Employing Coded Relay in Multi-hop Wireless Networks

Zhenghao Zhang, Wei Hu, and Jin Xie
Computer Science Department
Florida State University
Tallahassee, FL 32306, USA

***Abstract*—In this paper, we propose Coded Relay (Crelay) for multi-hop wireless networks. Crelay exploits both the partial packets and the overhearing capabilities of wireless nodes, and uses an error correction code in packet forwarding. When a node overhears a partial packet from an upstream node, it estimates the number of errors in the packet and informs the upstream node. The upstream node calculates the number of parity bytes required to correct the errors and can usually send only a small amount parity bytes instead of sending the complete packet, hence improving the network efficiency. We propose a novel protocol for packet forwarding with partial packets and overhearing, which forwards individual packets and will work seamlessly with upper layers. We also propose a practical algorithm for finding the packet forwarding path. We implement Crelay within the Click modular router and our experiments show that it can significantly improve the performance of wireless networks.**

I. Introduction

In this paper, we propose Coded Relay, abbreviated as *Crelay*, which is a packet forwarding protocol for multi-hop wireless networks. Crelay is particularly suited for wireless mesh networks where the nodes are mesh routers which are not constrained in power and processing capabilities. Crelay exploits two fundamental properties of transmissions over the wireless medium, namely the partially correct packets and the overhearing opportunities. That is, an erroneous packet often has just a few errors [1], [2]; also, when a node transmits a packet to second node, a third node may overhear this packet because the medium is shared.

The core idea of Crelay is simple and can be explained as follows. Basically, nodes relay *coded* messages to the next hop node depending on the amount of information that has already been overheard, where a coded message is constructed according to an *error correction code*. As a simple example, consider a path $A \rightarrow B \rightarrow C$, where node A wishes to send a packet $\mathbf{P}$ to node C. Node A first transmits $\mathbf{P}$, after which node B gets $\mathbf{P}$ while node C gets a partial packet with some errors. Node C estimates the number of errors using an error estimator such as AMPS [18], and asks node B to send *just enough* parity bytes to correct these errors, instead of sending the entire packet. Thus, fewer bytes are transmitted and a better efficiency is achieved.

Crelay is a novel opportunistic packet forwarding scheme for wireless networks. The core difference between Crelay and existing opportunistic routing protocols such as MIXIT [7], MORE [6] and ExOR [8] is that Crelay forwards individual packets while ExOR [8], MORE [6], and MIXIT [7] forward batches of packets. As a result, Crelay works seamlessly with the upper layers while protocols requiring packet batching are not suitable for interactive traffic such as the TCP traffic [9]. Crelay is related to *cooperative relaying*, which is an active research area in the wireless communication community [3], [4]. Existing research on cooperative relaying focuses on deriving the theoretical channel capacity; we focus on the design and implementation of practical systems.

In this paper, we design a protocol which allows the upstream nodes to obtain the receive status of a packet at the downstream nodes, such that the nodes can make good forwarding decisions, i.e., who should send the packet, how many parity bytes should be sent, etc. The challenge is that the coordination must be achieved in a timely manner at low cost. Our protocol is designed based on the fact that *packets usually experience queuing delays*, especially under high load when the throughput should be optimized. When a node receives a packet, with a non-empty queue, it does not have to take an immediate action for the packet because it has to serve other packets first, which allows extra time for coordination. In Crelay, when a node overhears a new packet, it sends control messages about the packet *piggybacked with other packets that must be sent first*. After several transmissions, nodes are usually able to make informed decisions about the new packet before taking actions for the packet. The control overhead is low because of the piggybacking. We also propose a practical routing algorithm for Crelay. We implement Crelay and our experiments show that it can significantly improve the performance of wireless networks.

The rest of the paper is organized as follows. Section II describes the Crelay protocol. Section III discusses routing in Crelay. Section IV describes our implementation. Section V gives experimental evaluations. Section VI concludes the paper.

II. The Crelay Protocol

A. Overview

Crelay is a link state protocol with which nodes learn and propagate the quality of the links. A link is measured by two values, namely the *erasure ratio* and the *error ratio*, which are the fraction of the lost frames and the fraction of the erroneous bytes in a received frame, respectively. A *path* is an ordered list

of nodes that participate in relaying the packet. Given a path, with respect to a node on the path, all nodes appear earlier on the path are the *upstream nodes* and all nodes appear later are the *downstream nodes*. The path is found by a common routing algorithm based on the link states; every node should install the same path for a source and destination pair. With overhearing, Crelay's path is not a path in the traditional sense and is similar to the concept of forwarder list in [6]. A node is a better forwarder than its upstream nodes.

