# MIXIT: The Network Meets the Wireless Channel

Sachin Katti Dina Katabi
skatti@mit.edu dk@mit.edu

**Abstract**– The traditional contract between the network and the lower layers states that the network does routing and the lower layers deliver correct packets. In a wireless network, however, different nodes may hear most bits in a transmission, yet none of them receives the whole packet uncorrupted. The current approach imposes fate sharing on the bits, dropping a whole packet because of a few incorrect bits. In contrast, this paper proposes MIXIT, a new architecture that performs opportunistic routing on groups of correctly received symbols. We show using simulations driven with Software Radios measurements that MIXIT provides 4x throughput improvement over state-of-the-art opportunistic routing.

## 1 INTRODUCTION

Multi-hop wireless networks have been designed to mimic wired networks. Conventional protocols ignore the broadcast capability of a wireless network, treat it as a set of independent point-to-point links, and route through it by stringing a sequence of such links. Recently, however, the networking community has recognized the importance of embracing this underlying characteristic of the wireless medium. ExOR [1] and MORE [2] are opportunistic routing protocols that exploit wireless broadcast. They allow any node that hears a transmitted packet to participate in forwarding it toward its destination. This allows them to benefit from sporadic opportunistic receptions on long links, thereby providing significant throughput gains.

But these opportunistic protocols do not go far enough. They are hobbled by another holdover from the wired design: their insistence on using only correctly received packets. This works well in wired networks, where symbol[1] errors are rare, and almost all packet losses are due to congestion. In contrast, wireless networks show significant packet loss due to transient medium errors. All the symbols in a packet, however, do not share the same fate. Often only a few symbols are in error, while the rest are correct. It is wasteful to throw away the majority of the symbols that are correct due to a few incorrect ones. Significant performance gains can be obtained if nodes forward partial packets consisting of correctly received symbols, and drop the incorrect ones.

Our work is motivated by a simple key insight. There is a synergy between the ideas of opportunistic routing and partial packet forwarding. Opportunistic routing capitalizes on sporadic receptions over long links allowing a packet to make quick strides towards the destination [1]. But long links are inherently less reliable and likely to exhibit symbol errors. Yet, by insisting on receiving fully correct packets, current protocols are missing the bulk of their opportunities. Similarly, partial packet forwarding can also capitalize on opportunistic routing. When errors are high, no node receives a full packet correctly. But because of spatial diversity, each symbol will be received by some node correctly [16, 19], thus across a set of intermediate nodes the packet will be received correctly in aggregate. These intermediate nodes can then collaboratively route their opportunistically received partial packets to the destination, where they will be assembled into a complete packet.

This paper introduces MIXIT, an architecture that performs opportunistic routing on groups of correctly received symbols. MIXIT exploits the fact that the physical layer naturally computes a confidence measure for each decoded symbol [23, 8, 5]. This allows the routers to identify which symbols in a corrupt packet are likely correct and forward them. The core component of MIXIT is a novel symbol-level network code that also functions as a rateless error correcting code. This code addresses the two main challenges in forwarding partial packets. First, tracking the state of which node received which symbols to prevent duplicate transmissions can become a daunting coordination task. With symbol-level network codes, routers forward random linear combinations of their correctly received symbols, reducing the probability of sending duplicate information, and eliminating the need for coordination. Second, though the routers forward only symbols that were decoded with high confidence, there is a chance that a forwarded symbol is incorrect. Symbol-level network codes automatically function as rateless error correcting codes, providing an adaptive amount of redundancy to correct any erroneous symbols that seep through.

MIXIT embraces the basic characteristics of the wireless medium, presenting a unifying architecture that naturally exploits both space and time diversity. It disposes of artificial and self-defeating abstractions such as the point-to-point link and indivisible packets in favor of a more natural and useful abstraction. The new abstraction allows the network and the lower layers to collaborate on the common objectives of improving throughput and reliability. At the same time it maintains desirable properties such as being distributed, low-complexity, implementable and integrable

[1] A symbol is a unit of transmission. It is a sequence of bits that are mapped to a single real value using a modulation scheme, and then transmitted over the channel.

with the rest of the network stack.

We evaluate MIXIT using simulations that are driven with channel measurements from GNU Software Radios [4, 7]. Our preliminary results show that, on average, MIXIT provides 4x throughput improvement over packet-level opportunistic routing.

