CodeOR: Opportunistic Routing in Wireless Mesh Networks with Segmented Network Coding

Yunfeng Lin, Baochun Li, Ben Liang
Department of Electrical and Computer Engineering
University of Toronto
{ylin, bli}@eecg.toronto.edu, liang@comm.toronto.edu

***Abstract*—Opportunistic routing significantly increases unicast throughput in wireless mesh networks by effectively utilizing the wireless broadcast medium. With network coding, opportunistic routing can be implemented in a simple and practical way without resorting to a complicated scheduling protocol. Due to constraints of computational complexity, a protocol utilizing network coding needs to perform *segmented network coding*, which partitions the data into multiple segments and encode only packets in the same segment. However, existing designs transmit only one segment at any given time while waiting for its acknowledgment, which degrades performance as the size of the network scales up. In this paper, we propose *CodeOR*, a new protocol that uses network coding in opportunistic routing to improve throughput. By transmitting a window of multiple segments concurrently, it improves the performance of existing work by a factor of two on average (and a factor of four in some cases). *CodeOR* is especially appropriate for real-time multimedia applications through the use of a small segment size to decrease decoding delay, and is able to further increase network throughput with a smaller packet size and a larger window size.**

I. Introduction

Due to variable link qualities and wireless interference, one of the fundamental challenges is to maximize throughput of unicast communication sessions in wireless mesh networks. Gupta and Kumar [1] has shown that it may be infeasible to increase the network throughput in a geographic region by deploying more nodes, due to the effects of wireless interference. A traditional wisdom in designing routing protocols (*e.g.*, [2]) treats a wireless link as a point-to-point link, and utilizes a single path to transmit data packets from the source to the destination. It has essentially neglected the fact that a wireless link with a shared communication channel is a broadcast communication medium in nature.

The concept of *opportunistic routing* [3], [4] is able to substantially improve unicast throughput in wireless mesh networks, by effectively utilizing such a shared wireless broadcast medium. In opportunistic routing protocols, all neighboring nodes that are closer to the destination may overhear a data packet, and may be able to assist forwarding the packet to its destination.

With opportunistic routing, however, a delicate balance of tradeoffs has to be maintained: Since multiple nodes in a wireless broadcast region are able to overhear and obtain identical copies of a packet, one needs to avoid unnecessary forwarding of duplicates by multiple nodes. On the other hand, if too few nodes help forwarding the packet, it degrades to a single forwarding path from the source to the destination. Biswas *et al.* [3], for example, have designed a complex packet scheduling algorithm, with a large number of control messages, just to achieve such a delicate balanced tradeoff.

In Chachulski *et al.* [4], it has been realized that, with the help of *random network coding*, opportunistic routing can be achieved without complex scheduling, while still taking full advantage of the wireless broadcast medium. The fundamental insight is that, with random mixing of coded packets, although multiple receivers overhear the same packet in a wireless broadcast neighborhood, they are able to generate linearly independent coded packets with high probability, by combining the received coded packet with existing packets in their buffers.

However, due to the constraints of computational complexity involved in random linear codes, it is infeasible to apply network coding to a large number of data packets [4], [5]. Chachulski *et al.* [4], for example, divide the data stream to be transmitted into different *segments*, and perform coding operations on packets within the same segment. This is usually referred to as *segmented network coding.* Although segmented network coding attempts to achieve a tradeoff between the benefits of network coding and its complexity, it introduces an open challenge that severely affects throughput: *When shall the source stop transmitting coded packets of one segment and move on to the next one?* Intuitively, if the source stops prematurely, the destination may fail to receive a sufficient number of coded packets to decode the entire segment. If the source stops too late, however, a substantial waste of bandwidth resources ensues, since coded packets that are no longer useful to the destination are still being sent by the source.

In Chachulski *et al.* [4], a simple strategy is used as a stop-gap measure to answer this question: the source simply continues to transmit coded packets belonging to the same segment until an explicit acknowledgment from the destination has been received. It is our belief that such a "stop-and-wait" protocol reflects the latter extreme of the tradeoff, where the source stops its transmission *too late*, leading to wasted wireless bandwidth. In addition, if only a single segment is "in flight" in the network, it may not be sufficient to saturate its delay-bandwidth product, again leading to wasted network capacity. It is well understood in the study of flow control protocols that it is important to accurately estimate and saturate

the delay-bandwidth product of a network.

In this paper, with an objective of addressing this open but important challenge, our original contributions are as follows. *First*, with both mathematical analysis in tractable network models and simulations, we are able to show that the "stop-and-wait" protocol in [4] substantially affects network throughput, especially as the network scales up. *Second*, we present *CodeOR* ("*Coding* in *O*pportunistic *R*outing"), a new protocol to allow the source to transmit a *sliding window* of multiple segments using opportunistic routing, and with segmented network coding. CodeOR introduces the concept of flow control to the design of opportunistic routing protocols: rather than performing flow control on bytes or packets, it is performed on a sliding window of *segments*. In our simulations, CodeOR is able to improve throughput by a factor of two on average, as compared to previous work [4]. To our knowledge, there has been no existing work that addresses the problem of flow control with segmented network coding in opportunistic routing.

CodeOR is especially appropriate for real-time multimedia applications through the use of a small segment size to decrease decoding delay. Since the destination needs to wait to accumulate a sufficient number of coded packets before it is able to decode the segment, a smaller segment size (in terms of the number of coded packets in a segment) is beneficial towards reducing such a delay, which is important to real-time multimedia applications. With CodeOR, one may use a smaller segment size, since the number of segments that are "in flight" in the network adapts to its estimated delay-bandwidth product. In addition, CodeOR is also amenable to the use of smaller packet sizes to reduce the packet loss probability in wireless channels with a particular bit error rate (BER), and as such further improve throughput.

The remainder of this paper is organized as follows. We compare CodeOR with related work in Sec. II. In Sec. III, we describe the network model and briefly review network coding with its existing applications in opportunistic routing. In Sec. IV, we show that the network throughput of existing protocols decreases as the network scales up. In Sec. V, we describe the protocol design in CodeOR. In Sec. VI, we present and analyze the idea of increasing the throughput of CodeOR further by reducing the packet size. We use simulations to demonstrate the effectiveness of CodeOR in Sec. VII. Finally, Sec. VIII concludes the paper.

II. Related Work

With the advent of network coding [6] and random network coding [7] in information theory, it has been shown that network coding is able to help improving unicast throughput in wireless mesh networks. In particular, Katti *et al.* [8] focuses on the application of network coding on inter-session network coding in wireless mesh networks, and has shown that unicast throughout may be substantially increased if unicast paths overlap and coding opportunities exist. Our focus in this paper is on the use of intra-session network coding in wireless mesh networks to improve network throughput, combined with multi-path opportunistic routing.

