lower than with MORE in all but 2 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-D.

Figure 5(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. 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.



(a) Average per-flow throughputs (bars) and standard deviations (lines).

(b) CDF of relative throughput improvement of CCACK over MORE.

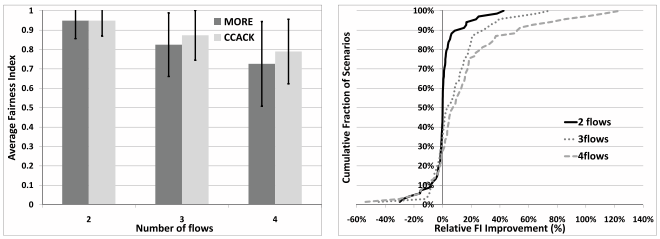
Fig. 6. Throughput comparison between CCACK and MORE – multiple flows.

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) [24]. 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 Figures 6(a), 6(b) compare throughput with CCACK and MORE with 2, 3, and 4 flows. Figure 6(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 2-flow case, and by 45% on average in the 3-flow and 4-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 ETX values. 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 with CCACK which can be used by the nodes or their neighbors for transmitting packets for other flows.

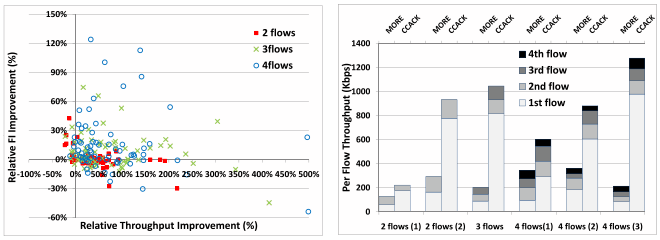
Figure 6(b) plots the CDF of per-flow relative throughput improvement with CCACK over MORE, as defined in Section IV-B, with 2, 3, and 4 flows. CCACK improves per-flow throughputs for more than 85% of the flows in all 3 cases (with 2, 3, and 4 flows). The median improvement is 33%, 55%, and 62%, respectively, with 2, 3, and 4 concurrent flows. Similar to the single flow experiments, some starving flows with MORE



(a) Average per scenario FIs (bars) and standard deviations (lines).

(b) CDF of relative FI improvement of CCACK over MORE.

Fig. 7. Fairness comparison between CCACK and MORE – multiple flows.



(a) Scatterplot of relative throughput improvement vs. relative FI improvement with 2, 3, and 4 flows.

(b) Per-flow throughputs with MORE and CCACK for the 6 scenarios with the largest FI decrease under CCACK.

Fig. 8. Investigating the relationship between throughput and fairness.

show a several-fold improvement with CCACK, up to 3.7x, 8.1x, and 20.4x, in the 2-, 3-, and 4-flow cases, respectively.[7]

Fairness Comparison Figures 7(a), 7(b) compare fairness with CCACK and MORE in case of 2, 3, and 4 concurrent flows. Figure 7(a) plots the average FI with the two protocols. We observe that the average FI is the same with the two protocols in the 2-flow case, but is higher with CCACK in the 3-flow, and 4-flow case by 5.8% and 8.8%, respectively.

Figure 7(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 2, 3, and 4 flows. We observe that CCACK improves fairness in more scenarios as the number of flows in the network increases – in 40% of the 2-flow scenarios, 65% of the 3-flow scenarios, and 72% of the 4-flow scenarios. Similar to the throughput results, the improvement is very large for some scenarios: up to 74% with 3 flows, and up to 124% with 4 flows. This shows again that CCACK improves throughput for some challenged flows, which completely starve with MORE.

Throughput vs. Fairness We now investigate more closely the relationship between throughput and fairness. Figure 8(a) shows the scatterplots of the relative total throughput improvement per-scenario vs. the relative FI improvement per-scenario, in the 2-, 3-, and 4-flow experiments.

We observe that CCACK improves at least one of the two metrics in all but two scenarios (two points in the 3rd quadrant of Figure 8(a)). There are a few points in the 2nd 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 2-flow case are gathered in the 1st and 4th 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 the 1st quadrant for the 3-flow and 4-flow cases. This shows that as the number of flows increases, CCACK improves both throughput and fairness in most scenarios.

We now focus on a few points in the 4th quadrant in Figure 8(a), corresponding to scenarios where FI is reduced by more than 20% with CCACK. There are two 2-flow, one 3-flow, and three 4-flow scenarios (points). Note that all 6 scenarios exhibit large throughput improvements. One may wonder if these improvements are achieved at the cost of compromising the fairness, i.e., throughput of one flow increases significantly, starving the remaining flows.

Figure 8(b) plots the individual per-flow throughputs with MORE and CCACK for these 6 scenarios. We observe that in all but 2 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 11x (from 85Kbps to 978Kbps), but also improves throughputs of the other 3 flows by 183%, 108%, and 111%.

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 11584 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 (48000 multiplications). If we use instead the average packet encoding cost at FNs (27240 multiplications), the total cost of transmitting a packet in CCACK is only 38824 multiplications, i.e., lower than MORE's encoding cost at the source.[8]

[7]The heavy tails of the 3-flow and 4-flow curves are not shown in Figure 6(b) for better clarity.

[8]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.

TABLE I
CODING OVERHEAD IN CCACK IN TERMS OF 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 $B_u \cup B_w$	292	169
Total	1046	416

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 H_tests (if the received packet comes from downstream) and maintaining the rank of the H pkts in $B_u \cup B_w$ (if a received packet from downstream passes all M 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_v.

