Computer Science and Artificial Intelligence Laboratory

Technical Report

MIT-CSAIL-TR-2007-014 February 23, 2007

Trading Structure for Randomness in Wireless Opportunistic Routing

Szymon Chachulski, Michael Jennings, Sachin Katti, and Dina Katabi

massachusetts institute of technology, cambridge, ma 02139 usa — www.csail.mit.edu

Trading Structure for Randomness in Wireless Opportunistic Routing

Szymon Chachulski szym@mit.edu
Michael Jennings mvj@mit.edu
Sachin Katti skatti@mit.edu
Dina Katabi dk@mit.edu

ABSTRACT

Opportunistic routing is a recent technique that achieves high throughput in the face of lossy wireless links. The current opportunistic routing protocol, ExOR, ties the MAC with routing, imposing a strict schedule on routers' access to the medium. Although the scheduler delivers opportunistic gains, it misses some of the inherent features of the 802.11 MAC. For example, it prevents spatial reuse and thus may underutilize the wireless medium. It also eliminates the layering abstraction, making the protocol less amenable to extensions of alternate traffic type such as multicast.

This paper presents MORE, a MAC-independent opportunistic routing protocol. MORE randomly mixes packets before forwarding them. This randomness ensures that routers that hear the same transmission do not forward the same packets. Thus, MORE needs no special scheduler to coordinate routers and can run directly on top of 802.11. Experimental results from a 20-node wireless testbed show that MORE's average unicast throughput is 20% higher than ExOR, and the gains rise to 50% over ExOR when there is a chance of spatial reuse. For multicast, MORE's gains increase with the number of destinations, and are 35-200% greater than ExOR.

1 INTRODUCTION

Wireless mesh networks are increasingly used for providing cheap Internet access everywhere [7, 4, 32]. City-wide WiFi networks, however, need to deal with poor link quality caused by urban structures and the many interferers including local WLANs. For example, half of the *operational* links in Roofnet [4] have a loss probability higher than 30%. Opportunistic routing has recently emerged as a mechanism for obtaining high throughput even when links are lossy [10]. Traditional routing chooses the nexthop before transmitting a packet; but, when link quality is poor, the probability the chosen nexthop receives the packet is low. In contrast, opportunistic routing allows *any* node that overhears the transmission and is closer to the destination to participate in forwarding the packet. Biswas and Morris have demonstrated that this more relaxed choice of nexthop significantly increases the throughput. They proposed the ExOR protocol as a means to achieve these gains [10].

Opportunistic routing, however, introduces a difficult challenge. Multiple nodes may hear a packet broadcast and unnecessarily forward the same packet. ExOR deals with this issue by tying the MAC to the routing, imposing a strict scheduler on routers' access to the medium. The scheduler goes in rounds. Forwarders transmit in order, and only one forwarder is allowed to transmit at any given time. The others listen to learn which packets were overheard by each node. Although the medium access scheduler delivers opportunistic throughput gains, it does so at the cost of losing some of the desirable features of the current 802.11 MAC. In particular, the scheduler prevents the forwarders from exploiting spatial reuse, even when multiple packets can be simultaneously received by their corresponding receivers. Additionally, this highly structured approach to medium access makes the protocol hard to extend to alternate traffic types, particularly multicast, which is becoming increasing common with content distribution applications [1] and video broadcast [3, 2].

In contrast to ExOR's highly structured scheduler, this paper addresses the above challenge with randomness. We introduce MORE, MAC-independent Opportunistic Routing & Encoding. MORE randomly mixes packets before forwarding them. This ensures that routers that hear the same transmissions do not forward spurious packets. Indeed, the probability that such randomly coded packets are the same is proven to be exponentially low [15]. As a result, MORE does not need a special scheduler; it runs directly on top of 802.11.

The main contribution of MORE is its ability to deliver the opportunistic routing gains while maintaining the clean architectural abstraction between the routing and MAC layers. MORE is MAC-independent, and thus can enjoy the basic features available to today's MAC. Specifically, it achieves better unicast throughput by exploiting the spatial reuse available with 802.11. Further, the clean separation between the layers makes MORE easily extendable to multicast traffic.

We evaluate MORE in a 20-node indoor wireless testbed. Our implementation is in Linux and uses the Click toolkit [25] and the Roofnet software package [4]. Our results reveal the following findings.

- In our testbed, MORE's average unicast throughput is 20% higher than ExOR. For 4-hop flows where the last hop can exploit spatial reuse, MORE's throughput is 50% higher than ExOR's. For multicast traffic,

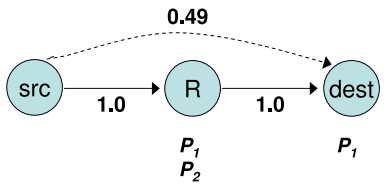


Figure 1—Unicast Example. The source sends 2 packets. The destination overhears p_2, while R receives both. R needs to forward just one packet but, without node-coordination, it may forward p_2, which is already known to the destination. With network coding, however, R does not need to know which packet the destination misses. R just sends the sum of the 2 packets $p_1 + p_2$. This coded packet allows the destination to retrieve the packet it misses independently of its identity. Once the destination receives the whole transfer (p_1 and p_2), it acks the transfer causing R to stop transmitting.

MORE's gains increase with the number of destinations; For 2-4 destinations, MORE's throughput is 35-200% higher than ExOR's.

- In comparison with traditional routing, the average gain in the throughput of a MORE flow is 70%, and the maximum throughput gain exceeds 10x.
- Finally, coding is not a deployment hurdle for mesh wireless networks. Our implementation can sustain a throughput of 44 Mb/s on low-end machines with Celeron 800MHz CPU and 128KiB of cache.

2 MOTIVATING EXAMPLES

MORE's design builds on the theory of network coding [5, 26, 15]. In this section, we use two toy examples to explain the intuition underlying our approach and illustrate the synergy between opportunistic routing and network coding.