ExOR [3] introduces opportunistic routing in wireless mesh networks by effectively utilizing the wireless broadcast medium to increase network throughput, as compared to traditional single-path routing protocols (*e.g.*, [2]), which neglected the wireless broadcast advantage. In order to realize the benefit of opportunistic routing, ExOR introduces a special complex packet scheduling algorithm. With random network coding, however, Lun *et al.* [9] have noted that the wireless broadcast medium can be utilized optimally if no interference is considered. Inspired by this work, Chachulski *et al.* [4] have proposed a more practical opportunistic routing protocol based on random network coding, referred to as *MORE*, which is feasible to be implemented in practice, and achieves higher throughput than ExOR. However, MORE uses a "stop-and-wait" protocol with a single segment in its sending window, which is not efficient utilizing the delay-bandwidth product in large-scale networks.

With the observation that error probabilities of symbols are much lower than that of packets on a wireless link, MIXIT [10] improves the performance of MORE by operating network coding on symbols rather than on packets as in MORE, where a packet consists of multiple symbols and a symbol is the smallest transmission unit over the wireless link. With such a simple modification, MIXIT can utilize correct symbols in a corrupted packet, and therefore attains higher throughput than MORE. However, as an extension to MORE, MIXIT adopts the same "stop-and-wait" paradigm, and hence shares the same drawback as MORE in large-scale networks.

A number of recent papers [11]–[13] have used different variants of optimization frameworks to extend MORE to scenarios involving multiple sessions. They require the transmission of a large number of control messages in a timely and reliable fashion, which may not be practical in lossy wireless networks. The heuristic proposed in Gkantsidis *et al.* [14] has also attempted to extend to multiple sessions, but suffers from the drawback that the state of a node grows exponentially with the number of its neighbors. More importantly, none of these papers has raised the question on the constraints of the stop-and-wait paradigm on session throughput. The objective of this paper is to address this problem with minimal modifications to maintain the simplicity and practicality of using network coding in opportunistic routing.

III. Network Model and Preliminaries

We first present the network model used in this paper, briefly review the concept of random network coding, and introduce its application to wireless opportunistic routing.

In this paper, we consider a single unicast communication session in a wireless mesh network, where the source has a stream of data to be transmitted to the destination. We model the wireless network as a directed hypergraph (V, E), where V is the sets of nodes and E is the sets of links. A wireless broadcast link is modeled as a hyper link $(i, J) \in E$, where J is a subset of V. The packet loss events on multiple receivers

of a wireless broadcast link are assumed to be independent, as supported by a measurement study performed by Miu *et al.* [15].

Between a pair of connected nodes, node a and b, a coded packet x in random network coding is a linear combination of K source packets $E_1, \ldots, E_K$ in a *segment* with the form $x = \sum_{i=1}^{K} \alpha_i E_i$, where α_i are coding coefficients chosen from a Galois field. The stream of data to be transmitted from the source is divided into multiple segments, each with a predetermined number of packets. When an arbitrary intermediate node a between the source and the destination wishes to transmit coded packets within a segment, a produces a coded packet x_a by encoding all coded packets in its buffer belonging to the segment, namely $x_1, \ldots, x_m$, where m is the total number of coded packets in the buffer that belong to the segment:

$$x_a = \sum_{i=1}^{m} \beta_i x_i, \tag{1}$$

where all multiplication and addition operations are defined on a Galois Field (such as GF(2^8) when the operations are performed on each byte), and β_i is *randomly* chosen from the field. It is easy to see that x_a is also a linear combination of the K original packets from the source, and the coefficients can be derived. Node a then broadcasts x_a along with its coding coefficients over the original packets to all its neighbors.

Suppose node b, one of the neighbors of node a, successfully receives the coded packet x_a, it first check whether x_a is linearly independent with all the buffered coded packets within the same segment. If so, node b inserts x_a into its buffer. Otherwise, x_a is discarded. If node b is the destination, it recovers all K source packets in one segment by the following algorithm. Because the coding coefficients and the coded packet are known, each coded packet represents a linear equation with the K source packets as unknown variables. Decoding the K source packets is equivalent to solving a linear system composed of all coded packets received so far. The *decoding matrix* represents the coefficient matrix of such a linear system. When the rank of the decoding matrix is K, the linear system can be solved and the K source packets are decoded. Otherwise, there exists linear dependence among coded packets, and the destination continues to receive coded packets from its neighbors until all K source packets are decoded. Gaussian elimination is usually used to solve the linear system, with complexity of $O(K^3)$.

In MORE [4], network coding is used with opportunistic routing in the following fashion. The source divides the data stream to be transmitted to multiple segments and transmits them sequentially and independently. For each segment, all neighboring nodes are first ordered according to their shortest distances to the destination in the metric of *ETX* [16], the expected number of transmissions for a packet, with link quality taken into account. A node is chosen as a *forwarding node* only if it is closer to the destination than the previous sending node. Then, a low-complexity algorithm has been proposed to compute the transmission rates for each forwarding node, with the goal of minimizing the amount of redundant transmissions and reduce wireless interference. All nodes produce and broadcast coded packets, using random network coding as we previously described.

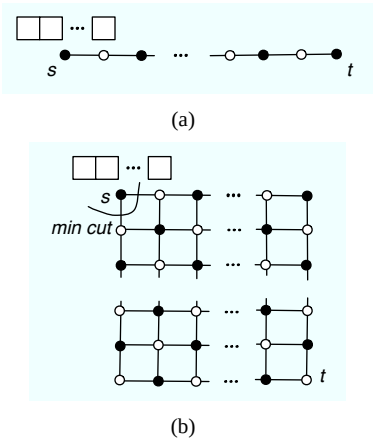

Fig. 1. Transmitting K packets from the source to the destination: (a) a line topology; (b) a grid topology.

IV. Throughput Penalty of MORE on Large-Scale Networks

With an analytical study on tractable network topologies, we are able to show that the throughput of MORE degrades if the network size scales up.

In this analysis, we assume that the segment size is K, the number of nodes in the network is N, and the link transmission probability on all wireless links is p. The segment size K and p are independent of the total number of nodes N. For the sake of analytical tractability, our analysis assumes an ideal synchronized network model, where a time slot is used to transmit either one data packet or one ACK packet. We consider two tractable network topologies: the line and the grid topology, shown in Fig. 1. With respect to wireless interference, we assume the one-hop interference model [17], where two nodes do not interfere with each other as long as they are not neighbors. In topologies shown in Fig. 1, black and white nodes transmit packets alternatively to avoid interference. Analysis for the two-hop interference model only differs on the node scheduling to avoid interference, and is omitted due to space constraints.

