

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 study Coded relay (Crelay) in multi-hop wireless networks. Crelay exploits both partial packets and overhearing capabilities of the wireless nodes, and uses Forward Error Correction code in packet forwarding. When a node overhears a partial packet from an upstream node, it informs the upstream node about the number of parity bytes needed to correct the errors, such that the upstream node need only send a small amount of parity bytes instead of the complete packet, hence improving the network efficiency. Our main contributions include the following. First, we propose an efficient network protocol that can exploit partial packets and overhearing. Second, we study the routing problem in networks with Crelay and propose a greedy algorithm for finding the paths. Third, we propose an error ratio estimator, called AMPS, that can estimate the number of byte errors in a received frame with good accuracy at a low overhead of only 8 bytes per frame, where the estimator is needed for a node to find the number of needed parity bytes. Fourth, we implement the proposed protocol and algorithm within the Click modular router, and our experiments show that Crelay can significantly improve the performance of wireless networks.

I. INTRODUCTION

Wireless multi-hop networks, e.g., wireless mesh networks, wireless sensor networks, have attracted much interests in recent years. In this paper, we propose Coded Relay, abbreviated as *Crelay* in the following, which exploits two fundamental properties of transmissions over the wireless medium, namely the existence of partial packets and the overhearing capability. That is, partial packets are often received by wireless nodes that are not completely correct but still contain a significant amount of correct information. Also, because the medium is shared, when a node transmits a packet to second node, a third node may overhear this packet.

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 Forward Error Correction (FEC) code. As a simple example, suppose a path is $v_A \rightarrow v_B \rightarrow v_C$, while v_A wishes to send a packet \mathbf{P} to v_C . v_A first transmits \mathbf{P} , after which v_B gets \mathbf{P} while v_C gets a partial packet with some errors. v_C estimates the number of errors using an error ratio estimator, and asks v_B to send just enough number of parity bytes correct these errors, instead of sending the entire packet. Thus, fewer bytes are transmitted and a better efficiency is achieved. By sending FEC-coded messages, Crelay opens up new possibilities for packet forwarding in multi-hop wireless networks.

Although the idea is simple, the design of Crelay faces the

following challenges. First, a protocol should be designed for control message exchange that facilitates packet forwarding at low overhead and low delay. The control message should allow an upstream node to be informed about the receiving status of a packet to determine whether the packet should be transmitted and if so, the number of parity bytes needed. Note that a packet transmission may reach a far node on the path due to a lucky overhearing, and the bypassed nodes should be exempted from the duty of forwarding because their transmissions are pointless at this moment. Also, the upstream node should be aware of the number of errors in a received packet such that it can send just enough number of parity bytes to correct the errors. Avoiding pointless transmission is the classic challenge facing all opportunistic routing protocols. Existing approaches include structured forwarder coordination [6] which may discourage spatial reuse [4], or randomized network coding [4], [5] which cleverly eliminates the need of feedback but cannot be used in Crelay because Crelay needs the feedback to determine the number of parity bytes. To this end, we give a novel, simple solution, based on two key observations: (1) *packets usually experience queuing delays at the nodes*, specially under high load when throughput should be optimized; (2) *the lucky overhearing usually bypasses a small number of nodes on the path* such that it is possible to propagate the status of an overheard packet to the upstream in a timely manner. Therefore, with a good feedback mechanism, a node can often obtain the receiving status of a packet from its downstream nodes before starting to serve this packet, because it has to serve other packets first. We design an efficient feedback mechanism for Crelay which scavenges all overheard useful information and adopts two tricks we call the *ACK triggered record* and *ACK propagation*. The overhead is low because all feedback information is piggybacked with data packets whenever possible, not necessarily the packets belonging to the same flow.

Second, an algorithm is needed to calculate the best path in Crelay. The routing problem is interesting because a sub-path of an optimal path may not be optimal, due to partial packets and overhearing. We study the routing problem and propose a practical heuristic algorithm for finding the paths.

Third, an estimator is needed to find the number of errors in a received packet, because the receiving node actually does not know the number of errors. We propose an error ratio estimator, referred to as *AMPS*, which is based on the optimal maximum a posteriori (MAP) estimation. AMPS adds only 8 bytes per-frame as overhead and computes the estimate with a constant time table lookup. Our simulations also show that for

per-packet level estimation, AMPS is more accurate than EEC [13], a recently proposed error ratio estimator, at much less overhead. Our experiments show that AMPS can achieve good accuracy, e.g., for more than 58% of the time, its estimation error is no more than 3 bytes among typically 150 transmitted bytes.

We implement Crelay within the Click modular router [19], and test its performance in an 11-node testbed. The results show that Crelay achieves a significantly better performance than the traditional single path routing scheme as well as More [4] which the state-of-art opportunistic routing protocol without physical layer hint. For example, the average throughput gain of Crelay over More is 36% in our experiments.

The rest of the paper is organized as follows. Section II discusses the related works. Section III describes the Crelay protocol. Section IV discusses the problem of finding paths. Section V describes the AMPS error ratio estimator. Section VI gives the experimental results. Section VII concludes the paper.