## 2 ILLUSTRATIVE EXAMPLE

Consider the scenario in Fig. 1, where the source $S$ wants to deliver two packets, $P_a$ and $P_b$, to the destination. Let the bit error rate (BER) be relatively high such that when the source $S$ broadcasts $P_a$ and $P_b$, the nodes in the network receive some symbols in errors. Fig. 1 illustrates such corrupted symbols using hashed cells. Due to spatial diversity [16, 19], however, the few corrupted symbols at nodes $R$ and $R'$ are unlikely to be in the same locations. However, with the current approach, the existence of a few erroneous symbols causes $R$ and $R'$ to ignore their receptions and ask the source to retransmit both packets. If the routers however were able to forward their correctly received symbols, the destination would receive a clean copy of every symbol, without any retransmissions.

To approach this ideal scenario, we first recognize that a node can identify which symbols are correctly received with high probability. Current physical layers (PHYs) compute a confidence value for each decoded symbol [8, 23]. Using this information the PHY can mark the received symbols as clean or faulty. We say a symbol is *clean* if its confidence value is above a threshold, $\gamma$, and *faulty* otherwise. We refer the reader to [8] for measurements of these confidence values, and note that as $\gamma$ increases, the probability that a clean symbol is corrupted becomes vanishingly small [8].

Though the routers can now filter out the faulty symbols and forward the clean ones, they may still waste a lot of capacity. Specifically, most symbols are received correctly by both $R$ and $R'$. Hence without additional measures, the routers will transmit the same symbols to the destination, wasting the wireless capacity. To avoid such waste, MIXIT employs symbol-level network coding, i.e., it makes the routers forward linear combinations of the clean symbols they received. Assuming $a_i$ and $b_i$ are the i$^{th}$ symbols in $P_a$ and $P_b$ respectively, router $R$ picks two random numbers $\alpha$ and $\beta$, and creates a coded packet $P_c$, where the i$^{th}$ symbol, $c_i$ is computed as follows:

$$c_i = \begin{cases} \alpha a_i + \beta b_i & \text{if } a_i \text{ and } b_i \text{ are clean symbols} \\ \alpha a_i & \text{if } a_i \text{ is clean and } b_i \text{ is faulty} \\ \beta b_i & \text{if } a_i \text{ is faulty and } b_i \text{ is clean.} \end{cases}$$

If both $a_i$ and $b_i$ are faulty, no symbol is sent in that position. Similarly, $R'$ generates a coded packet $P_d$ by picking two random values $\alpha'$ and $\beta'$ and applying the same logic in the above equation.

When $R$ and $R'$ broadcast their respective packets, $P_c$ and $P_d$, the destination receives corrupted versions where some symbols are incorrect, as shown in Fig. 1. Thus the destination has four partially corrupted receptions: $P_a$ and $P_b$, directly overheard from the source, contain many erroneous symbols; and $P_c$ and $P_d$ which contain a few erroneous symbols. For each symbol position $i$, the destination needs to decode two original symbols $a_i$ and $b_i$. As long as the destination receives two uncorrupted independent symbols in location $i$, it will be able to properly decode [6]. For example, consider the symbol position $i = 2$, the destination has received:

$$\begin{aligned} c_2 &= \alpha a_2 + \beta b_2 \\ d_2 &= \alpha' a_2. \end{aligned}$$

Given that the header of a coded packet contains the multipliers (e.g., $\alpha$ and $\beta$), the destination has two linear equations with two unknowns, $a_2$ and $b_2$, which are easily solvable. Once the destination has decoded all symbols correctly, it broadcasts an ACK, causing the routers to stop forwarding packets.

The rest of this paper extends MIXIT to general topologies, and ensures that the routers do not generate spurious transmissions and that the destination can detect residual errors and correct them.

## 3 RELATED WORK

This paper builds on prior work on opportunistic routing [1, 2], cooperative spatial diversity [16, 14], and wireless network coding [11, 9, 17, 21]. In particular, MIXIT's design borrows from MORE's, but the distinction between them is clear, as MORE operates on packets and cannot deal with faulty symbols. We also note that MIXIT is aligned with work on analog and physical layer network coding [10, 18], but it operates on symbols rather than signals, making it simple enough to fit within the current network stack.

We also build on prior work on soft information. Physical layers compute a confidence value on their symbol decoding decisions [19]. This is typically called soft information and its benefits have been widely discussed in information theory [13, 5, 20]. Recent works [8, 23] have proposed to extend the interface to the physical layer to expose this information to higher layers. MIXIT leverages this wider interface but uses it differently. The above proposals used the confidence information either to retransmit only low-confidence chunks in a corrupted packet [8] or make access points combine their confidence values over the wired Ethernet to reconstruct correct packets from erroneous receptions [23]. In contrast, MIXIT exploits the PHY confidence values to decide which symbols to forward and integrates it with opportunistic routing and network coding.