The Unicast Case: Consider the scenario in Fig. 1. Traditional routing predetermines the path before transmission. It sends traffic along the path "*src*→*R*→*dest*", which has the highest delivery probability. However, wireless is a broadcast medium. When a node transmits, there is always a chance that a node closer than the chosen nexthop to the destination overhears the packet. For example, assume the source sends 2 packets, p_1 and p_2. The nexthop, R, receives both, and the destination happens to overhear p_2. It would be a waste to have node R forward p_2 again to the destination. This observation has been noted in [10] and used to develop ExOR, an opportunistic routing protocol for mesh wireless networks.

ExOR, however, requires node coordination, which is hard to achieve in a large network. Consider again the example in the previous paragraph. R should forward only packet p_1 because the second packet has already been received by the destination; but, without consulting with the destination, R has no way of knowing which packet to transmit. The problem becomes harder in larger networks, where many nodes hear a transmitted packet. Opportunistic routing allows these nodes to participate in forwarding the heard packets. Without coordination, however, multiple nodes may unnecessarily forward the same packets, creating spurious transmissions. To deal with this issue, ExOR imposes a special scheduler on top of 802.11. The scheduler goes in rounds and reserves the medium for a single forwarder at any one time. The rest of the nodes listen to learn the packets overheard by each node. Due to this strict schedule, nodes farther away from the destination (which could potentially have transmitted at the same time as nodes close to the destination due to spatial reuse), cannot, since they have to wait for the nodes close to the destination to finish transmitting. Hence the scheduler has the side effect of preventing a flow from exploiting spatial reuse.

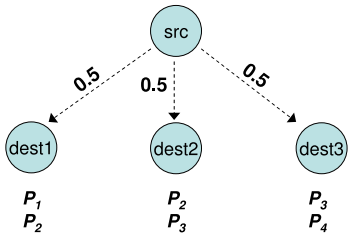


Figure 2—Multicast Example. Instead of retransmitting all four packets, the source can transmit two linear combinations, e.g., $p_1 + p_2 + p_3 + p_4$ and $p_1 + 2p_2 + 3p_3 + 4p_4$. These two coded packets allow all three destinations to retrieve the four original packets, saving the source 2 transmissions.

Network coding offers an elegant solution to the above problem. In our example, the destination has overheard one of the transmitted packets, p_2, but node R is unaware of this fortunate reception. With network coding, node R naturally forwards linear combinations of the received packets. For example, R can send the sum $p_1 + p_2$. The destination retrieves the packet p_1 it misses by subtracting from the sum and acks the whole transfer. Thus, R need not know which packet the destination has overheard.

Indeed, the above works if R sends any random linear combination of the two packets instead of the sum. Thus, one can generalize the above approach. The source broadcasts its packets. Routers create random linear combinations of the packets they hear (i.e., $c_1p_1 + \ldots + c_np_n$, where c_i is a random coefficient). The destination sends an ack along the reverse path once it receives the whole transfer. This approach does not require node coordination and preserves spatial reuse.

The Multicast Case: Our second example illustrates the synergy between network coding and multicast. In Fig. 2, the source multicasts 4 packets to three destinations. Wireless receptions at different nodes are known to be highly independent [31, 29]. Assume that each destination receives the packets indicated in the figure–i.e., the first destination receives p_1 and p_2, the second destination receives p_2 and p_3, and the last destination receives p_3 and p_4. Note that each of the four packets is lost by some destination.

Without coding, the sender has to retransmit the union of all lost packets, i.e., the sender needs to retransmit all four packets. In contrast, with network coding, it is sufficient to transmit 2 randomly coded packets. For example, the sender may send $p'_1 = p_1 + p_2 + p_3 + p_4$ and $p'_2 = p_1 + 2p_2 + 3p_3 + 4p_4$. Despite that they lost different packets, all three destinations can retrieve the four original packets using these two coded packets. For example, the first destination, which has received p'_1, p'_2 and p_1, p_2, retrieves all four original packets by inverting the matrix of coefficients, and multiplying it with the packets it received, as follows:

$$\begin{pmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}^{-1} \begin{pmatrix} p'_1 \\ p'_2 \\ p_1 \\ p_2 \end{pmatrix}.$$

Thus, in this simple example, network coding has reduced the needed retransmissions from 4 packets to 2, improving the overall throughput.

The Challenges: To build a practical protocol that delivers the above benefits, we need to address a few challenges.
(a) How Many Packets to Send? In traditional best path routing, a node keeps sending a packet until the nexthop receives it or until it gives up. With opportunistic routing however, there is no particular nexthop; all nodes closer to the destination than the current transmitter can participate in forwarding the packet. How many transmissions are sufficient to ensure that at least one node closer to the destination has received the packet?
(b) Stop and Purge? With network coding, routers send linear combinations of the packets. Once the destination has heard enough such coded packets, it decodes and retrieves the file. We need to stop the sender as soon as the destination has received the transfer and purge the related data from the forwarders.
(c) Efficient Coding? Network coding optimizes for better utilization of the wireless medium, but coding requires the routers to multiply and add the data bytes in the packets. We need efficient coding and decoding strategies to prevent routers' CPU from becoming a bottleneck.

3 Related Work

We begin with a brief survey of prior work on opportunistic routing and a summary of network coding.

3.1 Opportunistic Routing & Wireless Diversity

Opportunistic routing has been introduced by Biswas and Morris, whose paper explains the potential throughput increase and proposes the ExOR protocol as a means to achieve it [10]. Opportunistic routing belongs to a general class of wireless algorithms that exploit multi-user diversity. These techniques use receptions at multiple nodes to increase wireless throughput. They either optimize the choice of forwarder from those nodes that received a transmission [10], or combine the bits received at different nodes to correct for wireless errors [29], or allow all nodes that overheard a transmission to simultaneously forward the signal acting as a multi-antenna system [16]. Our work builds on this foundation but adopts a fundamentally different approach; it combines random network coding with opportunistic routing to address its current limitations. The resulting protocol is practical, allows spatial reuse, and supports both unicast and multicast traffic.