In [9], the authors found that the bottleneck operation in MORE (packet encoding at the source) takes on average $270\mu s$ on a low-end Celeron 800MHz, limiting the maximum achievable throughput with MORE to 44Mbps with a 1500 byte packet. In CCACK, the cost of the bottleneck operation is 24% higher, so we can expect a maximum achievable throughput of 35Mbps. Note that this value is still higher than the effective bitrate of current 802.11a/g WMNs [25].

Memory overhead. Same as in MORE, routers in CCACK maintain an innovative packet buffer B_v for each flow, and also a 64KB look up table for reducing the cost of the GF(2^8) multiplications [9]. With a packet size of 1500 bytes, the size of B_v is 48KB. The extra overhead in CCACK comes from the two additional buffers B_u and B_w, which store, however, only 32-byte vectors, and not whole packets. In our implementation, the total size of B_u and B_v is $2 \times 5 \times 32 \times 32 = 10KB$, which is relatively small compared to the size of MORE's structures.

Header overhead. The N-byte ACK vector and the total differential backlog $\Delta Q_{n_j}^{tot}$ are the two fields we add to the MORE header. The differential backlog per flow is bounded by the batch size N. With $N = 32$, two bytes are enough to support up to 2048 flows, and the total size of the two fields is equal to 34 bytes. 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. CONCLUSION

In this paper, we presented CCACK, a new efficient NC-based OR protocol. CCACK exploits a novel Cumulative Coded Acknowledgment scheme that allows nodes to acknowledge network coded traffic to their upstream nodes in a simple and efficient way, oblivious to loss rates, and with practically zero overhead. The cumulative coded acknowledgment scheme in CCACK also enables an efficient credit-based, rate control algorithm. Our evaluation shows that CCACK significantly improves throughput by up to 20x and fairness by up to 124%, compared to MORE. The coding, memory, and header overhead of CCACK are comparable to those of MORE, making it easily deployable in WMNs equipped with routers with network coding capabilities.

ACKNOWLEDGMENT

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

REFERENCES

[1] "MIT Roofnet," http://www.pdos.lcs.mit.edu/roofnet.
[2] "Bay area wireless users group," http://www.bawug.org.
[3] "Seattle wireless," http://www.seattlewireless.net.
[4] D. Aguayo, et al., "Link-level measurements from an 802.11b mesh network," in *Proc. of ACM SIGCOMM*, August 2004.
[5] D. B. Johnson and D. A. Maltz, *Dynamic Source Routing in Ad Hoc Wireless Networks*. Kluwer Academic, 1996.
[6] C. E. Perkins and E. M. Royer, "Ad hoc on-demand distance vector routing," in *Proc. of IEEE WMCSA*, February 1999.
[7] J. Bicket, et al., "Architecture and evaluation of an unplanned 802.11b mesh network," in *Proc. of ACM MobiCom*, 2005.
[8] S. Biswas and R. Morris, "ExOR: Opportunistic multi-hop routing for wireless networks," in *Proc of ACM SIGCOMM*, 2005.
[9] S. Chachulski, et al., "Trading structure for randomness in wireless opportunistic routing," in *Proc of ACM SIGCOMM*, 2007.
[10] C. Gkantsidis, et al., "Multipath code casting for wireless mesh networks," in *Proc. of ACM CoNEXT*, 2007.
[11] B. Radunovic, et al., "An optimization framework for opportunistic multipath routing in wireless mesh networks," in *Proc. of IEEE INFOCOM Minisymposium*, 2008.
[12] D. S. J. De Couto, et al., "A high-throughput path metric for multi-hop wireless routing," in *Proc. of ACM MobiCom*, 2003.
[13] X. Zhang and B. Li, "Optimized multipath network coding in lossy wireless networks," in *Proc. of IEEE ICDCS*, 2008.
[14] ——, "Dice: a game theoretic framework for wireless multipath network coding," in *Proc. of ACM MobiHoc*, 2008.
[15] Y. Lin, et al., "CodeOR: Opportunistic routing in wireless mesh networks with segmented network coding," in *Proc. of IEEE ICNP*, 2008.
[16] J. Camp, et al., "A measurement study of multiplicative overhead effects in wireless networks," in *Proc. of IEEE INFOCOM*, 2008.
[17] S. M. Das, et al., "Studying Wireless Routing Link Dynamics," in *Proc. of ACM SIGCOMM/USENIX IMC*, 2007.
[18] Y. Li, et al., "Effects of interference on throughput of wireless mesh networks: Pathologies and a preliminary solution," in *Proc. of ACM HotNets-VI*, 2007.
[19] R. Draves, et al., "Routing in multi-radio, multi-hop wireless mesh networks," in *Proc. of ACM MobiCom*, September 2004.
[20] J. Sang Park, et al., "Codecast: a network-coding-based ad hoc multicast protocol," *IEEE Wireless Communications*, vol. 13, no. 5, 2006.
[21] D. Koutsonikolas, et al., "CCACK: Efficient Network Coding Based Opportunistic Routing Through Cumulative Coded Acknowledgments," TR-ECE-09-13, Purdue University, December 2009.
[22] L. Tassioulas and A. Ephremides, "Stability properties of constrained queing systems and scheduling for maximum throughput in multihop radio networks," *IEEE Transactions on Automatic Control*, vol. 37, no. 12, 1992.
[23] X. Zeng, et al., "Glomosim: A library for parallel simulation of large-scale wireless networks," in *Proc. of PADS Workshop*, May 1998.
[24] R. K. Jain, et al., "A quantitative measure of fairness and discrimination for resource allocation in shared computer systems," DEC-TR-301, Digital Equipment Corporation, Tech. Rep., September 1984.
[25] A. Kamerman and G. Aben, "Net throughput with IEEE 802.11 wireless LANs," in *Proc. of IEEE WCNC*, 2000.