## 4 MIXIT'S ARCHITECTURE

MIXIT is designed for reliable file transfer over lossy stationary mesh networks. It assumes that the wireless cards are instrumented to deliver corrupted packets, where symbols with a confidence-level higher than $\gamma$ are marked as

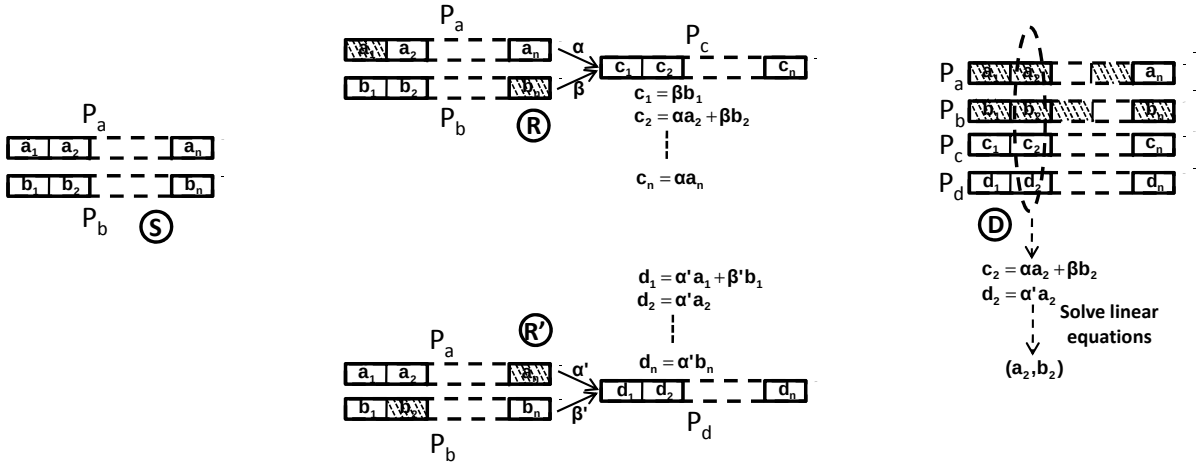


**Figure 1—Example:** The source broadcasts $P_a$ and $P_b$. The destination and the routers, $R$ and $R'$, receive corrupted versions of the packets. A hashed cell represents a corrupted symbol. If $R$ and $R'$ forward the correct symbols without coding, they generate spurious data and waste the capacity. With symbol-level network coding, the routers transmit linear combinations of the correct symbols, ensuring that they forward useful information to the destination.

| Term | Definition |
|---|---|
| Clean Symbol | A symbol that is received at the PHY with confidence higher than $\gamma$ |
| Faulty Symbol | an unclean symbol |
| Coded Symbol $s_j$ | Random linear combination of the clean symbols in the j$^{th}$ position in the received packets |
| Native Symbol | Uncoded Symbol |
| ETS of a link | The inverse of the symbol delivery probability on that link |
| Closer to destination | Node $X$ is closer than node $Y$ to the destination, if the shortest path from $X$ to the destination has a lower ETS than that from $Y$. |

**Table 1—Definitions used in the paper.**

clean, as defined in Table 1. The description below assumes that control information, i.e., the header, is correctly received. Since the header size is relative small in comparison with the packet size, it can be protected with a negligible amount of FEC.

**(a) The Source:** The source sends the file in batches of $K$ packets. When the 802.11 MAC is ready to send, the source creates a random linear combination of the $K$ native packets in the current batch and broadcasts the coded packet. Thus, the j$^{th}$ symbol in a coded packet, $s'_j$, is a linear combinations of the j$^{th}$ symbols in the $K$ native packets, i.e., $s'_j = \sum_i v_i s_{ji}$, where $s_{ji}$ is the j$^{th}$ symbols in the i$^{th}$ packet in the batch, and $v_i$ is a per-packet random multiplier. We call $\vec{v} = (v_1, \ldots, v_K)$ the *code vector* of the coded packet.

The source adds a MIXIT header to the coded packet and broadcasts it. The header describes which symbols were coded together. This description is trivial to articulate at the source because all symbols in a coded packet are generated using the packet's code vector, $\vec{v}$.

The header also contains the forwarders list. This is an ordered list that contains nodes closer to the destination than the source. While previous opportunistic routing protocols [1, 2] use the expected number of transmissions to deliver a packet (ETX) [3] as their distance metric, we use *the expected number of transmissions to correctly deliver a symbol (ETS)*. ETS is computed analogously to ETX. Nodes periodically ping each other with packets that contain known symbols, and compute the percentage of correctly delivered symbols. The inverse of this number is taken as the ETS of the link. The ETS of the path from node $X$ to the destination is the length of the shortest path from $X$ to the destination computed using the links' ETS as weights.