3.2 Network Coding

Work on network coding has started with a pioneering paper by Ahlswede et al. that establishes the value of coding in the routers and provides theoretical bounds on the capacity of such networks [5]. The combination of [26, 24, 18] shows that, for multicast traffic, linear codes achieve the maximum capacity bounds, and coding and decoding can be done in polynomial time. Additionally, Ho et al. show that the above is true even when the routers pick random coefficients [15]. Researchers have extended the above results to a variety of areas including content distribution [14], secrecy [11, 17], and distributed storage [19].

Of particular relevance is prior work on wireless network coding [27, 22, 23]. This work can be divided into three classes. The first is theoretical; it extends known information theory bounds from wired to wireless networks [27, 17]. The second is simulation-based; it designs and evaluates network coding protocols using simulations [30, 33]. The third is implementation-based; it uses implementation and testbed experiments to demonstrate achievable throughput gains for sensors and mesh networks [23, 21]. MORE belongs to this latter class.

This paper builds on the above foundational work described above, but differs from it in two main ways. First, the design of MORE including the choice of forwarders, the heuristic of when a node should forward, the traffic purging scheme, and the mechanisms used for achieving high bit-rate via efficient coding, is new. Second, our experimental results reveal important new findings, including how ExOR compares to a network coding protocol under a variety of settings, and whether there are practical benefits that support the use of random wireless network coding.

4 MORE In a Nutshell

MORE is a routing protocol for stationary wireless meshes, such as Roofnet [4] and community wireless networks [32, 6]. Nodes in these networks are PCs with ample CPU and memory.

MORE sits below IP and above the 802.11 MAC. It provides *reliable* file transfer. It is particularly suitable for delivering files of medium to large size (i.e., 8 or more packets). For shorter files or control packets, we use stan-

Term	Definition
Native Packet	Uncoded packet
Coded Packet	Random linear combination of native or coded packets
Code Vector of a Coded Packet	The vector of co-efficients that describes how to derive the coded packet from the native packets. For a coded packet $p' = \sum c_i p_i$, where p_i's are the native packets, the code vector is $\vec{c} = (c_1, c_2, \ldots, c_K)$.
Innovative Packet	A packet is innovative to a node if it is linearly independent from its previously received packets.
Closer to destination	Node X is closer than node Y to the destination, if the best path from X to the destination has a lower ETX metric than that from Y.

Table 1—Definitions used in the paper.

dard best path routing (e.g., Srcr [9]), with which MORE benignly co-exists.

Table 1 defines the terms used in the rest of the paper.

4.1 Source

The source breaks up the file into batches of K packets, where K may vary from one batch to another. These K uncoded packets are called *native 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. In MORE, data packets are always coded. A *coded packet* is $p' = \sum_i c_i p_i$, where the c_i's are random coefficients picked by the node, and the p_i's are native packets from the same batch. We call $\vec{c} = (c_1, \ldots, c_i, \ldots, c_K)$ the packet's *code vector*. Thus, the code vector describes how to generate the coded packet from the native packets.

The sender attaches a MORE header to each data packet. The header reports the packet's code vector (which will be used in decoding), the batch ID, the source and destination IP addresses, and the list of nodes that could participate in forwarding the packet (Fig. 3). To compute the forwarder list, we leverage the ETX calculations [12]. Specifically, nodes periodically ping each other and estimate the delivery probability on each link. They use these probabilities to compute the ETX distance to the destination, which is the expected number of transmissions to deliver a packet from each node to the destination. The sender includes in the forwarder list nodes that are closer (in ETX metric) to the destination than itself, ordered according to their proximity to the destination.

The sender keeps transmitting coded packets from the current batch until the batch is acked by the destination, at which time, the sender proceeds to the next batch.

4.2 Forwaders

Nodes listen to all transmissions. When a node hears a packet, it checks whether it is in the packet's forwarder list. If so, the node checks whether the packet contains new information, in which case it is called an *innovative packet*. Technically speaking, a packet is innovative if it is linearly independent from the packets the node has previously received from this batch. Checking for independence can be done using simple Algebra (Gaussian Elimination [24]). The node ignores non-innovative packets, and stores the innovative packets it receives from the current batch.

If the node is in the forwarder list, the arrival of this new packet triggers the node to broadcast a coded packet. To do so the node creates a random linear combination of the coded packets it has heard from the same batch and broadcasts it. Note that *a linear combination of coded packets is also a linear combination of the corresponding native packets.* In particular, assume that the forwarder has heard coded packets of the form $p'_j = \sum_i c_{ji} p_i$, where p_i is a native packet. It linearly combines these coded packets to create more coded packets as follows: $p'' = \sum_j r_j p'_j$, where r_j's are random numbers. The resulting coded packet p'' can be expressed in terms of the native packets as follows $p'' = \sum_j (r_j \sum_i c_{ji} p_i) = \sum_i (\sum_j r_j c_{ji}) p_i$; thus, it is a linear combination of the native packets themselves.

4.3 Destination

For each packet it receives, the destination checks whether the packet is innovative, i.e., it is linearly independent from previously received packets. The destination discards non-innovative packets because they do not contain new information. Once the destination receives K innovative packets, it decodes the whole batch (i.e., it obtains the native packets) using simple matrix inversion:

$$\begin{pmatrix} p_1 \\ \vdots \\ p_K \end{pmatrix} = \begin{pmatrix} c_{11} & \ldots & c_{1K} \\ \vdots & \ddots & \\ c_{K1} & \ldots & c_{KK} \end{pmatrix}^{-1} \begin{pmatrix} p'_1 \\ \vdots \\ p'_K \end{pmatrix},$$

where, p_i is a native packet, and p'_i is a coded packet whose code vector is $\vec{c_i} = c_{i1}, \ldots, c_{iK}$. As soon as the destination decodes the batch, it sends an acknowledgment to the source to allow it to move to the next batch. ACKs are sent using best path routing, which is possible because MORE uses standard 802.11 and co-exists with sortest path routing. ACKs are also given priority over data packets at every node.

5 PRACTICAL CHALLENGES

In §4, we have described the general design of MORE. But for the protocol to be practical, MORE has to address 3 additional challenges, which we discuss in detail below.