All Crelay packets are broadcast to enable overhearing. All control messages, including the ACK and the packet receive status, are sent in the same link layer frames as the data packets. Whenever possible, they are piggybacked with the data packets, not necessarily with the packets in the same flow. Crelay packet header includes the source and destination IDs, with which a node can look up its routing table and determine the path for the packet as well as the upstream and downstream relations with other nodes.

A node may keep the overheard packets in its buffer and may announce the receive status about the packets. An upstream node determines the data to be transmitted depending on packet receive status of its downstream nodes. The transmitted data may be the original data or may be the parity bytes. A node combines all overheard data for a packet, which may include the original data and several pieces of parity bytes, to decode a packet. A node sends an ACK if a packet has been decoded. A node may also send an ACK even when it has not decoded the packet, for example, when one of its downstream nodes has decoded the packet.

B. Preliminaries

1) Error Correction Code: Crelay may use any error correction code supporting incremental redundancy. The Reed-Solomon (RS) code is used in our current implementation because of its strong error correction capability and the availability of software implementations [13]. A *codeword* with the RS code is basically the data bytes followed by the parity bytes [10]. If there are e erroneous bytes in the received bytes belonging to a codeword, with *any* additional $2e$ bytes in the codeword, all errors can be corrected.

2) Error Estimator: One crucial aspect of Crelay is to request just enough number of parity bytes to correct the errors. This requires the receiver to have an estimate of the number of errors in a received packet. As the standard checksum test can only detect the existence of errors, an error estimator is needed. Crelay may adopt any error estimator with acceptable performance. Our current implementation adopts the AMPS estimator explained in details in the technical report at [18]. Here, we give a brief explanation of AMPS. With AMPS, the sender randomly samples K data bytes in the packet and computes their parity bit as a *sample*. Multiple samples are calculated in this manner which are transmitted along with the packet. When the packet is received, the receiver calculates local samples based on the received data bytes, which may be different from the received samples because some bytes may have been corrupted. AMPS estimates the number of corrupted bytes based on the number of mismatching samples according to the Maximum A Posteriori (MAP) criteria. AMPS has a very low overhead of 8 bytes for every transmitted frame. We find in our earlier works [17], [18] that AMPS has a good performance, e.g., for more than 78.2% of the times, its estimation error is no more than 40 bytes among no less than 1500 transmitted bytes.

3) Interleaving: Errors in the packets may occur in bursts. This may result in an uneven distribution of errors in the frame which challenges the error correction code. To cope with this, Crelay adopts *interleaving*. Basically, before transmission, the data bytes in a frame are mapped to random locations according to a random permutation. When a packet is received, the reverse of the random permutation is applied before processing. The effect of this is that errors are relocated to random locations and spread evenly in the frame.

C. Protocol Basics

1) Definitions: We give the definitions of terms to be used:

- *Block*: A data packet from the upper layer is divided into blocks of equal size, padded if necessary.
- *Codeword*: A block is encoded according to the RS code into a codeword, which is basically the data bytes followed by the generated parity bytes.
- *Segment*: A continuous segment of bytes in a codeword, represented by two integers as the start and end locations of the segment.
- *Record*: The received segment(s) in a codeword. The segment(s) may be scattered in the codeword and usually do not overlap. If they overlap, for the overlapping part, the one with less estimated errors is used.

We say a record is *decodable* if the original data block can be recovered from it. If a block has been recovered from the record, we say the block has been decoded. If all blocks in a packet have been decoded, we say the packet has been decoded.

2) Choice of Block Size: The block size should be selected such that a sufficient number of parity bytes are available for error correction. For example, in our current implementation, each block is 150 bytes and is encoded according to the (255, 150) RS code with 105 parity bytes. We note that although the number of parity bytes is large, not all parity bytes are transmitted in the majority of the cases, because Crelay estimates the number of errors and sends just enough number of parity bytes.