**(b) The Forwarders:** Nodes listen to all transmissions. When a node hears a packet, it checks whether it is in the forwarders list. If so, the node checks whether the packet contains new information, i.e., is innovative. Technically speaking, a packet is innovative if its code vector $\vec{v}$ is linearly independent from the vector of the packets the node has previously received from this batch. Checking for independence can be done using simple algebra (Gaussian Elimination [12]) over these short vectors. The node ignores non-innovative packets, and stores the innovative packets it receives from the current batch. Note that the symbols in the stored packets are marked as clean or faulty.

When the 802.11 MAC permits, the node may forward a packet. To do so the node creates a random linear combination of the clean symbols in the packets it has heard from the same batch and broadcasts it. The coded packet should contain linear combination only of clean symbols. Specifically, let $s_{ji}$ be the j$^{th}$ symbol in the i$^{th}$ packet that the forwarder has stored from this batch, and $v_i$ a per-packet random multiplier, the j$^{th}$ symbol in the forwarded packet is created as follows:

$$s'_j = \sum_i v_i s_{ji}, \quad \text{if } s_{ji} \text{ is a clean symbol.}$$

The MIXIT header in the forwarded packet has to articulate how each symbol is derived from the native symbols. This is more complex than in the case of the source because coding is performed only over clean symbols. Consider the simple example where the batch size $K = 2$ packets: $P_a$ and $P_b$. Say that our forwarder has received two coded packets $P_c = \alpha P_a + \beta P_b$ and $P_d = \alpha' P_a + \beta' P_b$. Now our forwarder picks two random numbers $v_1$ and $v_2$ and creates a

linear combination of the two packets it received.

$$P = v_1 P_c + v_2 P_d = (v_1\alpha + v_2\alpha')P_a + (v_1\beta + v_2\beta')P_b$$

Thus, the newly generated packet has a code vector $\vec{v} = (v_1\alpha + v_2\alpha', v_1\beta + v_2\beta')$. This vector would be sufficient to describe the whole packet if our forwarder received only clean symbols. But since some received symbols are faulty, we need a more detailed description of how individual symbols in the packet $P$ are derived from the native symbols.

Let us focus on the j$^{th}$ symbol position in packet $P$, called $s_j$. Depending on whether our forwarder has cleanly received the j$^{th}$ symbols in $P_c$ and $P_d$, called $c_j$ and $d_j$ respectively, the generated symbol $s_j$ might take one of four possible values.

$$s_j = \begin{cases} (v_1\alpha + v_2\alpha')a_j + (v_1\beta + v_2\beta')b_j & c_j \text{ and } d_j \text{ are clean} \\ v_1\alpha a_j + v_1\beta b_j & \text{only } c_j \text{ is clean} \\ v_2\alpha' a_j + v_2\beta' b_j & \text{only } d_j \text{ is clean} \\ 0 \times a_j + 0 \times b_j & c_j \text{ and } d_j \text{ are faulty} \end{cases} \quad (1)$$

The header has to articulate for each symbol in a transmitted packet which of the possible coding combinations was used to create it.

We exploit that wireless errors are bursty [16, 22], and use **run-length-encoding** to describe the encoding of the transmitted symbols in an efficient manner. Specifically, if the batch size is $K$, then there are $2^K$ possible coding combinations per symbol, which can be represented using $K$ bits. For example, let $K = 2$ and represent the possible coding states in Eq. 1 as 00, 01, 10, 11. Then the packet header will start by stating the four coefficients: $(v_1\alpha, v_2\alpha', v_1\beta, v_2\beta')$ This will be followed by the state of the various symbols which can be 00, 01, 10, or 11. Clearly, if each symbol can independently take a different state, the overhead will be excessive. On the other hand, if all symbols are clean, it is sufficient to state 00–1500, to indicate that all 1500 symbols in the packet are in state 00 (Assuming the symbol size is one byte).

In practice, one can control the header overhead and ensure that it stays small. On the one hand, wireless errors are known to be bursty [16, 22] and thus one would expect long runs of symbols that have the same state. On the other hand, our design actively ensures that the header stays within a bound. Note that the forwarder can always flip states 10, 01 (and even 11) to 00, if such flipping will create longer runs of the same state. Said differently the forwarder can decide to ignore some clean symbols to ensure the header has longer runs of the same state, and thus can be encoded efficiently. Note that as the forwarder ignores more clean symbols, the header becomes shorter and at the extreme MIXIT degrades to the current approach, which drops all packets that contain corrupted symbols.

**(c) The Destination:** The destination recovers the original symbols from the received coded symbols using standard decoding algorithms [15]. Once the original symbols are recovered for each symbol, the destination reassembles them into the original packets, and sends an ACK to the source to allow it to move to the next batch. ACKs are sent using best path routing, which is possible because MIXIT uses standard 802.11 and co-exists with shortest path routing. ACKs are also given priority over data packets at every node and protected using FEC.