A. Line Topology

We first discuss the line topology in Fig. 1(a).

Lemma 1: The throughput of a *perfect* protocol in a line topology is

$$T = \frac{p}{2}. \tag{2}$$

Proof: Each link is a binary erasure channel with capacity p since the link transmission probability is p [18]. For any graph, the capacity from a source to a destination is the min-cut of the graph. The min-cut of the line graph is any link given all links are identical. Hence, the capacity of the network is p if there is no interference. With interference, at any time, there are only half of the nodes transmitting data under the

one-hop interference model [17]. Therefore, the throughput of a perfect protocol achieving the network capacity is $p/2$. ■

We assume that the source and destination are two nodes randomly chosen from the line topology. It is easy to see that their distance is $\Theta(N)$. We use $c_1 N$ to denote their distance in hop counts. We then have the following Lemma to characterize the upper bound of the throughput of MORE.

Lemma 2: The throughput of MORE, denoted as T, in a line topology is at most

$$T \leq \frac{Kp}{2K + (2c_1 N - 1)p}. \tag{3}$$

Proof: Because the transmission of all segments are identical, we focus on studying the transmissions of one segment. The time to transmit K packets including the data transmission time t_{data} from the source to the destination and the ACK transmission time t_{ack} from the destination to the source. Clearly, a protocol needs at least $c_1 N$ time slots to transmit the ACK, *i.e.*, $t_{ack} = c_1 N$. On the other hand, given the link capacity is $p/2$ in the proof of Lemma 1, a protocol requires at least $K/(p/2) = 2K/p$ time slots to transmit K packets on the link adjacent to the destination, and at least $c_1 N - 1$ time slots to transmit them from the source to the node before the destination. Hence, the transmission time for the K data packets in a segment is at least $T_{data} = c_1 N - 1 + 2K/p$. Therefore, the protocol throughput is at most $K/(t_{data} + t_{ack}) = Kp/(2K + (2c_1 N - 1)p)$. ■

Given K, p, and c_1 are independent of N, we have the following proposition.

Proposition 1. The asymptotic throughput of MORE is $O(\frac{1}{N})$ on a line topology.

We remark that the network capacity in Lemma 1 is independent of network size N. However, Proposition 1 indicates the throughput of MORE degrades if N becomes very large.

B. Grid Topology

We now study the throughput penalty of MORE in a grid topology, as in Fig. 1(b). Without loss of generality, we assume that the source and the destination are not in the same line, since otherwise, the problem degrades to the case of a line topology in Sec. IV-A.

Lemma 3: The throughput of a *perfect* protocol in a grid topology is

$$T = \frac{2p - p^2}{2}. \tag{4}$$

Proof: Without consideration of interference, the capacity between two pairs of nodes on a wireless broadcast network is the min-cut between them. From Fig. 1(b), it is easy to see that the min-cut can only be the two links at the source s or the two links at the destination t. The min-cut at the source is $1 - (1-p)^2 = 2p - p^2$ [19], whereas the min-cut at the destination is $2p$. Therefore, the min-cut from the source to the destination is $2p - p^2$. Under the one-hop interference model [17], only half of the nodes can be transmitting packets at any time. Hence, the capacity between the source and the destination is $(2p - p^2)/2$. ■

We choose two nodes randomly as the source and the destination on a grid topology. It is well known that their distance is $\Theta(\sqrt{N})$ on average. We use $c_2\sqrt{N}$ to denote the distance between the source and destination.

Lemma 4: The throughput of MORE on a grid topology is at most

$$T \leq \frac{K(2p - p^2)}{2K + (2c_2\sqrt{N} - 1)(2p - p^2)}. \tag{5}$$

Proof: Similar to the proof of Lemma 2, we compute the transmission time of one segment. The ACK packet is transmitted from the destination to the source with $c_2\sqrt{N}$ time slots. Sending the K packets on the min-cut requires $K/((2p - p^2)/2)$ time slots. Furthermore, any of the K data packets requires an additional $c_2\sqrt{N} - 1$ time slots to travel from the source to the destination. Therefore, the throughput is $K/(t_{data} + t_{ack}) = K(2p - p^2)/(2K + (2c_2\sqrt{N} - 1)(2p - p^2))$. ■

From Lemma 4, we immediately have the following result in a grid topology.

Proposition 2. The throughput of MORE is $O(\frac{1}{\sqrt{N}})$ in a grid topology.

Comparing the throughput of MORE in Proposition 2 and the ideal throughput shown in Lemma 3, we conclude that MORE incurs a substantial performance penalty in large-scale networks with grid topologies.

V. CodeOR: Protocol Design

The analysis in Sec. IV offers the important insight that MORE [4] may perform poorly on large-scale simple topologies. In this section, we introduce a new opportunistic protocol, referred to as *CodeOR*, that substantially improves throughput on practical networks with randomly deployed nodes.

A. Baseline Protocol

We first present the baseline protocol assuming a fixed window size.

1) A Simple Motivating Example: CodeOR is inspired by the following observation. When a node on the multiple paths between the source and destination has sufficient data in segment i, it can represent the source to produce coded packets for segment i. Therefore, bandwidth resources allocated to segment i from the source to this node can be released and used instead to transmit segment $i + 1$. For the example in Fig. 2, in CodeOR, if node 1 and 2 have received a sufficient number of coded packets in segment i, node s may start to transmit the next segment after receiving ACKs from node 1 and 2. In MORE, however, s continues to transmit segment i even when node 1 and 2 has obtained all the required coded packets in this segment, until the end-to-end ACK from destination t indicates that segment i is decoded at the destination. Hence, the source in MORE stops transmitting a segment — and moves on to the next segment — much later than CodeOR. Similarly, in CodeOR, node 2 and 3 can notify node 1 to stop transmitting segment i and start to transmit segment $i + 1$ as long as they obtain a sufficient number of coded packets in segment i.

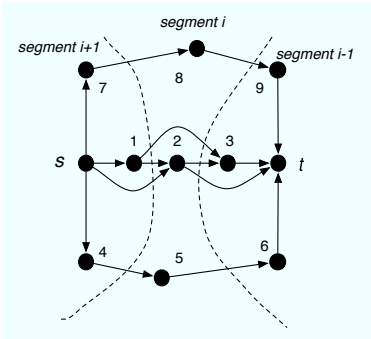

Fig. 2. CodeOR in a general topology, where segment $i-1$, i, and $i+1$ are "in flight" in the network at the same time.