3) Receiving a Packet: A node always monitors the channel. When it receives a packet, it checks if the packet is *relevant*, i.e., if it is on the path of the packet and is on the downstream of the sender. If the packet is relevant but has been decoded, it sends an ACK; if the packet is relevant but has not been decoded, it adds the received segments to its records of this packet. The node then runs AMPS to estimate the number of errors in the received segments to determine if the records are decodable. If the records are not decodable, when the node gets access to the medium, it announces the receive status of the packet, which includes the

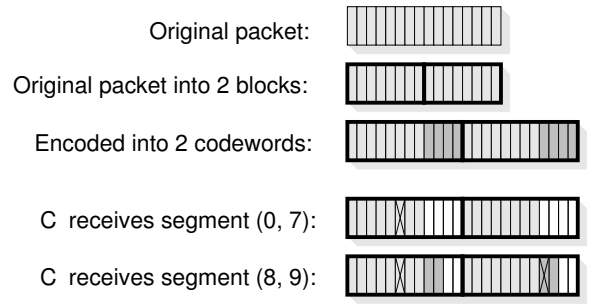

Fig. 1. An example. Light gray: data bytes. Dark gray: parity bytes.

start location, the end location, and the estimated number of errors of each received segment. If the records are decodable, the node attempts to decode. If the decoding is successful, it sends an ACK; otherwise it announces the new receive status and waits for additional segments until all records are decoded or until a timeout. It could happen that the decoding fails only at certain blocks; in this case the node announces a mask of the decoded blocks, and the upstream node will transmit data only for the undecoded blocks. A node delivers a decoded packet to the upper layer if it is the destination. When a node receives an ACK from *any* downstream node, it deletes the corresponding packet.

4) Sending a Packet: A node maintains a queue for each neighbor. A packet is in the queue for neighbor A if its next hop node is A. The queues are served according to round-robin; packets in the same queue are first-come-first-served.

A packet may be in three states, S0, S1, and S2. In state S0, some information has been overheard but the packet is not decodable. In state S1, the packet has been successfully decoded, but the receive status at the next hop is unknown. In state S2, either the packet has been decoded and the receive status at the next hop is known, or the packet has been in state S1 for longer than a threshold. To enable efficient packet forwarding, *only packets in state S2 can be sent*. As a special case, at the source node, a packet from the upper layer is automatically set to be in state S2.

A packet may have many blocks encoded into many codewords. When sending a packet, a node transmits the same segment in all codewords because the errors are evenly distributed after interleaving and deinterleaving. A node chooses a minimum size segment such that the next hop node should be able to decode the packet; the receive status of nodes further downstream are not considered for simplicity. To be specific, the node runs a linear search on a set of candidate locations as the start location of the segment to be sent, and picks the one that results in the minimum size segment. The list of locations includes 1) the first byte of the codeword, 2) the start location of each segment that has been overheard by the next hop, and 3) the byte immediately following the end location of each segment that has been overheard by the next hop.

D. A Simplified Example

Fig. 1 illustrates the main concepts related to packet forwarding in Crelay, continuing with the example in Section I where a packet is to be sent along path $A \rightarrow B \rightarrow C$. The data packet is assumed to be 16 bytes divided into 2 blocks. Each block is encoded into a codeword with 4 parity bytes. The packet forwarding process involves the following sequence of events:

1) Node A receives a packet from the upper layer and sets it in state S2. It transmits segment (0,7), i.e., the data bytes, for both blocks.
2) Node B receives the packet correctly, sets it in state S1, and sends ACK. Node A receives the ACK and deletes the packet. Node C overhears a partial packet with 1 error byte in the transmitted segment for block 0, and sets the packet in state S0.
3) Node C estimates the number of errors in each segment to be 1, and announces (0,7,1) as its receive status, which means that it overhears segment (0,7) and estimates that there is 1 error. Note that this announcement may be piggybacked with another data packet node C may have to send, for example, to node B.
4) Node B receives the announcement from node C, and promotes the packet to state S2. Seeing that there is 1 error, node B transmits segment (8,9), because the 2 parity bytes should correct 1 error.
5) Node C receives the segments transmitted by node B. The channel actually corrupts 1 byte in the transmitted segment for block 1. Node C estimates that there is no error in the new transmission, which is an incorrect estimation, and believes that it has two segments in its record: (0,7,1) and (8,9,0). As 2 parity bytes should correct 1 error, it attempts to decode and luckily decodes both records. Node C sends ACK and node B deletes the packet.

E. Optimizations

1) ACK Triggered Record: Consider a node A and suppose its next hop is node B. As mentioned earlier, node A must wait for the receive status of a packet at node B before sending the packet. If node B overhears the packet, it should be able to announce the receive status when it gets access to the medium. The challenge is that node B may never overhear the packet hence never announces the receive status, and node A may hold the packet in state S1 for a long time, increasing the packet delay. Crelay solves this problem with a simple trick called "ACK triggered record." That is, Crelay let node B create an empty record of a packet once it overhears an ACK for this packet, even when no data has been overheard. The empty record will prompt node B to announce an empty receive status, which will allow node A to promote the packet to state S2. Note that node A should send an ACK after it decodes the packet, and most likely, this ACK can be overheard by node B because they are neighbors on the path who should share a relatively good link. Therefore, node B is usually able to create a record for any packet node A decodes. As the MAC protocol should be fair, after node A sends the ACK, node B will likely get the medium and be able to announce the receive status.