## 5 MAXIMIZING THROUGHPUT

Naively broadcasting coded clean symbols does not increase throughput. There will be a large overlap between the packets heard by the routers. Whenever there is an overlap, it is more efficient to have the router closer to the destination forward the common information because that requires fewer transmissions. We want a forwarding strategy that considers such issues and maximizes the throughput.

We observe that we can leverage prior work on packet-based opportunistic routing because, similarly to MIXIT, these protocols have to resolve information overlap, albeit at the packet level. With simple modifications, we can adopt MORE's [2] routing algorithm to operate on clean symbols. In particular, we replace the ETX metric used in MORE with the ETS metric described in §4, the packet loss probabilities with the symbol loss probabilities at a particular $\gamma$, which we can compute from the same probes we used to compute the ETS metric. We describe these modifications in more details below. We note however that this adaptation of the MORE's algorithm to symbol-based routing allows MIXIT to inherit many desirable properties such as: 1) it is distributed; 2) it has low complexity that is comparable to current wireless routing (it is $O(n^2)$ where $n$ is the number of nodes); 3) it works with the 802.11 MAC.

### 5.1 Forwarding Algorithm

Intuitively, our algorithm works by ensuring that a common piece of information is forwarded by the node closer to its destination in ETS metric. Formally, let $n$ be the number of nodes in the network. For any two nodes, $i$ and $j$, let $i > j$ denote that node $i$ is farther from the destination than node $j$ in the ETS metric. Given a threshold $\gamma$, let $p_{ij,\gamma}$ be the probability that node $j$ fails to correctly receive a symbol that $i$ transmits. Last, let $z_{i,\gamma}$ be the expected number of transmissions that forwarder $i$ must make to forward one clean symbol from the source, $s$, to the destination, $d$, given a particular threshold $\gamma$. In the following, we assume that wireless receptions at different nodes are independent, an assumption that is supported by prior measurements [16].

Let us first calculate the expected number of transmissions that a forwarder $j$ must make to deliver a clean symbol from source, $s$, to destination, $d$. The expected number of symbols that $j$ receives from nodes with higher distance is $\sum_{i>j} z_{i,\gamma}(1 - p_{ij,\gamma})$. For each clean symbol $j$ receives, $j$ should forward it only if no node with lower distance gets that symbol. This happens with probability $\prod_{k<j} p_{ik,\gamma}$.

Thus, in expectation, the number of symbols that $j$ must forward, denoted by $L_j$, is:

$$L_j = \sum_{i>j} (z_{i,\gamma}(1 - p_{ij,\gamma}) \prod_{k<j} p_{ik,\gamma}). \quad (2)$$

Note that $L_s = 1$ because the source generates the symbol.

Now, consider the expected number of transmissions a node $j$ must make. $j$ should transmit each symbol until at least one node closer to the destination receives it. Thus, the number of transmissions that $j$ makes for each symbol it forwards is a geometric random variable with success probability $(1 - \prod_{k<j} p_{jk,\gamma})$. This is the probability that some node with lower distance than $j$ cleanly receives the symbol. Knowing the number of symbols that $j$ has to forward from Eq. (2), the expected number of transmissions that $j$ must make is:

$$z_{j,\gamma} = \frac{L_j}{(1 - \prod_{k<j} p_{jk,\gamma})}. \quad (3)$$

Given the similarity in the derivations with MORE, we leverage the argument in [2], which shows that the number of transmissions made by each node, the $z_j$'s, can be computed in $O(n^2)$. Further the routing algorithm is distributed and follows a link-state algorithm, where links weights are set to the symbol loss probabilities, the $p_{ij,\gamma}$'s. These probabilities can be computed in a way similar to how current routing algorithms compute packet loss probabilities, namely using pairwise probes.

Furthermore, the above algorithm can be easily integrated with the 802.11 MAC. In MIXIT, transmissions are triggered by packet receptions and performed only when the 802.11 MAC permits. For each node, we define a credit counter `credit_counter`. When a node receives a packet, the counter is incremented enough to allow the node to forward the clean symbols it received. In particular, for each symbol sent from source to destination, node $i$ receives $\sum_{j>i}(1 - p_{ji,\gamma})z_j$, where $z_j$ is the number of transmissions made by node $j$ and $p_{ji,\gamma}$ is the symbol loss probability from $j$ to $i$. Thus, the TX_credit of node $i$ is:

$$\text{TX\_credit}_i = \frac{z_i}{\sum_{j>i} z_j(1 - p_{ji,\gamma})}. \quad (4)$$

When the 802.11 MAC allows the node to transmit, the node checks whether the credits in the counter are more than the packet size measured in symbols. If yes, the node transmits a packet and decrements the counter by the size of the packet; else, the node waits until it has enough credits.