In a nutshell, MORE transmits one segment on the network at any time and waits for the ACK before transmitting the next segment. It is similar to a stop-and-wait protocol on segments. On the other hand, CodeOR allows multiple segments to be "in flight" simultaneously in the network. Therefore, CodeOR is reminiscence of TCP flow control at the segment level. It is intuitive to see that CodeOR outperforms MORE on large-scale networks.

2) Sending Window: Next we describe the motivation and the implementation of a sending window in CodeOR. We first define the upstream and downstream nodes of a node, namely node a. Similar to [4], a subset of neighbors of node a is referred to as its downstream nodes if they forward the data broadcasted from node a, *i.e.*, their shortest-path distances in terms of ETX to the destination are shorter than node a. For instance, in Fig. 2, node 3 and node t are the downstream nodes of node 2. In addition, the subset of neighbors of node a is referred to as its upstream nodes if it is one of their downstream nodes. Fig. 2 shows that node s and node 1 are the upstream nodes of node 2.

We are now ready to motivate the introduction of sending windows. For a forwarding node i in the network, it is common that the aggregated multi-path bandwidth $B_{s,i}$ from the source s to node i, and the bandwidth $B_{i,t}$ from node i to the destination t is not identical. In particular, if $B_{s,i} > B_{i,t}$, and with ACKs to trigger new segment transmissions as soon as possible after node i is able to obtain all required coded packets in an old segment, the buffer of node i may be overwhelmed because it receives data faster from upstream nodes than it is able to transmit data to downstream nodes. In Fig. 2, if we assume that a wireless link between any two nodes has a higher rate[1] if their geographic distance is shorter, the buffer on node 5 may be overflown since the link rate between node 4 and 5 is higher than that between node 5 and 6.

In this paper, we use a *sending window* to limit the number of outstanding segments that the source can transmit at any time in the network. Hence, the maximal amount of data that a forwarding node needs to hold is at most the number of segments in the sending window on the source.

[1]We assume that all nodes use a constant transmission rate. The *rate* in the remainder of the paper refers to the actual data throughput under packet losses.

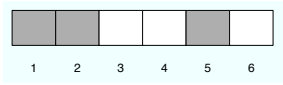

Fig. 3. Gray and white squares represent segments that are decoded or not decoded at the destination, respectively. When segment 5 is decoded, the destination does not transmit an E-ACK, but sends H-ACKs for segment 5 immediately.

With the additional need to implement sending windows, we use two types of ACKs in CodeOR. First, the destination transmits end-to-end ACKs (E-ACK) to the source via the shortest path between them to indicate that a segment of data packets have been received at the destination. Second, a node uses hop-by-hop ACKs (H-ACK) to notify its upstream nodes that a sufficient number of coded packets has been received in a segment, so that the upstream nodes can start to transmit new segments.

Since multiple segments are "in flight" in the network simultaneously, it is possible that segments may be received out of order at the destination. For instance, in Fig. 3, segment 5 is decoded before segment 3 and 4 at the destination. CodeOR handles H-ACKs and E-ACKs differently in such a case. In particular, H-ACKs are sent immediately as their purpose is to stop redundant transmissions of segment 5 immediately. On the other hand, similar to TCP ACKs, E-ACKs are used in flow control and are hence *cumulative* such that they only acknowledge segment i until segment i and all segments j, where $j < i$, have been decoded at the destination. Finally, we point out that the behavior of H-ACKs at the forwarding nodes is identical to H-ACKs at the destination, as they serve the same purpose. On the other hand, E-ACKs are forwarded at the forwarding nodes on the shortest path between the source and the destination.

3) Moving Towards The Next Segment: To simplify the protocol design, a node transmits segments sequentially, *i.e.*, CodeOR ensures that the downstream nodes of a node receive a *sufficient* number of coded packets of segment i before this node dedicates its resource to segment $i+1$, such that this node never needs to transmit segment i again. We then seek to solve the critical challenge in CodeOR: how does a node determines that it has received a *sufficient* number of coded packets from its upstream nodes on a general random topology?

In particular, the downstream nodes of a node receive packets with different rates because they have different distances to the sender. Hence, they complete receiving coded packets in a segment at different times. For example, in Fig. 2, node 1 completes receiving all coded packets in a segment from node s earlier than node 2, 4, and 7, since node 1 enjoys the highest rate from node s. In general, it may not be optimal to request node s to stop transmitting segment i as long as node 1 receives all the packets in the segment, due to the following reason. Under random losses, it may be possible that node 4 and 7 have received no packets, when node 1 has received all the packets in segment i. Therefore, the protocol may degrade to a single-path protocol on path $s \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow t$. However, on the other hand, node s may wait too long for all

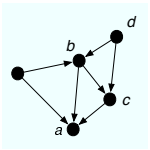

Fig. 4. The local neighborhood of node a.

downstream nodes to complete receiving all the coded packets in segment i. Therefore, node s may start transmitting segment $i+1$ too late and the session throughput degrades.

Hence, to combat random losses, we use a heuristic with the objective that the number of coded packets on each downstream node is in proportion to their receiving rates, referred to as the *receiving threshold* (RT). In the following, we describe the algorithm to compute the receiving threshold RT for a node a that has multiple upstream nodes (illustrated in Fig. 4). Let b be one of upstream nodes of node a and node c be the node with the maximal downloading rate among all downstream nodes of node b. Assuming node c is able to receive R_c linearly independent coded packets from its upstream nodes (we will describe the details to estimate R_c later). We then set the receiving threshold RT_{ab} of node a corresponding to node b to be

$$RT_{ab} = R_c \frac{\sum_{u_a \in U(a)} p_{u_a a}}{\sum_{u_c \in U(c)} p_{u_c c}}, \tag{6}$$

where p_{ij} denotes the transmission probability between node i and j, and $U(x)$ denotes the set of upstream nodes of x.

We set the final receiving threshold RT_a of node a as the maximum of all RT_{ab} corresponding to each upstream node b:

$$RT_a = \max_{b \in U(a)} RT_{ab}. \tag{7}$$

We note that the computation of RTs requires knowledge of the transmission probabilities of wireless links in the local neighborhood. We believe that such a requirement is reasonable and practical, because MORE [4] needs them as well. In practice, R_c can be piggy-backed in the packets generated from node c to all its neighbors.