5.1 How Many Packets Does a Forwarder Send?

In traditional best path routing, a node keeps transmitting a packet until the nexthop receives it, or the number of transmissions exceeds a particular threshold, at which time the node gives up. In opportunistic routing, however, there

is no particular nexthop; all nodes closer to the destination than the current transmitter are potential nexthops and may participate in forwarding the packet. How many transmissions are sufficient to ensure that *at least one* node closer to the destination has received the packet? This is an open question. Prior work has looked at a simplified and theoretical version of the problem that assumes smooth traffic rates and infinite wireless capacity [27]. In practice, however, traffic is bursty and the wireless capacity is far from infinite.

In this section, we provide a heuristic-based practical solution to the above problem. Our solution has the following desirable characteristics: 1) It has low complexity. 2) It is distributed. 3) It naturally integrates with 802.11 and preserves spatial reuse. 4) It is practical– i.e., it makes no assumptions of infinite capacity or traffic smoothness, and requires only the average loss rate of the links.

5.1.1 Practical Solution

Bandwidth is typically the scarcest resource in a wireless network. Thus, the natural approach to increase wireless throughput is to decrease the number of transmissions necessary to deliver a packet from the source to the destination [10, 12, 9]. Let the distance from a node, i, to the destination, d, be the expected number of transmissions to deliver a packet from i to d along the best path– i.e., node i's ETX [12]. We propose the following heuristic to route a packet from the source, s, to the destination, d: when a node transmits a packet, the node closest to the destination in ETX metric among those that receive the packet should forward it onward. The above heuristic reduces the expected number of transmissions needed to deliver the packet, and thus improves the overall throughput.

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 closer to the destination than node j, or said differently, i has a smaller ETX than j. Let p_{ij} denote the loss probability in sending a packet from i to j. Let z_i be the expected number of transmissions that forwarder i must make to route one packet from the source, s, to the destination, d, when all nodes follow the above routing heuristic. In the following, we assume that wireless receptions at different nodes are independent, an assumption that is supported by prior measurements [31, 29].

We focus on delivering one packet from source to destination. Let us calculate the number of packets that a forwarder j must forward to deliver one packet from source, s to destination, d. The expected number of packets that j receives from nodes with higher ETX is $\sum_{i>j} z_i(1-p_{ij})$. For each packet j receives, j should forward it only if no node with lower ETX metric hears the packet. This happens with probability $\prod_{k<j} p_{ik}$. Thus, in expectation, the number of packets that j must forward, denoted by L_j, is:

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

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

Now, consider the expected number of transmissions a node j must make. j should transmit each packet until a node with lower ETX has received it. Thus, the number of transmissions that j makes for each packet it forwards is a geometric random variable with success probability $(1 - \prod_{k<j} p_{jk})$. This is the probability that some node with ETX lower than j receives the packet. Knowing the number of packets that j has to forward from Eq. (1), the expected number of transmissions that j must make is:

$$z_j = \frac{L_j}{(1 - \prod_{k<j} p_{jk})}. \quad (2)$$

(a) Low Complexity: The number of transmissions made by each node, the z_i's, can be computed via the following algorithm. We can ignore nodes whose ETX to the destination is greater than that of the source, since they are not useful in forwarding packets for this flow. Next, we order the nodes by increasing ETX from the destination d and relabel them according to this ordering, i.e. $d = 1$ and $s = n$. We begin at the source by setting $L_n = 1$, then compute Eqs. (1) and (2) from source progressing towards the destination. To reduce the complexity, we will compute the values incrementally. Consider L_j, as given by Eq. (1). If we computed it in one shot, we would need to compute the product $\prod_{k<j} p_{ik}$ from scratch for each $i > j$. The idea is to instead compute and accumulate the contribution of node i to L_j's of all nodes j with lower ETX, so that each time we only need to make a small update to this product (denoted P in the algorithm).

1 Computing the number of transmissions each node makes to deliver a packet from source to destination, z_i's

```
for i = n ... 1 do
    L_i ← 0
L_n ← 1    {at source}
for i = n ... 2 do
    z_i ← L_i/(1 − ∏_{j<i} p_ij)
    P ← 1
    for j = 2 ... i − 1 do
        {compute the contribution of i to L_j}
        P ← P × p_{i(j−1)}    {here, P is ∏_{k<j} p_ik}
        L_j ← L_j + z_i × P × (1 − p_ij)
```

Alg. 1 requires $O(N^2)$ operations, where N is the number of nodes in the network. This is because the outer loop is executed at most n times and each iteration of the inner loop requires $O(n)$ operations, where n is bounded by the number of nodes in the network, N.

(b) Distributed Solution: Each node j can periodically measure the loss probabilities p_{ij} for each of its neighbors via ping probes. These probabilities are distributed to other nodes in the network in a manner similar to link state protocols [9]. Each node can then build the network

graph annotated with the link loss probabilities and compute Eq. (2) from the p_{ij}'s using the algorithm above.

(c) Integrated with 802.11: A distributed low-complexity solution to the problem is not sufficient. The solution tells each node the value of z_i, i.e., the number of transmissions it needs to make for every packet sent by the source. But a forwarder cannot usually tell when the source has transmitted a new packet. In a large network, many forwarders are not in the source's range. Even those forwarders in the range of the source do not perfectly receive every transmission made by the source and thus cannot tell whether the source has sent a new packet. Said differently, the above assumes a special scheduler that tells each node when to transmit.

In practice, a router should be triggered to transmit only when it receives a packet, and should perform the transmission only when the 802.11 MAC permits. We leverage the preceding to compute how many transmissions each router needs to make for every packet it receives. Define the TX_credit of a node as the number of transmissions that a node should make for every packet it receives from a node farther from the destination in the ETX metric. For each packet sent from source to destination, node i receives $\sum_{j>i} p_{ji} z_j$, where z_j is the number of transmissions made by node j and p_{ji} is the delivery probability from j to i. Thus, the TX_credit of node i is:

$$\text{TX\_credit}_i = \frac{z_i}{\sum_{j>i} z_j p_{ji}}. \quad (3)$$

Thus, in MORE, a forwarder node i keeps a `credit counter`. When node i receives a packet from a node upstream, it increments the counter by its TX_credit. When the 802.11 MAC allows the node to transmit, the node checks whether the counter is positive. If yes, the node creates a coded packet, broadcasts it, then decrements the counter. If the counter is negative, the node does not transmit. The ETX metric order ensures that there are no loops in crediting, which could lead to credit explosion.