### 5.2 From Clean To Correct Symbols

Up to now we have ignored the difference between a clean and a correct symbol and focused on delivering clean symbols to the destination. Yet, for any choice of the confidence threshold, $\gamma$, there is a chance that a clean symbol is actually corrupted. The destination however needs to recover a correct copy of the source's native symbols. This problem can be addressed by noting that the coded symbols in MIXIT are simply linear error correcting codes [15].

Specifically, let $\epsilon(\gamma)$ be the symbol error rate at the destination, for a confidence threshold, $\gamma$. Thus, a fraction $\epsilon(\gamma)$ of the clean symbols that the destination receives are incorrect. These symbols can be corrected with added redundancy. Said differently, for a batch of $K$ packets, if all clean symbols are correct, then the destination can decode the $K$ native symbols in position $j$ in the packets after receiving $K$ linearly independent coded symbols for that position. If the received clean symbols are potentially corrupted, then $K$ coded symbols are not enough for decoding; the destination needs added redundancy. In particular, to correct for an error rate of $\epsilon(\gamma)$ it is well-known that the destination needs an extra $2\epsilon(\gamma)$ of coded symbols [15].

Thus, in the above algorithm, we need to add the redundancy required to compensate for errors. We have to replace every $z_j$ with $z_j(1+2\epsilon(\gamma))$. Note however that the whole objective of the algorithm is to compute the TX_credit in Eq.4. Multiplying the $z$'s in Eq.4 by $(1+2\epsilon(\gamma))$ however does not change the equation because this term cancels out. Thus, the nodes need not know $\epsilon(\gamma)$ to perform their forwarding computation.

Note that the destination's decoding algorithm is different from standard decoding of network codes [2]. The destination treats the coded symbols for each position as a linear block-error-correcting code such as a Reed-Solomon code [15]. It therefore uses decoding algorithms which can recover from errors in the original symbols, standard network coding algorithms cannot recover from errors. These decoding algorithms can correct half as many errors as redundancy added, which is the best possible. Thus if among the $K$ symbols at a particular position, one of them is corrupted, it is sufficient to obtain $K + 2$ coded symbols to correct that error [15].

One may wonder about the benefits of MIXIT if at the end it just functions as an error correcting code. Indeed this is the beauty of the approach. The nodes need not estimate the error probability or how much FEC to add. MIXIT functions as a distributed rateless error correcting code, i.e., the destination keeps receiving coded symbols until it can decode (which it can verify by adding a CRC to each native packet). MIXIT, however, differs from prior work on rateless codes because it is distributed and naturally integrates and exploits the spatial diversity in a wireless network.

## 6 RESULTS

We present preliminary results that illustrate MIXIT's throughput gains and explore when such an approach would be useful.

Our evaluation uses a combination of simulation and software radio experiments. Each simulation has 20 nodes randomly placed in an area of $100\times100$ m$^2$. The signal in the wireless channel is attenuated proportionally to the cube of the distance from sender to receiver. The simula-

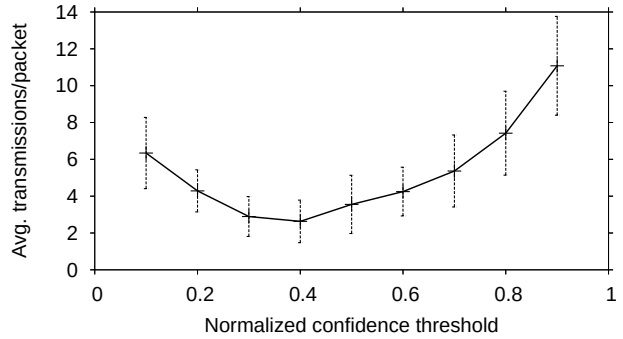


**Figure 2**—Effect of the confidence threshold on the average number of transmissions to deliver a packet-length of symbols to the destination.