II. RELATED WORK

Partial packets and overhearing opportunities have been realized and studied extensively in recent years, see, for example, [11], [7], [6], [4], [5]. Compared to other opportunistic routing protocols, Crelay operates more like a traditional routing protocol, in that (1) Crelay forwards packets in a per-packet basis while ExOR [6], More [4], and MIXIT [5] require a batch of packets to be assembled before the transmission, (2) Crelay maintains per-neighbor buffer while ExOR, More, and MIXIT maintain per-flow buffer hence have higher protocol complexity. In addition, network-coding-based solutions such as More and MIXIT need to solve linear equations to recover every packet, which poses high requirements on the computational capabilities as well as power capabilities. Another core difference between Crelay and MIXIT is that Crelay does not rely on physical layer hints to handle partial packets, hence can be used in a wider range of application scenarios because physical layer hints are not always available [7].

Crelay uses FEC code in a network, however it is different from network coding. Wireless network coding combines multiple packets into a coded packet, usually through a linear transformation, and broadcast the coded packet [8], [4], [5], [9], [10]. Crelay does not mix multiple packets and does not incur the associated computational cost.

Introducing relaying nodes has been proposed for cellular phone networks. For example, LTE base stations may employ relaying devices in a cell which can significantly improve the throughput [3]. The idea of Crelay was actually originated from *corporate relaying*, where itself is still an active research area in the signal processing community [1], [2]. Existing research on corporate relaying considers forwarding messages in a two-hop scenario; however, Crelay focuses on multi-hop networks, where the routing problem is much more challenging.

Recently, Error Estimation Coding (EEC) [13] was proposed for estimating the error ratios. EEC focuses on the average error ratio of a link, while APMS focuses on the individual error ratio of the packets. EEC, with non-trivial probability, may output 0 as the estimated number of errors for partial

packets, which is tolerable for link-level estimation after taking the average; however, it cannot be applied directly to Crelay which needs accurate, per-packet estimation. We make a more detailed comparison between EEC and AMPS in Section V-C, where we show that for per-packet estimation, AMPS outperforms EEC with lower overhead.

III. THE CRELAY PROTOCOL

We describe the Crelay protocol in this section.

A. Preliminaries

Crelay is basically a link state protocol where nodes learn and propagate the quality of the links. A link is measured by two values, namely the *erasure ratio* and the *error ratio*, where the former is the fraction of frames that are unaware of by the receiver and the latter is the fraction of the erroneous bytes in a received frame. A *path* is an ordered list of nodes that will participate in relaying the packet. All nodes appearing earlier than a node in the list are the *upstream nodes*, and all nodes appear later the *downstream nodes*. Nodes determine the packet forwarding paths based on the link states, and every node should find the same path for a particular pair of source and destination. When received a packet from the upper layer, the source node inject the packet into the network whenever it gets the opportunity and nodes in the forwarding path will collaborate in relaying it, bypassing some nodes sometimes, until it reaches the destination. A node maintains a queue for each neighbor, and buffers a packet in the queue for neighbor v_A if the next hop of the packet is v_A . Packets in the a queue are first-come-first-served; different queues are served according to round-robin.

Crelay can work with any error correction code. The current implementation uses the Reed-Solomon (RS) code, because of its strong error correction capability and the availability of software implementations [17]. Using the RS code, if there are e erroneous bytes in the received data belonging to a codeword, with *any* additional $2e$ bytes in the codeword, all errors can be corrected. A node does not have to receive all bytes in the codeword; the part that are not transmitted can be treated as *erasures* [12].

Errors in the packets may occur in bursts, which is not desirable because it may result in one codeword handling too many errors, thus exceeding the decoding capability, while others handling too few. To cope with this, Crelay adopts *interleaving*. Basically, before transmission, bytes in the data field of a frame are mapped to random locations according to a random permutation, and at the receiver, the reverse of the random permutation is applied before handing the packet for processing. The effect of this is that errors are relocated to random locations such that they are spread evenly in the packet.

B. Block, Codeword, Segment, and Record

The four main concepts in Crelay with regarding to packet forwarding are *block*, *codeword*, *segment*, and *record*:

- **Block:** a data packet from the upper layer is divided into a number of 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 an interval of integers.
- Record: the received segment(s) in a codeword. The segment(s) may be scattered in the codeword. They usually do not overlap; in case of overlap, for the overlapping part, the one with less errors is used.

In our current implementation, each block is 150 data bytes, and each codeword is 255 bytes according to the (255, 150) RS code. When transmitting a packet, for simplicity, a node transmits the same segment in all codewords. 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 also say it is decoded. If all blocks in a packet are decoded, we say the packet is decoded.

C. Basic Operations

1) *Receiver*: A node always monitors the channel. When receives a data packet, if it is on the forwarding path of this packet, found based on the source and destination ID, it checks whether it is on the downstream of the packet sender. If yes, it adds the newly received segments to its records of this packet. It then runs AMPS to estimate the number of errors. As the number of errors in the transmitted segments may vary, it estimates the maximum number of errors in a segment among all segments, and uses it as the estimate for all segments. It then estimates whether the records are decodable. If no, when gets access to the medium, it announces the receiving status of the packet, which includes the start location, end location, and estimated number of errors of each segment that has been overheard. After receives another segment, a node may attempt to decode. If the decoding is successful, it sends an ACK; otherwise it announces the new receiving status and waits for the next segment until all records are decoded or a timeout. It could happen that the decoding fails only at certain records; in this case the node announces a mask of the decoded records, and the upstream node will transmit segments only for the undecoded records.

2) *Sender*: The sender, when transmitting for a packet, chooses a minimum size segment such that the next hop node should be able to decode the records. The receiving status of nodes further downstream are not considered because they may be better served by nodes closer to them. Basically, it runs a linear search at selected locations and estimates the number of bytes needed if the segment starts at this location, and picks the one that needs the minimum. The list of locations include the start and end of each segment that has been overheard, plus the first byte of the codeword.