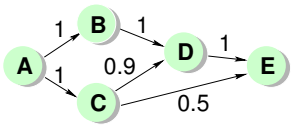

Fig. 2. Routing example with Crelay.

2) ACK Propagation: Crelay should also reduce duplicate transmissions. That is, a node should not transmit a packet $\mathbf{P}$ if at least one of its downstream nodes has already received $\mathbf{P}$ correctly. The main challenge is that the upstream node needs to know that the downstream node has got $\mathbf{P}$, but it may not be able to hear the ACK from the downstream node directly. For example, they may not be adjacent on the path and the link between them may be poor. To solve this problem, Crelay exploits the fact mentioned earlier, i.e., the upstream node usually has to send other packets first before sending $\mathbf{P}$ due to queuing delay, which allows extra time for the ACK from the downstream node to be *propagated* to the upstream node. Basically, in Crelay, a node is called a *bypassed node* with respect to a packet $\mathbf{P}$, if it has not decoded $\mathbf{P}$ while at least one of its downstream nodes has decoded $\mathbf{P}$. A node finds that it has been bypassed if it hears an ACK for $\mathbf{P}$ or hears $\mathbf{P}$ itself from a downstream node. In such cases, the bypassed node sends an ACK for $\mathbf{P}$, which, in essence, is to propagate the ACK from the downstream to the upstream. Once a bypassed node sends such an ACK, it considers itself decoded the packet and will not send such ACKs again.

III. Routing With Crelay

In this section, we discuss the routing problem in Crelay. Routing in Crelay is interesting because a sub-path of an optimal path may no longer be optimal. For example, consider the simple network shown in Fig. 2. The number besides a link is the *receiving ratio*, defined as the number of bytes that can be decoded if the sender sends one byte. It represents the quality of the link and is determined by the error ratio and the FEC code adopted. The best path from A to D is clearly $A \rightarrow B \rightarrow D$, but the best path from A to E is $A \rightarrow C \rightarrow D \rightarrow E$, due to the overhearing link from C to E.

A. Path Metric

We use the *Expected Transmission Byte (ETB)* as the metric of a path, which is the expected number of bytes sent in total such that the destination can receive one byte of data. It can be calculated based on the erasure ratio and error ratio of the links on the path.

To be more specific, denote a path as $P = v_0 \rightarrow v_1 ... \rightarrow v_n$. Let $L[v_i]$ denote the expected *load* of v_i, defined as the expected number of bytes v_i should send. The metric of path P is clearly $M[P] = \sum_{i=0}^{n-1} L[v_i]$. ETB captures the overhearing capabilities by considering the amount of bytes that v_{i+1} has overheard from (v_0, v_1, ..., v_{i-1}) when determining $L[v_i]$; the more v_{i+1} has overheard, the less v_i has to send. Let the erasure ratio and error ratio of link $v_i \rightarrow v_j$ be $e_{i,j}$ and $q_{i,j}$, respectively. With the RS code, the receiving ratio defined earlier is $d_{i,j} = (1 - e_{i,j})(1 - 2q_{i,j})$. We maintain the expected number of bytes that have been overheard at each node, denoted as $H[v_i]$ for node v_i, which is initially 0. Assuming the transmissions are independent of each other, $L[v_i]$ can be computed iteratively, starting from v_0, shown in Algorithm 1. Note that line 3 calculates the expected number of bytes v_i should send to v_{i+1} where $1 - H[v_{i+1}]$ is the number of bytes still missing at v_{i+1}; the reverse link quality $d_{i+1,i}$ is considered because v_i will be expecting the ACK from v_{i+1} and will transmit again if the ACK is lost. The check in line 5 makes sure that the path is valid, because if the condition is true, the path does not have to visit v_{i+1}; otherwise, $H[v_j]$ is increased by $L[v_i]d_{i,j}$ due to overhearing. With two levels of loops, Algorithm 1's complexity is $O(n^2)$ where n is the number of nodes on the path.