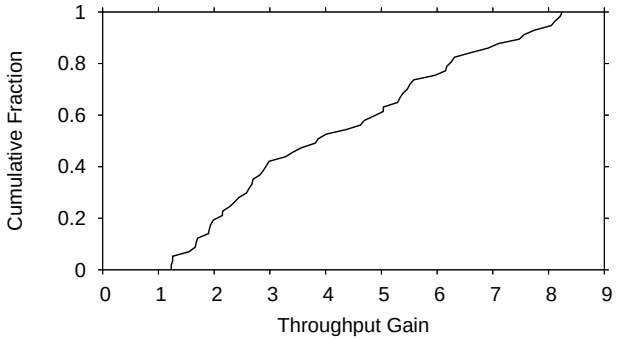


**Figure 3**—MIXIT's throughput gains relative to packet-based opportunistic routing

tor needs to compute two probability distributions. First, for each wireless channel we need the distribution of the per-symbol confidence values so that we can sample that distribution as we simulate the transmission of symbols. Second, we need to compute what is the probability of a symbol being in error, given a particular confidence value. We compute both distributions empirically from GNURadio experiments. We transmit DBPSK modulated packets between a GNURadio sender-receiver pair, and take the output of the matched filter on the receiver as the per-symbol confidence, as proposed in [23]. For each channel in a simulated network, we scale the distribution according to the ratio of the simulated attenuation and the attenuation of the actual GNURadio channel. Furthermore, we use the same experiments to compute the symbol error rate as a function of the confidence threshold, $\gamma$ and feed that function to the simulator. The simulator then simulates the outcome of the algorithm in §5 for each network instance.

**(a) Effect of the Confidence Threshold**

The confidence threshold plays a critical role in balancing the gains of spatial diversity with the potential of marking a corrupted symbol as clean. We quantify this effect in Fig. 2 which plots the average number of transmissions to reliably deliver a packet-length of correct symbols to its destination, as a function of the confidence threshold. We simulate 200 random topologies of 20 nodes. For each network, we pick a random source-destination pair and compute the average number of transmissions as a function of the confidence threshold. This number includes the additional transmissions required to correct any corrupt symbols at the destination.

The figure shows that there is an optimal confidence threshold. Below that value (i.e., less than $< 0.4$), the per-symbol confidence is too low, and hence a significant number of forwarded symbols may be corrupted and have to be corrected again by retransmissions. When the threshold is high on the other hand, many correct symbols are unnecessarily dropped, reducing the gains of spatial diversity.

The relationship between the confidence threshold and the average number of transmissions required is convex and has an optimal value, but the curve is flat near the optimal threshold. Further, even though the simulations were performed with randomly picked source-destination pairs, the optimal threshold hovers around $0.3 - 0.5$. This suggests that one can pick a threshold offline for a network. The slight loss of optimality is negligible given MIXIT's large throughput gains, reported below.

**(b) Comparing with Traditional Opportunistic Routing**

How does MIXIT compare with packet-based opportunistic routing protocols like ExoR [1] and MORE [2]? Given the simple capabilities of our simulator, we cannot compare the details of these protocols with MIXIT. Instead, we compare the MORE algorithm described in Section 5.1 of [2] with the MIXIT algorithm in §4, both under ideal estimates of the error probabilities. We compute the **Throughput Gain** as the ratio of the average number of transmissions to deliver a fully-correct packet from source to destination in MORE and MIXIT. This does not account for the fact that MIXIT's header is larger than that of MORE's, and thus should be taken as an upper bound on the throughput gain.

Fig. 3 plots the CDF of throughput gain of MIXIT relative to MORE. The simulation is conducted over 200 random topologies. For each topology the optimal confidence threshold is computed, then the average number of transmissions required to ship a full packet of 1500 symbols from the source to the destination with MIXIT is compared with MORE. The figure shows that MIXIT provides a median gain of 4x when compared to packet-based opportunistic routing.

**(c) When Does MIXIT Help?**

Fig. 3 shows a range of throughput gains that varies from 1.2x to 8x. We would like to understand the differences between topologies with low and high gains. Isolating the networks with smaller improvements, we have discovered that they shared one common characteristic. The pairwise wireless channels are roughly bimodal, i.e., there are very good links where the large fraction of the symbols in a packet were received error-free and there were bad links where most of the symbols were incorrect. MIXIT derives its gains from exploiting packets received with errors, but in such networks these opportunities are relatively few. Thus, the gains are smaller. In contrast for the networks with the largest gains, the channels show a wide variation over the entire spectrum; from good links to bad links. MIXIT there-

fore can salvage many correct symbols that would otherwise be dropped, achieving larger gains.

## 7 DISCUSSION

MIXIT changes how protocol designers think about wireless network architecture. Instead of treating wireless networks as a set of links where nodes have to code for each channel separately, MIXIT treats the entire wireless network like a single logical link and creates an adaptive error-correcting code on the fly. Further, MIXIT increases network throughput by building on the inherit characteristics of the wireless medium; it embraces wireless broadcast and exploits both space and time diversities. While MIXIT makes clear departures from conventional network design, it maintains its desirable properties such as being distributed, of low-complexity, implementable, and integrable with the rest of the network stack.