Next, we describe the algorithm for node c to estimate the maximum number of coded packets R_c it can receive from all its upstream nodes. For most of the cases, a node is able to receive all K linearly independent coded packets from all its upstream nodes. However, because the receiving thresholds (7) may be low on nodes that are far away from all other nodes and have low transmission rates due to packet loss, these nodes and their downstream nodes may not be able to receive all coded packets in a segment. In such a case, node c estimates R_c by detecting whether there is new information from its upstream nodes. Node d, an upstream node of node c, does not have the knowledge of any new information for node c with high probability [7], if more than m coded packets transmitted from node d are linearly dependent, where m is a small positive number. Similarly, node c uses the same algorithm to detect all of its upstream nodes and receives the value of R_c until all of its upstream nodes have no new information.

TABLE I
THE PROCEDURES TO PROCESS INCOMING PACKETS FOR SEGMENT i.

Upon receiving a E-ACK at the source
  obtain decoding information from the E-ACK
  move the sending window to $(i+1, i+W)$
  if transmissions of segment $i+1$ to $i+W-1$ stop
    transmit segment $i+W$
  end if

Upon receiving a E-ACK at an intermediate node
  forward this E-ACK on the shortest path

Upon receiving a H-ACK from node b
  record b in the downstream H-ACK table
  If receive all H-ACKs from downstream nodes
    stop transmitting segment i
    remove segment i
  end if
  transmit segment $i+1$

Upon receiving a data packet P at a forwarding node
  remove the segments older than the decoding information piggybacked in P
  if P is linearly dependent with existing packets
    P is discarded
  end if
  the number of innovative packets r increases by 1
  cache P in buffer
  if $r \geq$ receiving threshold
    send a H-ACK for segment i
  end if

Upon receiving a data packet P at the destination
  if P is linearly dependent with existing packets
    P is discarded
  end if
  the number of innovative packets r increases by 1
  store P in the buffer
  if $r = K$
    send a H-ACK for segment i
    if all segments older than i are decoded
      send an E-ACK for segment i
    end if
  end if

Finally, a node stops transmitting segment i, removing it from node buffer, and starts to transmit segment $i+1$ after collecting all H-ACKs for segment i. We further present a complement way to stop unnecessary transmissions in our protocol. When a source receives an E-ACK, it obtains the *decoding information* that the destination have decoded certain amount of segments such as segment i and the older segments than segment i. The source piggybacks this decoding information on all data packets it produces. A forwarding node then stops and removes all segments with a segment ID smaller than or equal to i when receiving the decoded information. All forwarding nodes also piggyback this information on the data packets they generate. It is easy to see that all nodes in the network obtain this information and stop unnecessary transmissions in a timely fashion. We summarize the protocol to process each incoming packet in Table I.

B. Estimating Proper Window Size

Similar to TCP, the size of the sending window should approximately equals the delay-bandwidth product between the source and the destination. Otherwise, if the window size is too small, the protocol is unable to efficiently utilize bandwidth resources, and its throughput suffers. On the other hand, if the window size is too large, the forwarding nodes need to buffer a large amount of data. Such a conjecture is verified with simulations in Sec. VII-C. Therefore, we need an algorithm to estimate the correct window size. With the analogy of CodeOR with TCP, it is natural to apply the ideas of TCP flow control to the design of CodeOR.

CodeOR operates as an end-to-end rateless code and does not introduce segment losses in the network, since forwarding nodes do not drop packets. Hence, we adopt a similar algorithm to TCP Vegas [20], using increased queueing delay as congestion signals. The major distinction between TCP Vegas and CodeOR is the difference on the units of the sending window size. Specifically, TCP window size is in terms of the number of packets[2], whereas the unit in CodeOR is segments. Therefore, all quantities in TCP Vegas should be converted from packets to segments. With a similar algorithm to TCP Vegas, CodeOR seeks to estimate a window size W — the optimal number of segments to saturate the delay-bandwidth product. It also seeks to maintain a few more segments in the network to prevent window size oscillation by providing some damping effect.

We now describe the details of the window estimation algorithm in CodeOR, which is similar to the congestion avoidance algorithm in TCP Vegas. The slow start stage is similarly adapted and is omitted due to space constraints. We define the *decoding time* (DT) of a segment as the interval between the time that the source transmits the first coded packet of a segment and the source receives its E-ACK. Around once per DT, the algorithm performs the window adjustment algorithm by comparing the actual sending rate to the expected sending rate. First, it computes the expected sending rate E. We use *BaseDT* to denote the decoding time of a segment when the network is not congested. *BaseDT* is set to the smallest DT observed so far. It is usually the decoding time of the first segment, since the window size is initially 1 and the network is not congested initially. Then, we have the expected sending rate E:

$$E = W/BaseDT, \tag{8}$$

where W is the current window size.

Next, we compute the actual sending rate A. We record the number N_i of the segments that have been sent since the source transmits the first coded packet of segment i, where segment i is the latest segment that has been acknowledged by an E-ACK. Let DT_i denote the DT of segment i. We then have

$$A = N_i/DT_i. \tag{9}$$

[2]More precisely, TCP window size is in bytes. However, packets are equivalent to bytes as units if all data packets have the same size.

Let $D = E - A$ be the difference between the expected sending rate and the actual sending rate. We define two thresholds α and β, if $D < \alpha$, the algorithm increases the window size by 1. If $D > \beta$, the algorithm decreases the window size by 1. If $\alpha \leq D \leq \beta$, the algorithm maintains the same window size. The intuition of the algorithm is that if the actual rate A is significantly lower than the expected rate E, the network is congested such that the window size should be decreased. On the other hand, if the actual rate A is too close to the expected rate E, there is a risk that CodeOR does not fully utilize the delay-bandwidth product, hence the window size should be increased. In practice, we recommend $\alpha = 1/BaseDT$ and $\beta = 3/BaseDT$, such that the additional number of segments buffered in forwarding nodes is between 1 and 3, which is the smallest feasible positive numbers.

VI. Effect of Overhead and Packet Size on Network Throughput

In the above analysis, we assume the time to complete delivering a packet consists of only the transmission time. However, in reality extra *overhead time* is required, such as the contention window in IEEE 802.11, software processing overhead, and the time to switch a half-duplex device between sending and receiving modes. In this section, we analyze the effect of transmission overhead and different packet sizes on the throughput of a network with lossy wireless links. Through this analysis, we demonstrate the different effect of packet size on MORE and CodeOR. We show that the throughput of CodeOR can be further improved by using packets with a moderate to small size.