Algorithm 1 Path Metric Calculation

```
1:  Set H[v_i] ← 0 and L[v_i] ← 0 for all 0 ≤ i ≤ n.
2:  for i = 0 to n − 1 do
3:      L[v_i] ← (1 − H[v_{i+1}]) / (d_{i,i+1} d_{i+1,i})
4:      for j = i + 2 to n do
5:          if H[v_j] + L[v_i] d_{i,j} d_{j,i} > 1 then
6:              return INVALID
7:          end if
8:          H[v_j] ← H[v_j] + L[v_i] d_{i,j}
9:      end for
10: end for
11: return Σ_{i=0}^{n−1} L[v_i]
```

B. Routing Algorithm

We adopt a greedy routing algorithm inspired by the Dijkstra's algorithm described in Algorithm 2. Same as Dijkstra, a set π is maintained which keeps the nodes whose paths to the source node have been determined. In each iteration, a node not in π is selected and added to π, if its best path to the source node is the shortest among all nodes not in π. Different from Dijkstra, each node has up to w candidate paths. In each iteration, the candidate paths are updated when a new node is added to π by checking if better paths can be found by going through this new node. The algorithm returns the best candidate path for each node when terminates. The source node is denoted as v_0 and the k_{th} candidate path from v_0 to v_i is denoted as $P_k(v_i)$ where $0 \leq k < w$. The complexity is $O(N^2 w)$ where N is the number of nodes in the network.

IV. Implementation

The Crelay protocol discussed earlier is independent of implementation. We implement a Crelay prototype in software within the Click modular router [15] due to the relative simplicity over the hardware implementation. The prototype consists of around 5,000 lines of C++ code.

A. Frame Format

The Crelay frame format is shown in Fig. 3. After the MAC header, a Crelay frame consists of three main sections, the header, the announcements, and the data packets:

Algorithm 2 A Greedy Routing Algorithm

1: $\pi \leftarrow \{v_0\}$.
2: $P_0(v_i) \leftarrow (v_0, v_i)$ for all v_i where $i > 0$. All other candidate paths set to be empty.
3: **while** there are nodes not in π **do**
4: Let v_u be the node not in π with the best candidate path.
5: $\pi \leftarrow \pi \cup \{v_u\}$
6: **for** all v_j not in π **do**
7: **for** $k = 0$ to $w - 1$ **do**
8: $P \leftarrow (P_k(v_u), v_j)$.
9: Let $P_t(v_j)$ be the candidate path of v_j with the largest metric. Replace $P_t(v_j)$ with P if $M[P_t(v_j)] > M[P]$.
10: **end for**
11: **end for**
12: **end while**

- The header is fixed 28 bytes, containing information such as the frame sequence number, the sender's ID, AMPS samples, etc., and is protected by an FEC code.
- The announcements section is variable length, containing ACKs, packet receive status, and headers of the data packets in this frame, also protected by an FEC code.
  - The ACK contains simply the source and destination IDs and the packet sequence number.
  - The packet receive status contains the source and destination IDs, the packet sequence number, the number of blocks in the packet, the mask of decoded blocks, and the information of the received segments.
  - The data packet header contains the source and destination IDs, the packet sequence number, the number of blocks, and the information of the transmitted segment.
- The data packets section contains the data. A frame may have multiple data packets, because one may be a fresh packet and the other may be the parity bytes for another packet.

B. Optimizations in the Implementation

In addition to the core of the Crelay protocol discussed earlier, we also incorporated several further optimizations in the implementation to better cope with the wireless channel. First, to cope with hidden terminals, we allow a node A to poll another node B, if node A has not heard any message from node B for longer than a threshold while having many packets to node B. After hearing the message, node B transmits while others backoff for a time. The reason is that node B's packets may be lost in collisions due to hidden terminals; the RTS/CTS mechanism can alleviate the hidden terminal problem but cannot be used for broadcast frames in 802.11. Second, we allow nodes to send a small amount of parity bytes along with the data bytes in the first transmission attempt if the link error ratio $q > 0$, such that packets with a small number of errors