D. A Simplified Example

Fig. 1 is a simple example for the illustration of the concepts. The data packet from the upper layer is assumed to be 16 bytes, and is organized into 2 blocks. Each block is encoded into a codeword with 4 parity bytes. The first transmission is segment (0,7), i.e., the data bytes, for both blocks. The channel corrupted 1 byte in the transmitted segment for block

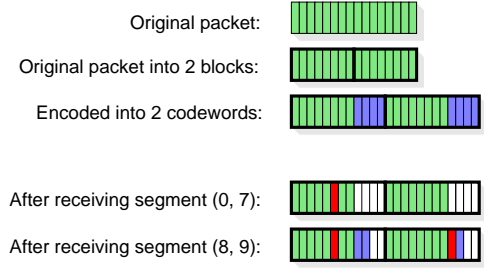


Fig. 1. An example of Crelay. Green: data bytes. Blue: parity bytes. Red: corrupted bytes.

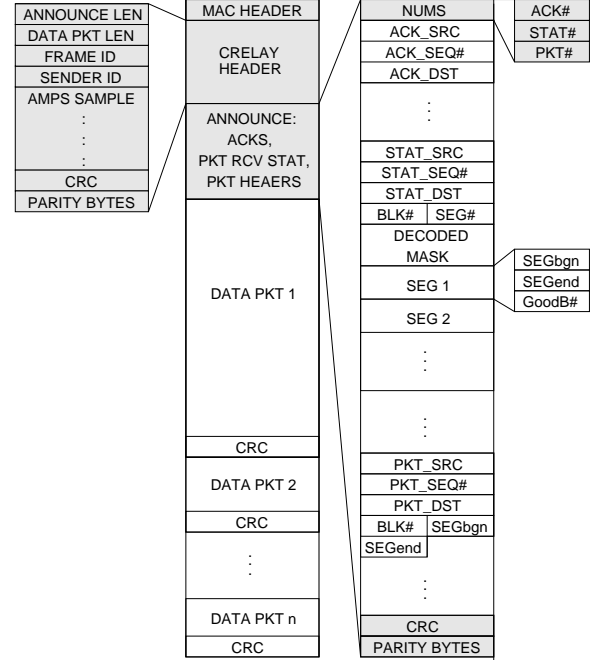


Fig. 2. Crelay frame format. The shaded fields are mandatory, others are optional.

0. The receiving node estimates the maximum number of errors among the two transmitted segments to be 1, and announces (0,7,1) as the receiving status. The sending node, seeing that there is 1 error, transmits segment (8,9), because the 2 parity bytes should correct the error. The channel actually corrupts 1 byte in the transmitted segment for block 1. The receiving node estimates that there is no error in the received segments, 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 the records and luckily decoded both.

E. Frame Format

After the MAC header, a Crelay frame consists of three main sections, the header, the announcements, and the data packets, as shown in Fig. 2. The header contains information such as the frame sequence number, the sender's ID, AMPS samples, etc., and is protected by an FEC code. The announcements section contains ACKs, packet receiving status, and headers of the data packets in this frame, also protected by an FEC code. The ACK contains simply the source and destination ID and the packet sequence number. The packet receiving status contains the source and destination ID, the packet sequence number, the

number of blocks in the packet, the mask of decoded blocks, and the list of received segments. The data packet header contains the source and destination ID, the packet sequence number, the number of blocks, and 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.

The overhead of the Crelay protocol is mainly the Crelay header and the announcement section. The Crelay header is fixed 28 bytes. The length of the announcement section may vary depending on the number of ACKs, packet receiving status, and data packets, and is usually no more than 60 bytes. Our experiments show that Crelay is able to achieve improved performance over other protocols at this overhead.

F. Addressing the Protocol Design Challenges

We now discuss how Crelay meets the main design challenges.

1) *ACK Triggered Record*: Crelay should first ensure that a node sends the correct amount of parity bytes to the next hop node. Denote a node as v_A and its next hop as v_B . This requires v_B to get a good estimate of the number of errors, addressed by AMPS to be discussed later; also, v_A should wait for the receiving status of v_B before sending the packet. Crelay therefore classifies packets into three states, S0, S1, and S2: (1) in state S0, some information has been overheard about this packet, but it still not decodable; (2) in state S1, the packet has been successfully decoded, but the receiving status of the packet at the next hop is not known; (3) in state S2, the packet has been successfully decoded, and the receiving status of the next hop is known. Only packets in state S2 can be transmitted.

If v_B overhears the packet, v_B should be able to announce the receiving status when it gets access to the medium. The challenge is that v_B may not have overheard it hence never announces the receiving status, and v_A may hold the packet in state S1 forever. Crelay solves this problem with a simple trick called “ACK triggered record.” That is, it let nodes create an empty record of a packet once overheard an ACK for this packet, even when no data is overheard. The empty record will prompt v_B to announce an empty receiving status, which will allow v_A to promote the packet into state S2. The rationale behinds this is that v_A will send an ACK once it decoded the packet, and most likely, this ACK can be overheard by v_B because they are neighbors on the path who should share a relatively good link. As nodes should follow a fair MAC protocol to access the channel, after v_A sends the ACK, v_B is likely to get the channel and be able to announce the receiving status. Thus this solution does not increase much of the delay. In the case that v_B did not overhear the data packet and the ACK, Crelay relies on timeout and allows a packet to be promoted to state S2 if it has been in state S1 for longer than a threshold.