To simplify the analysis, we assume that all bits in a packet has the same probability e to be corrupted independently of each other. Furthermore, a packet is successfully transmitted when all bits are transmitted correctly[3]. Hence, for a packet with size m, its successful transmission probability is $p = (1 - e)^m$. Clearly, p increases if packet size is reduced. However, a reduced packet size accentuates the negative effect of transmission overhead, effectively decreasing the fraction of network resource utilization.

To quantify the above tradeoff, let δ and r denote the overhead time and the wireless link speed (*e.g.*, 11Mbps in IEEE 802.11b) respectively. Then we have link throughput T as the expected amount of delivered data pm divided by the delivery time $m/r + \delta$ for a packet:

$$\begin{aligned} T &= \frac{pm}{m/r + \delta} \\ &= \frac{(1-e)^m mr}{m + r\delta} \ . \end{aligned} \tag{10}$$

Note that (10) can be applied also to network throughput if we consider δ as the overall overhead time along a packet's path. We show a numerical illustration of (10) in Fig. 5. This figure

[3]This is equivalent to using no forward error control coding. It is easy to see that when error control coding is used, the presented analysis remains applicable to illustrate the general trends of network throughput. A simple adjustment to the presented analysis to approximately account for the effect of error control coding is to reduce the value of e.

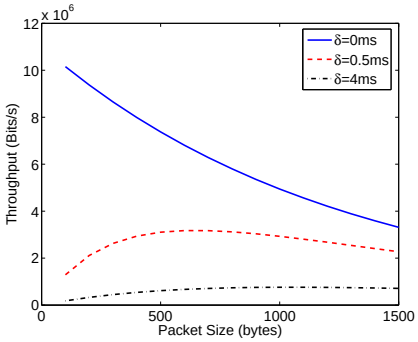

Fig. 5. Throughput under different overhead times, where the bit error probability e is 0.0001, and the link speed r is 11Mbps.

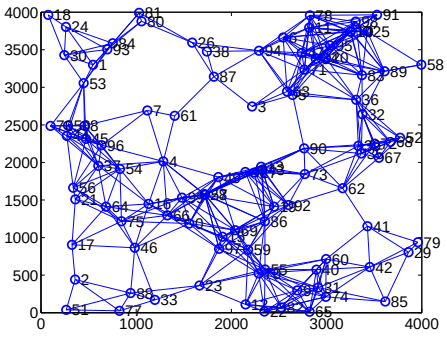

Fig. 6. The random graph with 100 nodes uniformly distributed in a square with size 4000×4000.

quantifies the three different trends that match our intuitive expectations. *First*, if there is no overhead time, reducing the packet size can increase throughput since the probability that a packet is successfully transmitted increases. *Second*, when the overhead time is modest, the throughput increases first and then decreases, as the packet size is reduced. *Third*, when the overhead time is sufficiently large, decreasing the packet size only reduces throughput.

The simple model of (10) provides some insights on the difference between the impact of packet sizes on MORE and CodeOR. If we consider the packet size in (10) as the amount of data transferred in one sending window, the overhead time then refers to any time that is not used to transmit data packets in the sending window. Besides the usual overhead time discussed previously, an additional overhead time in MORE is the ACK transmission time. In contrast, CodeOR does not have such overhead since it transmits new segments concurrently with E-ACKs for old segments. Therefore, the overhead time of MORE is significantly larger than CodeOR. Henceforth, we expect that it is more beneficial to reduce the packet size in CodeOR than in MORE. This conjecture is consistent with the experimental results in Sec. VII-D.

VII. Performance Evaluation

We study the effectiveness and properties of CodeOR through simulation. We have developed a customized discrete event simulator, which implements randomized network coding, wireless opportunistic routing protocols, and basic MAC functions. For the physical layer, we use the measurement-based model from [21] to capture the effect of opportunistic reception in a lossy wireless environment, which empirically maps link distance to the transmission success probability between two wireless nodes. In our simulation, two nodes are regarded as neighbors only if the link quality between them is sufficient to achieve a transmission success probability higher than 0.05.

We conduct experiments on a random topology shown in Fig. 6 with 100 nodes that are deployed, uniformly at random, in a square of size 4000×4000. By default, unless explicitly stated otherwise, we set the data packet size to 1500 bytes and ACK packets to 40 bytes in most experiments. In addition, we set the segment size to 10 for the purpose of illustration. Obviously, a larger ratio between the segment size and the source-destination distance favors the stop-and-go scheme of MORE. In this work, we use a fixed segment size, along with different source-destination distance values, to illustrate the general trends of the throughput gain of CodeOR. Finally, we set the overhead time per link (discussed in Sec. VI) for each packet to be 2 ms, which we obtained by excluding the packet transmission time from the round-trip time measurements from a laptop to its nearest access point. We note that this overhead time cannot be ignored, compared with the transmission time 1.1 ms of a data packet on a wireless link with speed 11Mbps.

We assume that a node occupies the wireless channel in a local neighborhood during the transmission time of a packet. However, the wireless channel is released and can be used by other nearby nodes during the overhead time for this packet. We have implemented a simple MAC protocol, which schedules a random node from all eligible nodes that have data to transmit and do not interfere with other transmitting nodes, provided that the wireless channel becomes idle in a local neighborhood. Similar to MORE, we give higher priority to control messages.

A. Behavior of a Single Flow

We first study the behavior of a single flow in a large-scale network. We set the source and destination to nodes 51 and 91 in Fig. 6, respectively. The window size is static and is set to 5 in CodeOR. The results are shown in Fig. 7. For MORE, we observe that a time gap exists between different segments where the destination does not receive any useful coded packets. This time gap includes the ACK transmission time of MORE, and results from the stop-and-wait paradigm of MORE. On the other hand, because CodeOR can transmit other segments in the network when ACKs are in transmission, the destination is able to obtain useful coded packets almost all the time. Hence, CodeOR increases the network throughput significantly.

B. Throughput Gain of CodeOR

In this section, we compare the throughput of CodeOR and MORE over a large number of flows. We randomly choose 500 source-destination pairs and plot the Cumulative Distribution Function (CDF) of the throughput over these 500 flows in Fig. 8(a). Here, the throughput of a flow is measured

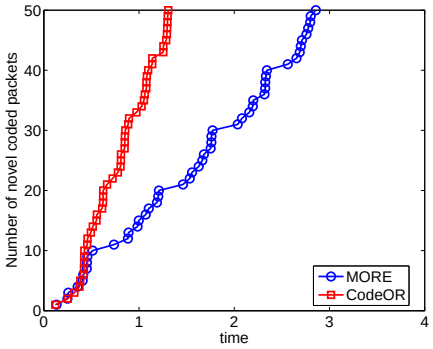

Fig. 7. The number of received innovative coded packets at the destination over time. The throughput gain of CodeOR to MORE is 2.8276 in this flow.