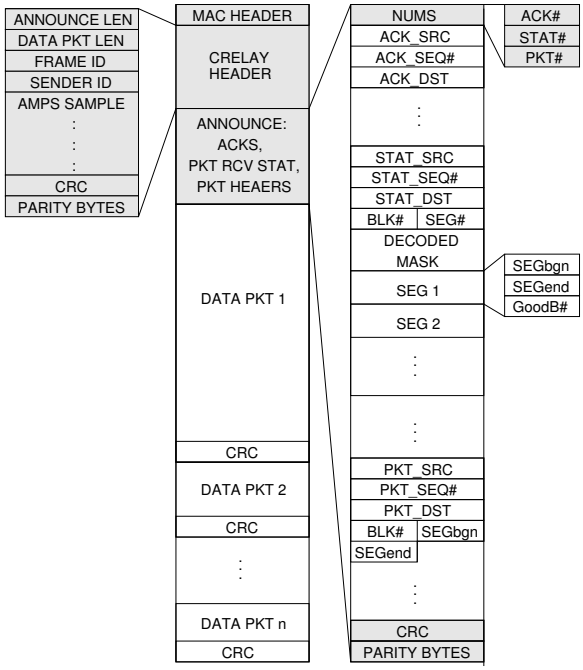

Fig. 3. Crelay frame format. The shaded fields are mandatory, others are optional.

can be recovered immediately without incurring the feedback overhead. In our current implementation, a node transmits bytes that can correct $\min\{0.05, \max\{0.02, q\}\}$ fraction of errors for the bytes it sends. Such parity bytes cannot immediately recover packets with more errors but are still useful, i.e., they can be combined with the parity bytes transmitted later to correct errors. Third, in AMPS, we set the minimum number of estimated errors in a segment to be 3, as this does not increase much data transmission time but reduces the underestimation probability. Fourth, if AMPS underestimates the number of errors in a record, found by a decoding failure, we reduce the estimated number of correct bytes in each segment by 20 or set it to be 0 if it becomes negative. Fifth, we adopt a simple congestion control mechanism with which nodes check their buffers roughly every 100 ms and consider a flow congested if the number of buffered packets for this flow is above a threshold and not congested otherwise. If the flow is congested, a node sets a bit in the header for all transmitted packets of this flow and an upstream node will hold the transmission of packets of this flow once it overhears this bit. For each flow, this requires maintaining only a 1-bit state.

C. Limitation

Our current software implementation has one limitation: it runs at low data rates such as 1 Mbps because decoding the RS code in software can be time consuming [17]. However, the prototype enables us to evaluate the main features of Crelay such as packet forwarding and routing with over-the-air wireless transmissions. The future Crelay routers may add a dedicated decoding circuit to speedup the decoding; note that this does not require changing the wireless transceiver.

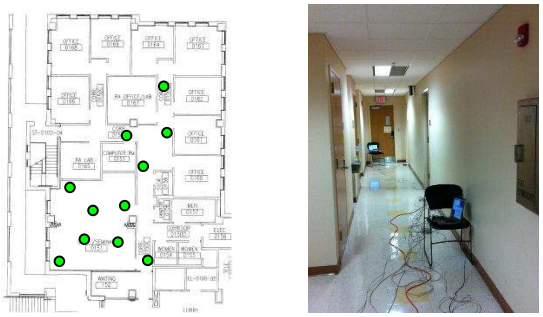

Fig. 4. The setup of the testbed.

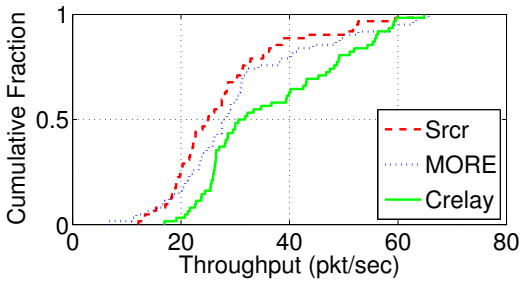

Fig. 5. The CDF of the flow throughput.

V. Evaluation

We evaluate the performance of Crelay with our prototype implementation. We compare with: 1) *MORE* [6], the benchmark opportunistic routing protocol, and 2) *Srcr* [11], the benchmark traditional routing protocol, where nodes use the shortest path according to the ETX metric to forward packet hop-by-hop without exploiting partial packets and overhearing. We use the original implementation at [16] for MORE and use our own implementation for Srcr. We cannot compare with MIXIT [7] because it needs to modify the wireless transceiver.

A. Testbed and Experiment Setup

Our testbed has 11 nodes, where the wireless nodes are laptop computers with the Cisco Aironet cardbus adapter. The 802.11b/g channel 3 is used when there is little external traffic during the experiments. Packets are transmitted in broadcast frames, same as MORE [16]. The Madwifi [12] driver runs in the monitor mode. The MTU is 2200 bytes. The transmission power is 1dBm. The data rate is 1Mbps. The testbed setup, as well as two of machines used in the experiments, are shown in Fig. 4. With this set up, there are 33 links where we consider a link exists if the erasure ratios of both directions are lower than 0.9. Among the links, the average RSSI is 10.6dB, the average erasure ratio is 0.138, and the average error ratio is 2%. The numbers of 2-hop, 3-hop, and 4-hop paths found by Crelay are around 30, 26, and 6 respectively which may change slightly due to channel fluctuations.