2) *ACK Propagation*: Crelay should also ensure that the “good forwarders” forward the packet to reduce pointless transmissions. That is, suppose a path is $v_0 \rightarrow v_1 \rightarrow \dots \rightarrow v_{n-1} \rightarrow v_n$. After v_i gets the packet due to a lucky overhearing, v_j should not transmit if $j < i$. To achieve this, Crelay adopts a simple strategy based on “ACK propagation.” Basically, an ACK will

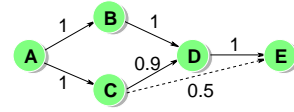


Fig. 3. Routing example with Crelay.

be propagated from the downstream to the upstream whenever needed, helping the upstream nodes to remove packets that no longer need to be transmitted.

To be more specific, in Crelay, a node sends ACK in three cases: (1) when it decoded the packet, (2) when it removed the packet recently because received an ACK from a downstream node but heard an upstream node sending it again, and (3) when it overheard an ACK or the data packet from a downstream node while it has not decoded the packet itself. Any node who gets an ACK for a packet from a downstream node will delete the packet. Case (1) is obvious and Case (2) is because if the upstream node is still sending the packet, it did not get an ACK from any of its downstream nodes and the downstream node should send ACK again. Case (3) guarantees that if a node sends an ACK or sends the packet itself, both can be regarded as a valid ACK, this ACK will be propagated and eventually known to the upstream nodes that are waiting for ACKs. The propagation could take time, and it might be a concern that a node may make an unnecessary transmission before the ACK is propagated to it. However, as discussed earlier, the packet queuing delay usually prevents this from happening. Also, in most cases, this lucky overhearing bypasses a small number of links so the propagation delay is usually small.

IV. 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. 3. 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 of bytes needed to be 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 $P = v_0 \rightarrow v_1, \dots, \rightarrow v_n$. 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 $M[P] = \sum_{i=0}^{n-1} L[v_i]$. $L[v_i]$ is determined by the quality of the link between v_i and v_{i+1} , as well as the amount of bytes that v_{i+1} has overheard from $(v_0, v_1, \dots, v_{i-1})$. The more it 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 is

$d_{i,j} = (1 - e_{i,j})(1 - 2q_{i,j})$. We maintain the expected number of bytes that are overheard so far at each node, denoted it as $H[v_i]$ for node v_i , which is initially 0. Assuming transmissions are independent, $L[v_i]$ can be computed iteratively, starting from v_0 , shown in Algorithm 1. Note that line 3 calculates

Algorithm 1 Path Metric Calculation

```

1: Set  $H[v_i] \leftarrow 0$  and  $L[v_i] \leftarrow 0$  for all  $0 \leq i \leq n$ .
2: for  $i = 0$  to  $n - 1$  do
3:    $L[v_i] \leftarrow \frac{1 - H[v_{i+1}]}{d_{i,i+1}d_{i+1,i}}$ 
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:      $H[v_j] \leftarrow H[v_j] + L[v_i]d_{i,j}$ 
9:   end for
10: end for
11: return  $\sum_{i=0}^{n-1} L[v_i]$ 

```

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} , and $d_{i,i+1}d_{i+1,i}$ is the expected number of bytes that v_i has to transmit to *make sure* that v_{i+1} receives one byte. 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 an amount of $L[v_i]d_{i,j}$ due to overhearing. With two levels of loops, Algorithm 1's complexity is $O(n^2)$.

B. Routing Algorithm

We adopt a greedy algorithm described in Algorithm 2 which is similar to the Dijkstra's algorithm. 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 that has the shortest path to the source node visiting only nodes in π . Different from Dijkstra, each node has up to w candidate paths. In each iteration, the candidate paths will be updated when a new node is added to π . 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 $O(N^2w)$ where N is the number of nodes in the network.

V. ERROR ESTIMATION

A key component of Crelay is error estimation. A node must know the number of error bytes in a partial packet to be able to ask for the correct number of parity bytes from its upstream node. Our error estimator is referred to as *AMPS*, because it is based on the idea of *Amplified Sampling*. A naive sampling method would be taking samples of the data bytes and use the ratio of erroneous sampled bytes as an estimate of the error ratio. However, because the raw byte error ratio is usually within $[0, 0.2]$ and is often very small, e.g., 0.01, the naive method may result in high estimation error as it may never

Algorithm 2 A Greedy Routing Algorithm

```

1:  $\pi \leftarrow \emptyset$ .
2:  $M[P_0(v_i)] \leftarrow \frac{1}{d_{0,i}d_{i,0}}$ , for all  $v_i$ 
3:  $M[P_k(v_i)] \leftarrow \infty$  for all  $v_i$  and  $k$  where  $k \neq 0$ .
4: while there are nodes not in  $\pi$  do
5:   Let  $v_u$  be the node not in  $\pi$  with the best candidate path.
6:    $\pi \leftarrow \pi \cup v_u$ 
7:   for all  $v_j$  not in  $\pi$  do
8:     for  $k = 0$  to  $w - 1$  do
9:        $P \leftarrow P_k(v_u) \parallel v_j$ .
10:      Let  $P_t(v_j)$  be the candidate path with largest metric. Replace  $P_t(v_j)$  with  $P$  if  $M[P_t(v_j)] > M[P]$ .
11:    end for
12:  end for
13: end while

```

U	number data bytes in a packet
S	number of segments in a packet
K	number of selected bytes for a sample
T	number of samples
X	number of error samples
Y	number of error bytes in the packet
Z	maximum number of error bytes in a segment

TABLE 1
LIST OF NOTATIONS FOR AMPS

sample any erroneous byte. AMPS computes a *sample* with multiple bytes which, in effect, amplifies the raw byte error ratio into a much larger sample error ratio, and achieves better estimation accuracy.