5.1.2 Pruning

MORE's solution to the above might include forwarders that make very few transmissions (z_i is very small), and thus, have very little contribution to the routing. In a dense network, we might have a large number of such low contribution forwarders. Since the overhead of channel contention increases with the number of forwarders, it is useful to prune such nodes. MORE prunes forwarders that are expected to perform less than 10% of all the transmissions for the batch (more precisely, it prunes nodes whose $z_i < 0.1 \sum_{j \in N} z_j$).

5.2 Stopping Rule

In MORE, traffic is pumped into the network by the source. The forwarders do not generate traffic unless they receive new packets. It is important to throttle the source's transmissions as soon as the destination has received enough packets to decode the batch. Thus, once the destination receives the K^{th} innovative packet, and before fully decoding the batch, it sends an ACK to the source.

To expedite the delivery of ACKs, they are sent on the shortest path from destination to source. Furthermore, ACKs are given priority over data packets at all nodes and are reliably delivered using local retransmission at each hop.

When the sender receives an acknowledgment for the current batch, it stops forwarding packets from that batch. If the transfer is not complete yet, the sender proceeds to transmit packets from the next batch.

The forwarders are triggered by the arrival of new packets, and thus stop transmitting packets from a particular batch once the sender stops doing so. Eventually the batch will timeout and be flushed from memory. Additionally, forwarders that hear the ACK while it is being transmitted towards the sender immediately stop transmitting packets from that batch and purge it from their memory. Finally, the arrival of a new batch from the sender causes a forwarder to flush all buffered packets with batch ID's lower than the active batch.

5.3 Fast Network Coding

Network coding, implemented naively, can be expensive. As outlined above, the routers forward linear combinations of the packets they receive. Combining N packets of size S bytes requires NS multiplications and additions. Due to the broadcast nature of the wireless medium, routers could receive many packets from the same batch. If a router codes all these packets together, the coding cost may be overwhelming, creating a CPU bottleneck.

MORE employs three techniques to produce efficient coding that ensure the routers can easily support high bit rates.

(a) Code only Innovative Packets: The coding cost scales with the number of packets coded together. Typically, network coding makes routers forward linear combinations of the received packets. Coding non-innovative packets, however, is not useful because they do not add any information content. Hence, when a MORE forwarder receives a new packet, it checks if the packet is innovative and throws away non-innovative packets. Since innovative packets are by definition linearly independent, the number of innovative packets in any batch is bounded by the batch size K. Discarding non-innovative packets bounds both the number of packets the forwarder buffers from any batch, and the number of packets combined together to produce a coded packet. Discarding non-innovative packets is particularly important in wireless because the broadcast nature of the medium makes the number of received packets much larger than innovative packets.

(b) Operate on Code Vectors: When a new packet is

received, checking for innovativeness implies checking whether the received packet is linearly independent of the set of packets from the same batch already stored at the node. Checking independence of all data bytes is very expensive. Fortunately, this is unnecessary. The forwarder node simply checks if the code vectors are linearly independent. (Checking for vector independence can be done using Gaussian elimination [13]. To amortize the cost over all packets each node keeps code vectors of the packets in its buffer in row echelon form.) The data in the packet itself is not touched; it is just stored in a pool to be used later when the node needs to forward a linear combination from the batch. Thus, operations on individual data bytes happen only occasionally at the time of coding or decoding, while checking for innovativeness, which occurs for every overheard packet, is fairly cheap.

(c) Pre-Coding: When the wireless driver is ready to send a packet, the node has to generate a linear combination of the buffered packets and hand that coded packet to the wireless card. Linearly combining packets involves multiplying individual bytes in those packets, which could take hundreds of microseconds. This inserts significant delay before every transmission, decreasing the overall throughput.

To address this issue, MORE exploits the time when the wireless medium is unavailable to pre-compute one linear combination, so that a coded packet is ready when the medium becomes available. If the node receives an innovative packet before the prepared packet is handed over to the driver, the pre-coded packet is updated by multiplying the newly arrived packet with a random coefficient and adding it to the pre-coded packet. This approach achieves two important goals. On one hand, it ensures the transmitted coded packet contains information from all packets known to the node, including the most recent arrival. On the other hand, it avoids inserting a delay before each transmission.

6 Implementation Details

Finally, we put the various pieces together and explain the system details.

6.1 Packet Format

MORE inserts a variable length header in each packet, as shown in Fig. 3. The header starts with a few required fields that appear in every MORE packet. The type field distinguishes data packets, which carry coded information, from ACKs, which signal batch delivery. The header also contains the source and destination IP addresses and the flow ID. The last required field is the batch ID, which identifies the batch to which the packet belongs. The above is followed by a few optional fields. The code vector exists only in data packets and identifies the coefficients that generate the coded packet from the native packets in the batch. The list of forwarders has variable length and identifies all potential forwarders ordered according to their proximity to the source. For each forwarder, the packet also contains its TX_credit (see §*5.1.1*). Except for the code vector, all fields are initialized by the source and copied to the packets created by the forwarders. In contrast, the code vector is computed locally by each forwarder based on the random coefficients they picked for the packet.