In each experiment, nodes first learn the link states by sending Hello packets in the first 9 seconds at random intervals, where a node sends around 50 packets in total. In the next 9 seconds, similar to MORE [16], the link state is propagated with the help of a central node using wired links. At time 18 second, the topology learning phase ends and nodes begin to send data packets of size 1500 bytes and the experiment runs for another 10 seconds.

B. Flow Analysis

We select 62 pairs of nodes in the network, between which Crelay always finds the path to be more than one hop. Each pair is a *flow*. We run the experiment with only one active flow where the source node generates a packet every 10ms.

We note that two flows may be selected between one pair of nodes, one flow in each direction, because the link quality may be asymmetrical such that the forward flow and backward flow may use different paths. We only consider the multi-hop paths because Crelay is protocol for multi-hop networks and we wish to highlight performance of Crelay and other protocols on multi-hop paths. We note that a single-hop path is simply a link. MORE and Srcr will have similar performances on a single link because opportunistic routing relies on packet overhearing at multiple nodes and does not have any gain on a single link. Crelay should have better performance than MORE and Srcr on a single link due to efficient partial packet recovery; the gain of partial packet recovery on a single link has been demonstrated in earlier works such as [2], [17].

1) Throughput Comparison: We first show in Fig. 5 the CDF of the throughput of Crelay, MORE, and Srcr measured in the number of received packets by the destination per second. We can see that Crelay's throughput is significantly higher than both MORE and Srcr. Fig. 6 shows the scattered plots of Crelay v.s. MORE and Crelay v.s. Srcr for each flow. In the scattered plot, a point on the 45-degree line represents a flow in which two compared schemes have the same throughput. We can see that Crelay outperforms MORE in most flows and outperforms Srcr in almost all the flows.

2) Throughput Gain Analysis: We note that MORE is an efficient opportunistic routing protocol capable of exploiting overhearing with random network coding. Crelay is also capable of exploiting overhearing with its control message exchange mechanism; in addition, Crelay can recover the partial packets efficiently. The gain of Crelay over MORE is more likely to be due to the exploitation of partial packets. This is confirmed by Fig. 7 which shows the throughput gain of Crelay over MORE and the partial packet ratio of each flow; we can see that there is a positive correlation between the gain and the partial packet ratio. The gain of Crelay over Srcr is due to both overhearing and partial packet recovery.

3) Avoiding Duplicate Transmissions: We define the *duplicate data* as the data transmitted by an upstream node for a packet when at least one of the downstream nodes has already received the packet correctly. An opportunistic routing protocol can be measured by the duplicate percentage, i.e., the percentage of data bytes in the duplicate packets over all transmitted data bytes. We collect the duplicate percentage of Crelay for each tested flow and show the CDF in Fig. 8. We can see that the median is only 1.52%, therefore Crelay effectively avoids duplicate transmissions.

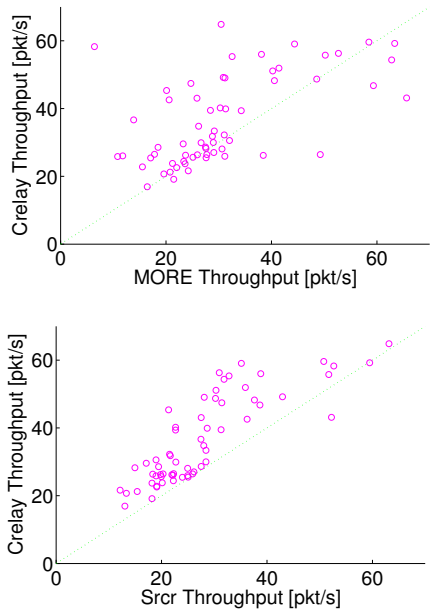

Fig. 6. The scattered plot of the flow throughput.

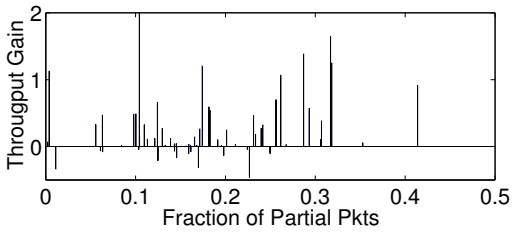

Fig. 7. The gain and the fraction of partial packets.