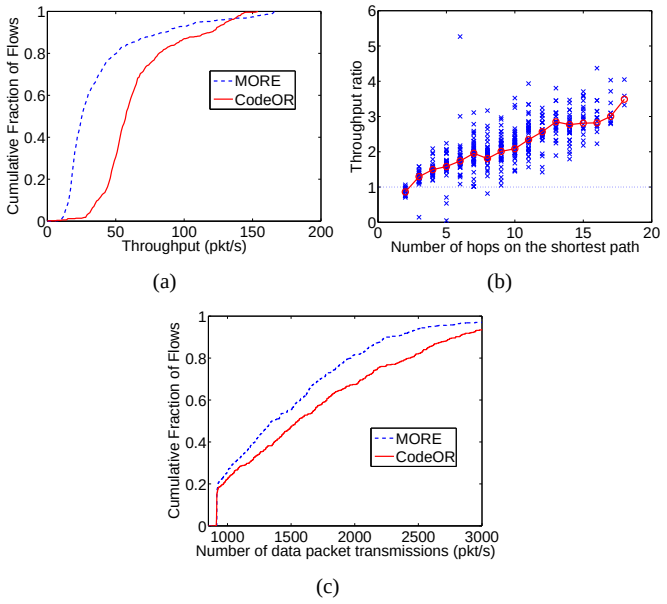

Fig. 8. (a) CDF of throughput. The median throughput gain of CodeOR to MORE is 2.2325. About 80% of the flows achieve throughput above 50 pkt/s in CodeOR, compared with less than 20% of the flows in MORE. (b) Throughput ratio of CodeOR to MORE vs. number of hops on the shortest path. The median of throughput gain is shown in the figure. (c) CDF of the number of data packet transmissions. The median of CodeOR is 1.1435 of MORE.

as the average number of received innovative coded packets per second, over 20 seconds at the destination. The window size again is set to 5. We observe that CodeOR achieves significantly higher throughput than MORE.

To further study the CodeOR performance gain, in Fig. 8(b), we organize all flow throughputs by their *path lengths*, the number of hops on the shortest path between their source and destination. We observe that the throughput ratio between CodeOR and MORE increases nearly linearly with the path length, which is justified by the fact that the delay-bandwidth product of a multi-path increases with the path length, so the stop-and-wait paradigm of MORE becomes increasingly inefficient. In contrast, CodeOR utilizes all network resource as long as the window size is large enough to allow sufficient data to be transmitted into the pipeline of the delay-bandwidth product.

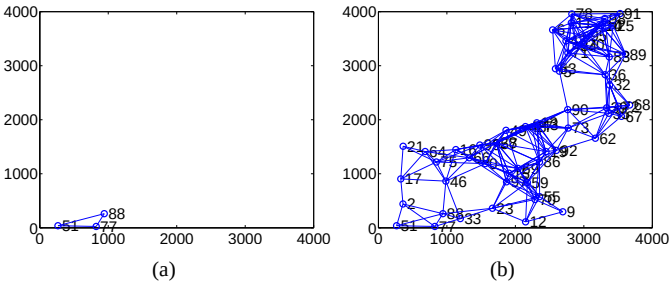

Fig. 9. Forwarding nodes and the multi-path sub-topologies in data transmissions. The source and the destination are (a) nodes 51 and 88 (b) nodes 51 and 91, where node 51 and 91 are in the southwest and northeast corner, respectively.

Finally, Fig. 8(c) compares the number of data transmissions by these two protocols. We observe that CodeOR induces only slightly more data transmissions. Therefore, the throughput gain of CodeOR comes relatively "free" without introducing too many more transmissions.

C. Effect of Window Size

We first study the impact of a static window size on throughput. We consider two typical cases: the short and long paths, where the source and the destination are nodes 51 and 88, and nodes 51 and 91, respectively. The forwarding nodes and the sub-topologies to transmit data are illustrated in Fig. 9 for these two cases. From Fig. 10(a), we observe that for the long path, the network throughput increases significantly until the window size reaches 5 or 6. Hence, CodeOR does not fully utilize all multi-path network resource between the source and destination when the window size is small. For the short path, CodeOR starts to utilize all network resource when the window size reaches 2. Then, the network throughput does not increase further with a window size larger than 2 because the multi-path topology between this pair of source and destination allows only 2 segments in transmission, so that the extra data from a larger sending window are merely buffered in the network.

In Fig. 10(b), we show the buffer usage with different window sizes for the long path. We omit similar results observed for the short path. This figure shows that the buffer usage when the window size is 12 is significantly larger than when the window size is 6. Such observation motivates the integration of an algorithm to detect the proper window size in CodeOR.

We next evaluate the adaptive window estimation algorithm described in Sec. V-B. We trace the dynamics of the window size in Fig. 10(c). We observe that in the slow-start stage, the window size doubles every other segment decoding time (defined in Sec. V-B). Afterwards, CodeOR enters the stage of congestion avoidance, and maintains the same window size or adjusts it only slightly. Overall, CodeOR reaches the stable window size consistent with the value observed in Fig. 10(a), and in a timely fashion. We note that the window sizes are sampled whenever they change and for every segment decoding time. Hence, the short path has more frequent samples than the long path because its segment decoding time is shorter. Finally, we remark that it is not necessary that the

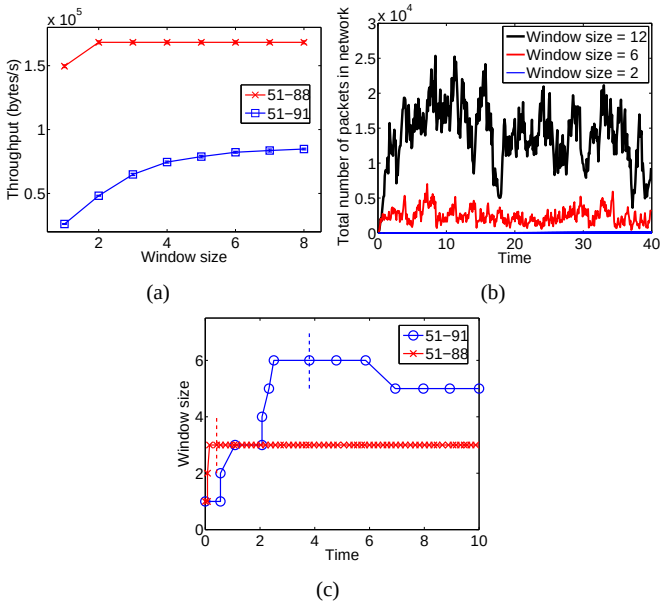

Fig. 10. (a) Throughput under different sending window sizes for two pairs of nodes: node 51 and 88, node 51 and 91. (b) Total number of buffered packets evolves with time under different window sizes for the flow between node 51 and 91. (c) Window size evolves with time, where the vertical line marks that CodeOR transits from the slow start stage to the congestion avoidance stage.