A. The Estimation Procedure

We first assume a frame contains only one data packet with U bytes into S segments of equal size. The sender randomly samples K data bytes, allowing repeat, and computes their parity bit. Each parity bit is a sample. Clearly, the probability that the sample's parity is flipped is much larger than the probability that a byte has errors. For example, if data byte error ratio is 0.01 and $K = 32$, the probability that the sample's parity is flipped is $(1 - 0.99^{32})/2 = 0.14$, assuming the values of the errors are random. A total of T samples are calculated in this manner, and the samples are transmitted in the Crelay frame header, protected by error correction code. When the receiver receives the frame, it calculates samples in exactly the same way based on the received data bytes. As some bytes may have been corrupted, the samples it calculates may be different from the samples in the frame header. We call a mismatching sample an *error sample*, and denote the number of error samples as X . X carries information about the error conditions in the frame and is used by AMPS as input to calculate the estimate. From a high level, AMPS first finds the maximum a posteriori (MAP) estimation of Y , the number of error data bytes in the packet. It then finds an estimation of Z , the maximum number of errors in a segment among all transmitted segments, such that the probability that Z is greater than its estimate is below a threshold. Table 1 lists the main notations related to AMPS.

The estimation involves three main steps.

Step 1. The conditional probability $P(X = x|Y = y)$. Note

that after interleaving, the error bytes are randomly distributed in the packet. Also, we assume the error bytes take random values. Therefore, the probability that a sample is an error sample when there are y error bytes, denoted as η_y , is

$$\eta_y = [1 - \binom{U-y}{K} / \binom{U}{K}] / 2$$

For simplicity, we treat the samples as independent. In this case, the probability that there are x error samples follows the binomial distribution:

$$P(X = x | Y = y) = \binom{T}{x} \eta_y^x (1 - \eta_y)^{T-x}$$

Step 2. The MAP estimation of Y . $P(Y = y | X = x)$ can be calculated according to the Bayesian formula:

$$P(Y = y | X = x) = \frac{P(X = x | Y = y) P(Y = y)}{\sum_{y'=0}^U P(X = x | Y = y') P(Y = y')},$$

and the MAP estimation \hat{y} is the one that maximizes $P(Y = y | X = x)$. Note that this requires the prior distribution of $P(Y = y)$, which will be discussed shortly.

Step 3. The conditional probability $P(Z = z | Y = y)$. It can be calculated iteratively on the number of segments. To be more specific, we use $P_i(Z = z | Y = y)$ to denote the probability that $Z = z$ when there are i segments. By definition, $P(Z = z | Y = y) = P_S(Z = z | Y = y)$. First, when there is only one segment, clearly,

$$P_1(Z = z | Y = y) = \begin{cases} 1 & z = y \\ 0 & \text{otherwise} \end{cases}$$

For notational simplicity, we use $\delta_t^{y,i}$ to denote the probability that among the y error bytes, t bytes are in one particular segment when there are i segments. Because the error bytes are randomly distributed,

$$\delta_t^{y,i} = \binom{y}{t} \left(\frac{1}{i}\right)^t \left(1 - \frac{1}{i}\right)^{y-t}$$

Given $P_i(Z = z | Y = y)$, $P_{i+1}(Z = z | Y = y)$ can be found by conditioning on the number of error bytes in a tagged segment. That is, we single out one segment, called the tagged segment, and check the number of error bytes in this segment. The event that $Z = z$ occurs when (1) the tagged segment has less than z errors while the maximum number of errors in a segment among the remaining segment is exactly z , or (2) the tagged segment has exactly z errors while the maximum number of errors in a segment among the remaining segments is no more than z . Therefore,

$$\begin{aligned} P_{i+1}(Z = z | Y = y) &= \sum_{t=0}^{z-1} \delta_t^{y,i} P_i(Z = z | Y = y - t) \\ &+ \delta_z^{y,i} \sum_{z'=0}^z P_i(Z = z' | Y = y - z) \end{aligned}$$

After getting $P(Z = z | Y = y)$ as well as \hat{y} , AMPS outputs \hat{z} such that $P(Z \leq \hat{z} | Y = \hat{y})$ is greater than a threshold, set to be 0.95 in our implementation. Finally, if there are multiple data packets in the frame, AMPS first estimates the total number of errors in the whole frame and divides it into

each packet proportional to the packet sizes, then estimates of maximum number of errors in a segment for each packet.

B. AMPS In Practice

1) Multiple Resolutions: For the estimation of Y , we introduce AMPS with multiple resolutions, because a particular K will always be best for one range of error ratios, but may over-amplify or under-amplify others. We use three types of samples with $K = 128$, $K = 32$, and $K = 10$, referred to as the type-0, type-1, and type-2 samples, respectively. The type-0, type-1, and type-2 samples are responsible for data error ratios in the range of $[0.001, 0.01]$, $[0.01, 0.03]$, and $[0.03, 0.2]$, respectively. Assuming the error values are random, they amplify the data error ratios roughly into sample error ratios of $[0.06, 0.36]$, $[0.14, 0.31]$, and $[0.13, 0.45]$, respectively, such that every range is sufficiently amplified to allow enough number of error samples to be drawn. The number of samples for type-0, type-1, and type-2 samples are 8, 16, and 40, respectively, which are determined by considering the required number of outputs in each range. The number of errors in a typical packet with error ratio range of $[0.001, 0.01]$, $[0.01, 0.03]$, and $[0.03, 0.2]$ are below 20 bytes, 60 bytes, and 400 bytes, respectively, and being able to output respectively 9, 17, and 41 different values should suffice. We basically let each estimator run in parallel. In cases when the three estimators give different estimates, we take a conservative approach and pick the largest estimate. Note that the total number of samples is 64, or only 8 bytes.