4) Protocol Overhead: We also collect the percentage of Crelay overhead for each tested flow, which is defined as the percentage of bytes sent in the Crelay header and the announcement section among all network layer bytes. Fig. 9 shows the CDF of overhead percentage, where we can see the median is 7.67%.

VI. Conclusion

In this paper, we propose Coded Relay (Crelay) for multi-hop wireless networks. With Crelay, nodes exploit partial packets and overhearing for packet forwarding. To recover a partial packet, Crelay nodes can often send parity bytes to the next hop with size significantly smaller than the size of the packet itself. Our study includes the protocol design and a practical routing algorithm. We test Crelay on an 11-node testbed, and the results show that Crelay achieves significant gain over the compared protocols.

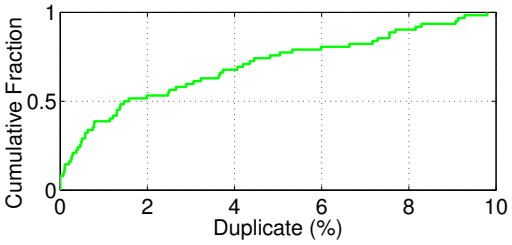

Fig. 8. Crelay duplicate percentage.

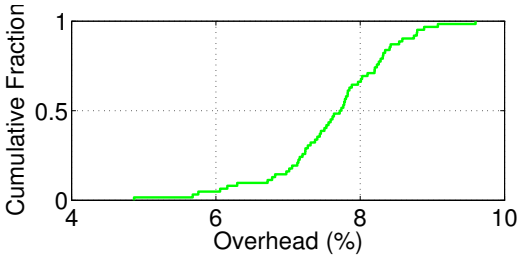

Fig. 9. Crelay overhead percentage.

References

[1] B. Han, A, Schulman, F. Gringoli, N. Spring, B. Bhattacharjee, L. Nava, L. Ji, S. Lee, and R. Miller, "Maranello: Practical partial packet recovery for 802.11," in *NSDI, 2010.*

[2] K. Lin, N. Kushman, and D. Katabi, "ZipTx: Harnessing partial packets in 802.11 networks," in *ACM Mobicom*, 2008.

[3] P. Razaghi and W. Yu, "Bilayer low-density parity-check codes for decode-and-forward in relay channels," *IEEE Trans. Inform. Theory*, vol. 53, no. 10, pp. 3723-3739, Oct. 2007.

[4] S. Avestimehr, S. Diggavi and D. Tse, "Wireless network information flow: a deterministic approach," *arXiv:0906.5394v6*, Dec., 2010.

[5] V. Venkatkumar, T. Wirth, T. Haustein, and E. Schulz, "Relaying in long term evolution: indoor full frequency reuse," in *European Wireless*, Aarlborg, Denmark, May 2009.

[6] S. Chachulski, M. Jennings, S. Katti and D. Katabi, "Trading structure for randomness in wireless opportunistic routing," in *ACM Sigcomm*, 2007.

[7] S. Katti, D. Katabi, H. Balakrishnan and M. Medard, "Symbol-level network coding for wireless mesh networks," in *ACM Sigcomm*, 2008.

[8] S. Biswas and R. Morris, "Opportunistic routing in multi-hop wireless networks," in *ACM Sigcomm*, 2005.

[9] T. Li, D. Leith, and L. Qiu, "Opportunistic routing for interactive traffic in wireless networks," in *ICDCS*, 2010.

[10] S. B. Wicker, *Error Control Coding for Digital Communication and Storage*, Prentice-Hall, NJ, 1995.

[11] J. Bicket, D. Aguayo, S. Biswas, and R. Morris, "Architecture and evaluation of an unplanned 802.11b mesh network," in *ACM Mobicom*, 2005.

[12] The MadWifi Project, *http://madwifi-project.org/.*

[13] http://www.ka9q.net/code/fec/

[14] Cisco Aironet 802.11a/b/g wireless cardbus adapter, http://www.cisco.com/.

[15] The Click Modular Router, *http://read.cs.ucla.edu/click/.*

[16] *http://people.csail.mit.edu/szym/more/README.html.*

[17] J. Xie, W. Hu, and Z. Zhang, "Revisiting partial packet recovery in 802.11 wireless LANs," in *ACM Mobisys*, 2011.

[18] Z.Zhang, W. Hu, and J. Xie, "Employing coded relay in multi-hop wireless networks," in *arXiv:1012.4136v1*, http://arxiv.org/abs/1012.4136.