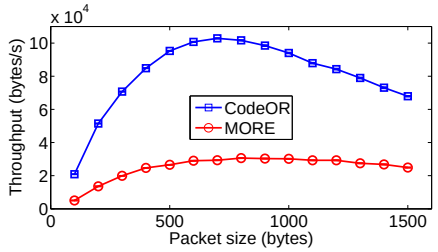

Fig. 11. Throughput under different packet sizes.

estimated window size matches exactly the optimal window size in Fig. 10 due to the noise introduced by the wireless random channel and the extra few segments that the algorithm attempts to maintain in the network.

D. Reducing Data Packet Sizes to Increase Throughput

We finally investigate the impact of reducing the packet size on MORE and CodeOR. We consider again the node pair, 51 and 91, illustrating the experimental results in Fig. 11. We observe the same trends as in the discussion of Sec. VI. In particular, we observe that reducing the packet size mostly does not improve MORE because the network utilization decreases as the packet transmission time decreases so that the end-to-end ACK overhead time dominates. In contrast, these experiments confirm that CodeOR benefits more with moderate to small packet sizes than MORE.

VIII. Conclusion

In this paper, we provide theoretical and practical evidences to show that the throughput of prior opportunistic routing protocols based on network coding degrades in a large-scale network. We then introduce CodeOR to allow the concurrent transmission of multiple segments to fully utilize network resources. We show that CodeOR significantly outperforms existing approaches in network throughput while maintaining a similar amount of data transmissions. Furthermore, we show that unlike existing protocols, CodeOR is able to achieve higher throughput for moderate to small packet sizes. Combining this with a smaller segment size decreases the decoding delay, which makes CodeOR especially appropriate for real-time multimedia applications.

References

[1] P. Gupta and P. R. Kumar, "The Capacity of Wireless Networks," *IEEE Transactions on Information Theory*, vol. 46, no. 2, pp. 388–404, March 2000.

[2] J. Bicket, D. Aguayo, S. Biswas, and R. Morris, "Architecture and Evaluation of an Unplanned 802.11b Mesh Network," in *Proc. of ACM MOBICOM*, 2005.

[3] S. Biswas and R. Morris, "ExOR: Opportunistic Multi-Hop Routing for Wireless Networks," in *Proc. of ACM SIGCOMM*, 2005.

[4] S. Chachulski, M. Jennings, S. Katti, and D. Katabi, "Trading Structure for Randomness in Wireless Opportunistic Routing," in *Proc. of ACM SIGCOMM*, 2007.

[5] M. Wang and B. Li, "How Practical is Network Coding?" in *Proc. 14th Proc. of IEEE International Workshop on Quality of Service (IWQoS)*, 2006.

[6] R. Ahlswede, N. Cai, S. R. Li, and R. W. Yeung, "Network Information Flow," *IEEE Transactions on Information Theory*, vol. 46, no. 4, pp. 1204–1216, July 2000.

[7] T. Ho, R. Koetter, M. Medard, D. R. Karger, and M. Effros, "The Benefits of Coding over Routing in a Randomized Setting," in *Proc. of IEEE International Symposium on Information Theory*, 2003.

[8] S. Katti, H. Rahul, W. Hu, D. Katabi, M. Medard, and J. Crowcroft, "XORs in The Air: Practical Wireless Network Coding," in *Proc. of ACM SIGCOMM*, 2006.

[9] D. S. Lun, M. Medard, and R. Koetter, "Network Coding for Efficient Wireless Unicast," in *Proc. of International Zurich Seminar on Communications (IZS)*, 2006.

[10] S. Katti and D. Katabi, "MIXIT: The Network Meets the Wireless Channel," in *Proc. of the Sixth ACM Workshop on Hot Topics in Networks (HotNets-VI)*, 2007.

[11] B. Radunovic, C. Gkantsidis, P. Key, and P. Rodriguez, "An Optimization Framework for Opportunistic Multipath Routing in Wireless Mesh Networks," in *Proc. of IEEE INFOCOM, Minisymposium*, 2008.

[12] X. Zhang and B. Li, "Optimized Multipath Network Coding in Lossy Wireless Networks," in *Proc. of IEEE ICDCS*, 2008.

[13] ——, "Dice: a Game Theoretic Framework for Wireless Multipath Network Coding," in *Proc. of ACM MobiHoc*, 2008.

[14] C. Gkantsidis, W. Hu, P. Key, B. Radunovic, P. Rodriguez, and S. Gheorghiu, "Multipath Code Casting for Wireless Mesh Networks," in *Proc. of CoNEXT*, 2007.

[15] A. Miu, H. Balakrishnan, and C. E. Koksal, "Improving Loss Resilience with Multi-Radio Diversity in Wireless Networks," in *Proc. of ACM MOBICOM*, 2005.

[16] D. S. J. D. Couto, D. Aguayo, J. Bicket, and R. Morris, "A High-Throughput Path Metric for Multi-Hop Wireless Routing," in *Proc. of ACM MOBICOM*, 2003.

[17] G. Sharma, R. R. Mazumdar, and N. B. Shroff, "On the Complexity of Scheduling in Wireless Networks," in *Proc. of ACM MOBICOM*, 2006.

[18] T. M. Cover and J. A. Thomas, *Elements of Information Theory*, 2nd ed. Wiley, June 2006.

[19] A. F. Dana, R. Gowaikar, R. Palanki, B. Hassibi, and M. Effros, "Capacity of Wireless Erasure Networks," *IEEE Transactions on Information Theory*, vol. 52, no. 3, pp. 789–804, March 2006.

[20] L. S. Brakmo and L. L. Peterson, "TCP Vegas: End to End Congestion Avoidance on a Global Internet," *IEEE Journal on Selected Area in Communications*, vol. 13, no. 8, pp. 1465–1480, October 1995.

[21] J. Camp, J. Robinson, C. Steger, and E. Knightly, "Measurement Driven Deployment of a Two-Tier Urban Mesh Access Network," in *Proc. of ACM International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2006.