2) Table Lookup Implementation: To reduce the computation complexity, AMPS obtains the estimation with a constant time, simple table lookup after X is computed. Note that $P(Z = z | Y = y)$ can be precomputed such that the estimation of Z can be found by a table lookup given the MAP estimation of Y . Because $P(X = x | Y = y)$ can be precomputed, the MAP estimation of Y can also be obtained by a table lookup given the value of X , if the distribution of Y is fixed. However, in practice, the distribution of Y can vary depending on the wireless channel condition. We cope with it by computing tables for 100 representative distributions. As the size of the frame and the number of segments may vary, we also choose representative sizes of the frame and segment and compute corresponding tables. For any received frame under any channel condition, the table that is closest to its parameters is chosen. All such tables are computed for each value of K . The total size of the tables used by AMPS is about 1.5MB in the current implementation and can be further reduced after relaxing the accuracy requirements.

3) Acquiring $P(Y = y)$: AMPS requires the prior $P(Y = y)$, which should be estimated and selected among the representative distributions. We tested the Cisco Aironet 802.11a/b/g wireless cardbus adapter [18] and found that, interestingly, the byte error ratio distribution can be fitted very well in many cases by the truncated Pareto distribution:

$$P(x) = \frac{\alpha \gamma^\alpha x^{-\alpha-1}}{1 - (\gamma/\nu)^\alpha},$$

where γ and ν are the lower limit and the upper limit of x , respectively, and α controls the heaviness of the tail [15]. This may be due to the heavy tailed nature of the error burst in the

received packets. Therefore, the distribution can be described by only one parameter α , as γ and ν are known in practice, i.e., γ is a very small number and ν is a number close to 1. As 100 distributions are needed, we precompute tables for α from 0.02 to 2 at a step of 0.02, while $\gamma = 0.001$, $\nu = 0.999$.

To select a distribution, α should be estimated. We implemented the estimation method in [15], where we record the error ratios of the received frames, denoted as q_i for frame i . Suppose there are n recent samples. According to Equation (4) in [15], the estimation, denoted as $\tilde{\alpha}$, should satisfy

$$\frac{n}{\tilde{\alpha}} + \frac{n(\gamma/\nu)^{\tilde{\alpha}} \ln \gamma/\nu}{1 - (\gamma/\nu)^{\tilde{\alpha}}} + n \ln \gamma = \sum_{i=1}^n \ln q_i.$$

Given the set of error ratio values, we find $\tilde{\alpha}$ that results in a left side of the above equation closest to the measured $\sum_{i=1}^n \ln q_i$ on the right side of the equation.

C. Comparing with EEC

The other error ratio estimator we are aware of is EEC [13], which also uses the parity bit of multiple bits as a sample and uses multiple levels of samples. We independently arrived at a similar idea used in AMPS. Besides that, the approaches of AMPS and EEC profoundly differ. AMPS is designed according to the standard procedure in estimation theory, namely the MAP estimation, which is optimal for minimizing the average estimation error required by our application. EEC is based on a simple algorithm without using the knowledge of the prior distribution of error ratios. We note that the prior distribution can usually be measured at the receiver. Both AMPS and EEC have low computational overhead. However, for 1500-byte packets, the overhead of AMPS and EEC are 8 bytes and 36 bytes, respectively.

To compare the per-packet level estimation performance, we implemented EEC and ran simulations. As AMPS and EEC estimate byte and bit errors, respectively, we injected random byte and bit errors into 1500-byte and 1500-bit packet for them where the error ratio ranged from 0.01 to 0.2 at a step of 0.01. For each ratio the simulation ran for 10,000 times. As EEC assumes no prior knowledge, for a fair comparison, AMPS was also not given the prior, and ran based on the distribution of error ratio with the heaviest tail in which case AMPS approximates a maximum likelihood estimator. The performance is measured by the difference between the estimated and the real number of injected error bytes/bits. Fig. 4 suggests that AMPS outperforms EEC significantly. We believe the reason is that EEC may discard certain samples while such samples still carry much information. On the other hand, without the prior distribution, the maximum likelihood estimator is the optimal estimator. We show in Section VI the performance of AMPS in real-life experiments when the prior knowledge is available.

VI. EXPERIMENTAL RESULTS

We conducted experiments to evaluate the performance of Crelay. The compared schemes include

- *More*: the benchmark opportunistic routing protocol,
- *Srcr*: the benchmark traditional routing protocol, where nodes use the shortest path according to the ETX metric

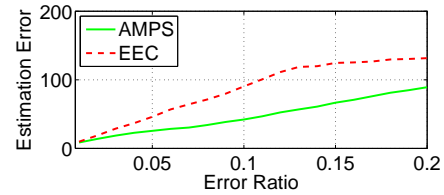


Fig. 4. Comparison of estimation error between AMPS and EEC.

to forward packet hop-by-hop without exploiting partial packets and overhearing.