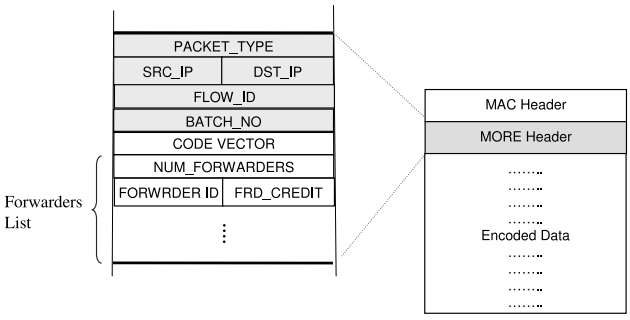
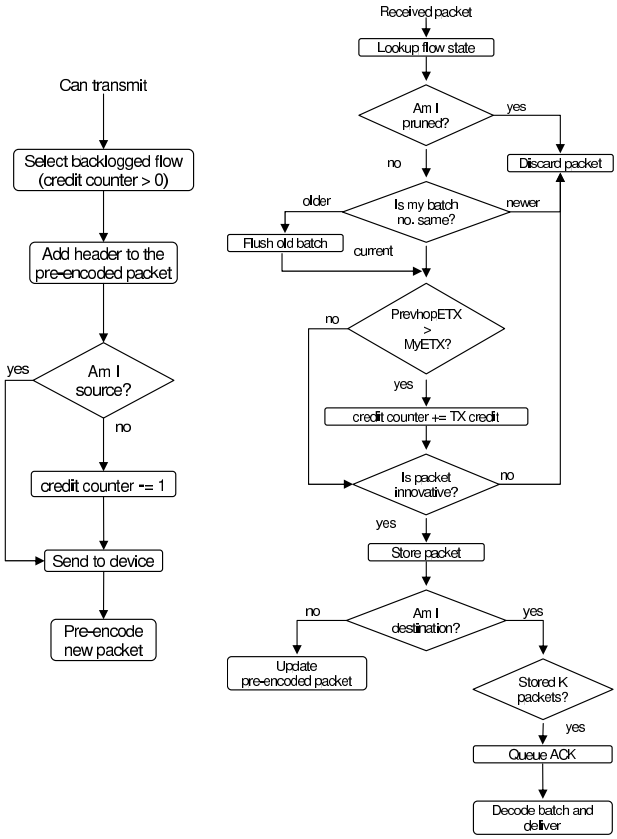


Figure 3—MORE Header. Grey fields are required while the white fields are optional. The packet type identifies batch ACKs from data packets.



(a) Sender side **(b) Receiver side**

Figure 4—MORE's Architecture. The figure shows a flow chart of our MORE implementation.

6.2 Node State

Each MORE node maintains state for the flows it forwards. The per-flow state is initialized by the reception of the first packet from a flow that contains the node ID in the list of forwarders. The state is timed-out if no packets from the flow arrive for 5 minutes. The source keeps transmitting packets until the destination acks the last batch of the flow. These packets will re-initialize the state at the forwarder even if it is timed out prematurely. The per-flow state includes the following.

- The `batch buffer` stores the received innovative packets. Note that the number of innovative packets in a batch is bounded by the batch size K.
- The `current batch` variable identifies the most recent batch from the flow.
- The `forwarder list` contains the list of forwarders and their corresponding TX_credits, ordered according to their distance from the destination. The list is copied from one of the received packets, where it was initialized by the source.
- The `credit counter` tracks the transmission credit. For each packet arrival from a node with a higher ETX, the forwarder increments the counter by its corresponding TX_CREDIT, and decrements it 1 for each transmission. A forwarder transmits only when the counter is positive.

6.3 Control Flow

Figure 4 shows the architecture of MORE. The control flow responds to packet reception and transmission opportunity signaled by the 802.11 driver.

On the sending side, the forwarder prepares a pre-coded packet for every backlogged flow to avoid delay when the MAC is ready for transmission. A flow is backlogged if it has a positive `credit counter`. Whenever the MAC signals an opportunity to transmit, the node selects a backlogged flow by round-robin and pushes its pre-coded packet to the network interface. As soon as the transmission starts, a new packet is pre-coded for this flow and stored for future use. If the node is a forwarder, it decrements the flow's `credit counter`.

On the receiving side, when a packet arrives the node checks whether it is a forwarder by looking for its ID in the forwarder list in the header. If the node is a forwarder, it checks if the batch ID on the packet is the same as its `current batch`. If the batch ID in the packet is higher than the node's `current batch`, the node sets `current batch` to the more recent batch ID and flushes packets from older batches from its `batch buffer`. If the packet was transmitted from upstream, the node also increments its `credit counter` by its TX_credit. Next, the node performs a linear independence check to determine whether the packet is innovative. Innovative packets are added to the `batch buffer` while non-innovative packets are discarded.

Further processing depends on whether the node is the packet's final destination or just a forwarder. If the node is a forwarder, the pre-coded packet from this flow is updated by adding the recent packet multiplied by a random coefficient. In contrast, if the node is the destination of the flow, it checks whether it has received a full batch (i.e., K innovative packets). If so, it queues an ACK for the batch, decodes the native packets and pushes them to the upper layer.

6.4 ACK Processing

ACK packets are routed to the source along the shortest ETX path. ACKs are also prioritized over data packets and transferred reliably. In our implementation, when a transmission opportunity arises, a flow that has queued ACK is given priority, and the ACK packet is passed to the device. Unless the transmission succeeds (i.e., is acknowledged by the MAC of the nexthop) the ACK is queued again. In addition, all nodes that overhear a batch ACK update their `current batch` variable and flush packets from the acked batch from their `batch buffer`.

7 MULTICAST

Multicast in MORE is a natural extension of unicast. All of our prior description carries on to the multicast case except for three simple modifications.

First, the source does not proceed to the next batch until all destinations have received the current batch.

Second, the list of forwarders and their TX_credits are different. The source computes the TX_credits and the forwarder list for hypothetical unicast flows from itself to each of the destinations in the multicast group. The forwarder list of the multicast flow is the union of the forwarders of the unicast flows. The TX_credit of each forwarder is computed as the maximum of the TX_credits that the forwarder gets in each of the hypothetical unicast flows.

Third, for multicast the TX_credit of a forwarder takes a dynamic nature. In particular, as the current batch progresses towards the end, more and more destinations are able to decode. Those forwarders that were included in the forwarder list in order to reach destinations that have already decoded the batch are temporarily not needed. Thus, whenever a destination acks the current batch, the source recomputes the TX_credits of the forwarders as the maximum TX_credit taken over only the hypothetical unicast flows to the destinations that have not yet decoded the batch. The forwarders that hear the new TX_credit in the packet update their information accordingly.