Going forward, we plan to implement MIXIT and use empirical measurements to design efficient run-length encoding schemes that capture the error structure with little overhead. We are also investigating if we can introduce sparsity into our network codes, so that decoding has lower complexity. Finally, we are looking at modifying the routing algorithm to react to congestion and perform load balancing among multiple flows.

## 8 ACKNOWLEDGMENTS

We thank Nate Kushman, Hariharan Rahul and the reviewers for their insightful comments. This work is supported by DARPA CBMANET and an Intel gift. The opinions and findings in this paper are those of the authors and do not necessarily reflect the views of DARPA or Intel.

## REFERENCES

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

[2] S. Chachulski, M. Jennings, S. Katti, and D. Katabi. Trading structure for randomness in wireless opportunistic routing. In *Proc. of ACM SIGCOMM 2007, Kyoto, Japan*.

[3] D. S. J. De Couto, D. Aguayo, J. Bicket, and R. Morris. A high-throughput path metric for multi-hop wireless routing. In *ACM MobiCom '03*, San Diego, California, September 2003.

[4] G. FSF. Gnu radio - gnu fsf project. http://www.gnu.org/software/gnuradio.

[5] J. Hagenauer and P. Hoecher. A Viterbi Algorithm with Soft-Decision Outputs and its Applications. In *IEEE GLOBECOM*, 1989.

[6] T. Ho, R. Koetter, M. Médard, D. Karger, and M. Effros. The Benefits of Coding over Routing in a Randomized Setting. In *Proc. of ISIT, 2003, Yokohoma, Japan*.

[7] E. Inc. Universal software radio peripheral. http://ettus.com.

[8] K. Jamieson and H. Balakrishnan. Ppr: Partial packet recovery for wireless networks. In *Proc. of ACM SIGCOMM 2007, Kyoto, Japan*.

[9] A. Kamra, V. Misra, J. Feldman, and D. Rubenstein. Growth Codes: Maximizing Sensor Network Data Persistence. In *Proc. of ACM SIGCOMM 2006, Pisa, Italy*.

[10] S. Katti, S. Gollakota, and D. Katabi. Analog network coding. In *Proceedings of ACM SIGCOMM 2007, Kyoto, Japan*.

[11] S. Katti, H. Rahul, D. Katabi, W. H. M. Médard, and J. Crowcroft. XORs in the Air: Practical Wireless Network Coding. In *Proc. of ACM SIGCOMM 2006, Pisa, Italy*.

[12] R. Koetter and M. Médard. An algebraic approach to network coding. *IEEE/ACM Transactions on Networking, Volume 11, Issue 5, Oct. 2003, Page(s):782 - 795*.

[13] F. J. J. R. L. Bahl, J. Cocke. Optimal decoding of linear codes for minimizing symbol error rate. *Information Theory, IEEE Transactions on*, 20(2):2020–2040, 1974.

[14] J. N. Laneman, D. N. C. Tse, and G. W. Wornell. Cooperative diversity in wireless networks: Efficient protocols and outage behavior. *IEEE Trans. on Inform. Theory, Volume 50, Issue 12, Dec. 2004 Page(s):3062 - 3080*.

[15] D. McKay. *Information Theory, Inference and Learning Algorithms*. Cambridge University Press, 2003.

[16] A. Miu, H. Balakrishnan, and C. E. Koksal. Improving loss resilience with multi-radio diversity in wireless networks. In *MobiCom '05: Proceedings of the 11th annual international conference on Mobile computing and networking*, 2005.

[17] J. S. Park, M. Gerla, D. S. Lun, Y. Yi, and M. Medard. Codecast: A network-coding based ad hoc multicast protocol. *IEEE Wireless Communications Magazine*, 2006.

[18] S. Zhang, S. Liew, and P. Lam. Physical layer network coding. In *Proc. of ACM MOBICOM 2006, Los Angeles, USA*.

[19] D. Tse and P. Vishwanath. *Fundamentals of Wireless Communications*. Cambridge University Press, 2005.

[20] M. Wang, X. Weimin, and T. Brown. Soft Decision Metric Generation for QAM with Channel Estimation Error. *IEEE Transactions on Communications*, 50(7):1058 – 1061, 2002.

[21] J. Widmer and J.-Y. L. Boudec. Network Coding for Efficient Communication in Extreme Networks. In *Proc. of SIGCOMM WDTN 2005, Philadelphia, USA*.

[22] A. Willig, M. Kubisch, C. Hoene, and A. Wolisz. Measurements of a wireless link in an industrial environment using an ieee 802.11-compliant physical layer. *IEEE Transaction on Industrial Electronics*, 49(6), 2002.

[23] G. Woo, P. Kheradpour, and D. Katabi. Beyond the bits: Cooperative packet recovery using phy information. In *Proc. of ACM MobiCom 2007, Montreal, Canada*.