We used the original implementation at [20] for *More* and used our own implementation for *Srcr*.

A. Implementation

We implemented a prototype of the Crelay protocol in around 5,000 lines of C++ code within the Click modular router [19] as a user space daemon. Packets are transmitted in broadcast frames, same as *More* [4]. The main addition to the Crelay protocol discussed in Section III is a simple mechanism to cope with interference. Basically, we allow a node v_A to send a polling message to another node v_B , if v_A has not heard any message from v_B for longer than a threshold, while v_A has many packets in state S_1 for v_B or has many packets in state S_2 for v_B that have been transmitted but have not been ACKed. Once heard the polling message, v_B transmits while others backoff for a time. The reason is that v_B 's transmission could be lost in collisions due to hidden terminal problems. This approach is adopted because RTS/CTS is not designed for broadcast packets in 802.11.

We also made two optimizations. First, we allow nodes to send parity bytes preemptively, i.e., parity bytes are sent along with data bytes in the first transmission attempt, if the link has non-zero error ratio. As a result, many packets with errors can be decoded at the receiver without requiring the overhead of feedback. In our current implementation, if the error ratio is $q > 0$, a node transmits bytes that can correct $\max\{0.02, \min\{0.05, q\}\}$ fractions of errors for the bytes it sends. Second, in AMPS, we set the minimum number of estimated errors in a segment to be 3, as this does not increase much overhead but can reduce the underestimation probability.

B. Testbed and Experiment Setup

We employed an 11-node wireless testbed. The wireless nodes are laptop computers with Cisco Aironet 802.11a/b/g cardbus adapter. In our experiments, the wireless cards ran on 802.11b/g channel 3 (2.427GHz) when there was little traffic (less than 3 beacon packets/sec) during the experiments. The transmission power was set to be 1dBm; in addition, aluminum foil was wrapped around the card to reduce the transmission/reception power to allow the experiments to be carried out within the confinements. The testbed setup, as well as two of machines used in the experiments, are shown in Fig. 5. With this set up, we were able to get 46 links where we consider a link exists if the erasure ratios of both directions are lower than 0.8. Among the links, the average RSSI was 6.98dB, the average erasure ratio was 0.271, and the average error ratio was 2%. We ran the Madwifi [16] driver in the monitor mode, which allows us to receive raw data frames for

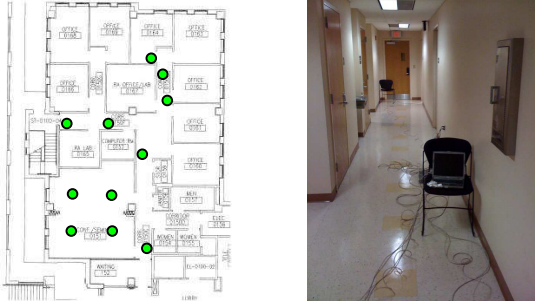


Fig. 5. The setup of the testbed.

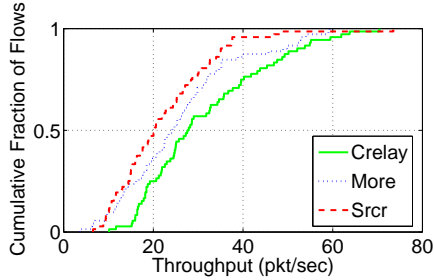


Fig. 6. The cumulative distribution of flow throughput.

exploiting partial packets. The MTU was set to be 2200 bytes and the data rate 1Mbps.

We selected 72 pairs of nodes in the network, between which Crelay found the path to be more than one hop. This selection was made because we are interested in the performance of multi-hop paths. Each pair is a flow, and we ran the experiment when only one flow was active. 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 [20], 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. The source node generates packets of size 1500 bytes every 10ms and the experiment runs for another 10 seconds.

C. Performance

We first show in Fig. 6 the cumulative distribution 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 has a significantly higher throughput than both More and Srcr. Fig. 7 reveals more interesting details, which shows the scattered plots of Crelay v.s. More and Crelay v.s. Srcr for each flow. In the scattered plot, a point the 45-degree line represents a flow in which two compared schemes have the same throughput. Fig. 7 shows that Crelay outperforms More in most flows and outperforms Srcr in almost all the flows. As a quantitative measure, for each flow, we define the throughput gain of scheme A over scheme B as $(\mu_A - \mu_B)/\mu_B$, where μ_A and μ_B are the throughput of scheme A and scheme B, respectively. We found that average throughput gain of Crelay over More is 36% and the average throughput gain of Crelay over Srcr is 52%.

One of the gains of Crelay is from exploiting partial packets. Fig. 8 shows the relation of gain and partial packet ratio for each flow, where the x axis is the percentage of partial packets

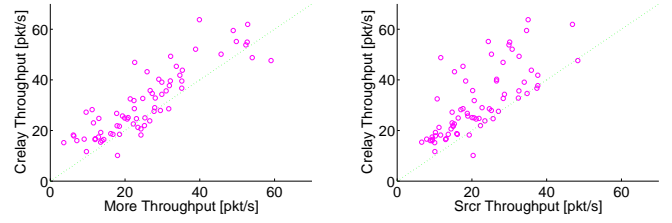


Fig. 7. The scattered plot comparison of the flow throughput.

received by the nodes on the packet forwarding path when running Crelay, and y axis is the throughput gain of Crelay over More. We can see that there is a positive correlation between the gain and the percentage of the partial packets.