8 EXPERIMENTAL RESULTS

We use measurements from a 20-node wireless testbed to evaluate MORE, compare it with both ExOR and traditional best path routing, and estimate its overhead. Our experiments reveal the following findings.

- On average, MORE achieves 20% better throughput than ExOR. In comparison with traditional routing, MORE improves the average throughput by 70%, and the maximum throughput gain exceeds 10x.
- MORE's throughput exceeds ExOR's mainly because of its ability to exploit spatial reuse. Focusing on flows that traverse paths with 25% chance of concurrent transmissions, we find that MORE's throughput is 50% higher than that of ExOR.
- For multicast traffic, MORE's throughput gain increases with the number of destinations. For 2-4 destinations, MORE's throughput is 35-200% larger than ExOR's. In comparison to traditional routing, the multicast gain can be as high as 3x.
- MORE significantly eases the problem of dead spots. In particular, 90% of the flows achieve a throughput higher than 50 packets/second. In traditional routing the 10^{th} percentile is only 10 packets/second.
- MORE keeps its throughput gain over traditional routing even when the latter is allowed automatic rate selection.
- MORE is insensitive to the batch size and maintains large throughput gains with batch size as low as 8 packets.
- Finally, we estimate MORE's overhead. MORE stores the current batch from each flow. Our MORE implementation supports up to 44 Mb/s on low-end machines with Celeron 800MHz CPU and 128KiB of cache. Thus, MORE's overhead is reasonable for the environment it is designed for, namely stationary wireless meshes, such as Roofnet [4] and community wireless networks [32, 6].

We will make our code public including the finite field coding libraries.

8.1 Testbed

(a) Characteristics: We have a 20-node wireless testbed that spans three floors in our building connected via open lounges. The nodes of the testbed are distributed in several offices, passages, and lounges. Fig. 5 shows the locations of the nodes on one of the floors. Paths between nodes are 1–5 hops in length, and the loss rates of links on these paths vary between 0 and 60%, and averages to 27%.

(b) Hardware: Each node in the testbed is a PC equipped with a NETGEAR WAG311 wireless card attached to an omni-directional antenna. They transmit at a power level of 18 dBm, and operate in the 802.11 ad hoc mode, with RTS/CTS disabled.

(c) Software: Nodes in the testbed run Linux, the Click toolkit [25] and the Roofnet software package [4]. Our implementation runs as a user space daemon on Linux. It sends and receives raw 802.11 frames from the wireless device using a libpcap-like interface.

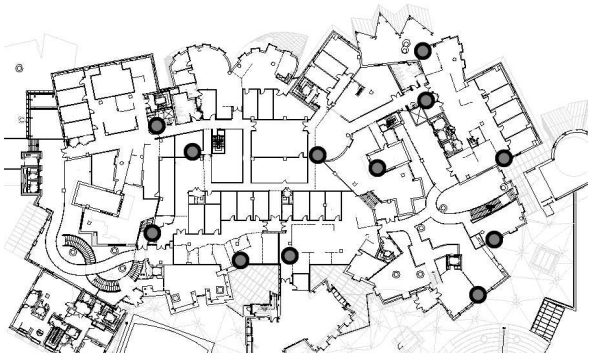


Figure 5—One Floor of our Testbed. Nodes' location on one floor of our 3-floor testbed.

8.2 Compared Protocols

We compare the following three protocols.

- *MORE* as explained in §6.
- *ExOR* [10], the current opportunistic routing protocol. Our ExOR code is provided by its authors.
- *Srcr* [9] which is a state-of-the-art best path routing protocol for wireless mesh networks. It uses Dijkstra's shortest path algorithm where link weights are assigned based on the ETX metric [12].

8.3 Setup

In each experiment, we run Srcr, MORE, and ExOR in sequence between the same source destination pairs. Each run transfers a 5 MByte file. We leverage the ETX implementation provided with the Roofnet Software to measure link delivery probabilities. Before running an experiment, we run the ETX measurement module for 10 minutes to compute pair-wise delivery probabilities and the corresponding ETX metric. These measurements are then fed to all three protocols, Srcr, MORE, and ExOR, and used for route selection.

Unless stated differently, the batch size for both MORE and ExOR is set to $K = 32$ packets. The packet size for all three protocols is 1500B. The queue size at Srcr routers is 50 packets. In contrast, MORE and ExOR do not use queues; they buffer active batches.

Most experiments are performed over 802.11b with a bit-rate of 5.5Mb/s. In §8.7, we allow traditional routing (i.e., Srcr) to exploit the autorate feature in the MadWifi driver, which uses the Onoe bit-rate selection algorithm [8]. Current autorate control optimizes the bit-rate for the nexthop, making it unsuitable for opportunistic routing, which broadcasts every transmission to many potential nexthops. The problem of autorate control for opportunistic routing is still open. Thus in our experiments, we compare Srcr with autorate to opportunistic routing (MORE and ExOR) with a fixed bit-rate of 11 Mb/s.

8.4 Throughput

We would like to examine whether MORE can effectively exploit opportunistic receptions to improve the throughput and compare it with Srcr and ExOR.

8.4.1 How Do the Three Protocols Compare?

Does MORE improve over ExOR? How do these two opportunistic routing protocols compare with traditional best path routing? To answer these questions, we use these protocols to transfer a 5 MByte file between various nodes in our testbed. We repeat the same experiment for MORE, ExOR, and Srcr as explained in §8.3.

Our results show that MORE significantly improves the unicast throughput. In particular, Fig. 6 plots the CDF of the throughput taken over 200 randomly selected source-destination pairs in our testbed. The figure shows that both MORE and ExOR significantly outperform Srcr. Interestingly, however, MORE's throughput is higher than ExOR's. On average, MORE has 20% throughput gain over ExOR. Its throughput gain over Srcr is 70%, but some challenged flows achieve 10-12x higher throughput with MORE than traditional routing.