Fig. 9 shows the average throughput gain of Crelay over More for paths of different lengths, where the path length is based on the path used by Crelay. The number of paths are 34, 26, 10 and 2 for path lengths of 2, 3, 4, and greater than 4, respectively. We can see that the gain is usually higher for longer paths, which may be because such paths usually have weaker channels and more opportunities to exploit partial packets.

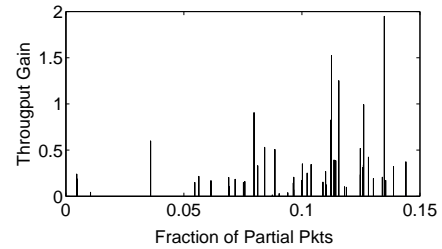


Fig. 8. The flow throughput gain and the fraction of partial packets.

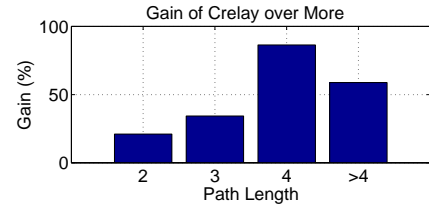


Fig. 9. Throughput gain for different path lengths.

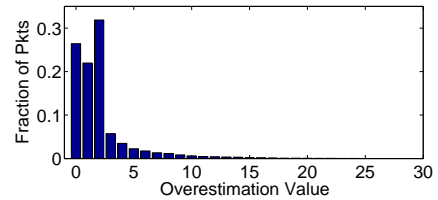


Fig. 10. The p.d.f. of overestimation of AMPS.

The performance of Crelay is largely dependent on AMPS. With our current choices of parameters, we found that for 23.7% of the times, AMPS underestimates the number of errors and more parity bytes have to be transmitted. However, even in such cases, usually most of the blocks are decoded and only a few blocks need more transmissions, because the numbers of errors in the records are different. For 3.89% of the time, AMPS underestimates the number of errors, but the available parity bytes, sent preemptively, are actually sufficient for correcting all the errors. For the rest of the cases

AMPS overestimates, and Fig.10 shows the probability density function of the number of overestimation. We can see that if overestimated, for more than 80.1% of the times, AMPS overestimates by no more than 3 bytes per codeword. It is possible to tune the parameters to achieve other underestimation/overestimation tradeoff.

VII. CONCLUSION

In this paper, we proposed Coded Relay (Crelay) for multi-hop wireless networks. With Crelay, nodes can exploit partial packets and overhearing for packet forwarding. One feature of Crelay is that nodes can often send some parity bytes to the next hop to recover the packet, which is significantly smaller than the size of the packet. We proposed and implemented the Crelay protocol in software. We studied the routing problem with Crelay and proposed a greedy algorithm for finding paths. We also designed an error ratio estimator, called AMPS, that can estimate the number of errors in a packet with good accuracy at very low overhead. We tested Crelay on an 11-node testbed, and the results show that Crelay is capable of achieving significant gain over existing protocols.

REFERENCES

- [1] 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.
- [2] A. Chakrabarti, A. de Baynast, A. Sabharwal, and B. Aazhang, "Low density parity check codes for the relay channel," *IEEE J. Select. Areas Commun.*, vol. 25, no. 2, pp. 280-291, Feb. 2007.
- [3] V. Venkatkumar, T. Wirth, T. Haustein, and E. Schulz, "Relaying in long term evolution: indoor full frequency reuse," in *Proc. of European Wireless*, Aarlborg, Denmark, May 2009.
- [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] S. Katti, D. Katabi, H. Balakrishnan and M. Medard, "Symbol-level network coding for wireless mesh networks," In *Proc. of ACM SIGCOMM*, 2008.
- [6] S. Biswas and R. Morris, "Opportunistic routing in multi-hop wireless networks," in *Proc. of ACM SIGCOMM*, 2005.
- [7] K. Lin, N. Kushman, and D. Katabi, "ZipTx: Harnessing partial packets in 802.11 networks," in *Proc. of ACM MOBICOM*, 2008.
- [8] R. Ahlswede, C. Ning, S.-Y.R. Li, and R.W. Yeung, "Network information flow," *IEEE Transactions on Information Theory*, vol. 46, no. 4, pp. 1204 - 1216, Jul. 2000.
- [9] S. Katti, S. Gollakota and D. Katabi, "Embracing wireless interference: analog network coding," in *Proc. of ACM SIGCOMM*, 2007.
- [10] 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.
- [11] K. Jamieson and H. Balakrishnan, "PPR: partial packet recovery for wireless networks," in *Proc. of ACM SIGCOMM*, 2007.
- [12] S. B. Wicker, *Error Control Coding for Digital Communication and Storage*, Prentice-Hall, NJ, 1995.
- [13] B. Chen, Z. Zhou, Y. Zhao, and H Yu, "Efficient error estimating coding: feasibility and applications," to appear in *Proc. of ACM SIGCOMM*, 2010.
- [14] 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.
- [15] I. B. Aban, M. M. Meerschaert and A. K. Panorska, "Parameter estimation for the truncated Pareto distribution," *Journal of the American Statistical Association*, vol. 101, no. 473, pp. 270-277, March 2006.
- [16] The MadWifi Project, <http://madwifi-project.org/>.
- [17] <http://www.ka9q.net/code/fec/>
- [18] Cisco Aironet 802.11a/b/g wireless cardbus adapter, <http://www.cisco.com/>.
- [19] The Click Modular Router, <http://read.cs.ucla.edu/click/>.
- [20] <http://people.csail.mit.edu/szym/more/README.html>.