Further, MORE and opportunistic routing ease the problem of dead spots. Fig. 6 shows that over 90% of MORE flows have a throughput larger than 50 packets a second. ExOR's 10^{th} percentile is at 40 packets a second. Srcr on the other hand suffers from dead spots with many flows experiencing very low throughput. Specifically, the 10^{th} percentile of Srcr's throughput is at 10 packets a second.

8.4.2 When Does Opportunistic Routing Win?

We try to identify the scenarios in which protocols like MORE and ExOR are particularly useful– i.e., when should one expect opportunistic routing to bring a large throughput gain? Fig. 7a shows the scatter plot for the throughputs achieved under Srcr and MORE for the same source-destination pair. Fig. 7b gives an analogous plot for ExOR. Points on the 45-degree line have the same throughput in the two compared schemes.

These figures reveal that opportunistic routing (MORE and ExOR) greatly improves performance for challenged flows, i.e., flows that usually have low throughput. Flows that achieve good throughput under Srcr do not improve further. This is because when links on the best path have very good quality, there is little benefit from exploiting opportunistic receptions. In contrast, a source-destination pair that obtains a low throughout under Srcr does not have any good quality path. Usually, however, many low-quality paths exist between the source and the destination. By using the combined capacity of all these low-quality paths, MORE and ExOR manage to boost the throughput of such flows.

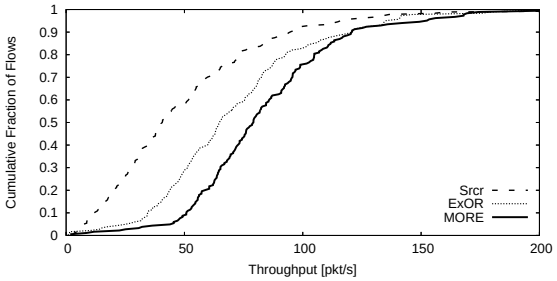
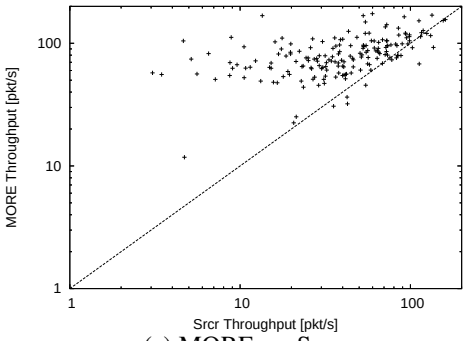
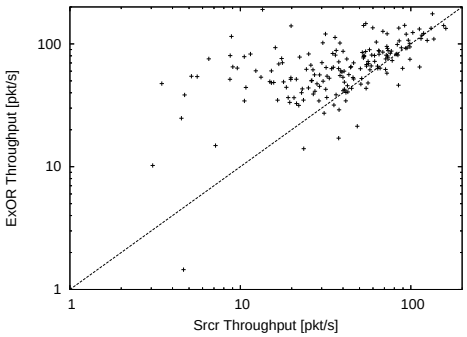


Figure 6—Unicast Throughput. Figure shows the CDF of the unicast throughput achieved with MORE, ExOR, and Srcr. MORE's average throughput is 20% higher than ExOR. In comparison to Srcr, MORE achieves an average throughput gain of 70%, while some source-destination pairs show as much as 10-12x.



(a) MORE vs. Srcr



(b) ExOR vs. Srcr

Figure 7—Scatter Plot of Unicast Throughput. Each point represents the throughput of a particular source destination pair. Points above the 45-degree line indicate improvement with opportunistic routing. The figure shows that opportunistic routing is particularly beneficial to challenged flows.

8.4.3 Why Does MORE Have Higher Throughput than ExOR?

Our experiments show that spatial reuse is a main contributor to MORE's gain over ExOR. ExOR prevents multiple forwarders from accessing the medium simultaneously [10], and thus does not exploit spatial reuse. To examine this issue closely, we focus on a few flows that

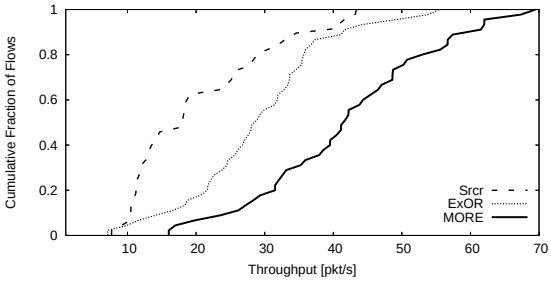


Figure 8—Spatial Reuse. The figure shows CDFs of unicast throughput achieved by MORE, ExOR, and Srcr for flows that traverse 4 hops, where the last hop can transmit concurrently with the first hop. MORE's average throughput is 50% higher than ExOR.

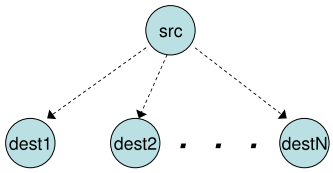


Figure 9—Multicast Topology. A simple topology used in the multicast experiments in Fig. 10.

we know can benefit from spatial reuse. Each flow has a best path of 4 hops, where the last hop can send concurrently with the first hop without collision. Fig. 8 plots the CDF of throughput of the three protocols for this environment. Focusing on paths with spatial reuse amplifies the gain MORE has over ExOR. The figure shows that for 4-hop flows with spatial reuse, MORE on average achieves a 50% higher throughput than ExOR.

It is important to note that spatial reuse may occur even for shorter paths. The capture effect allows multiple transmissions to be correctly received even when the nodes are within the radio range of both senders [31]. In particular, less than 7% of the flows in Fig. 6 have a best path of 4 hops or longer. Still MORE does better than ExOR. This is mainly because of capture. The capture effect, however, is hard to quantify or measure. Thus, we have focused on longer paths to show the impact of spatial reuse.

8.5 Multicast

We want to compare the performance of multicast traffic under MORE, ExOR, and Srcr. In §7, we have described how multicast works under MORE. In contrast, ExOR [10] and Srcr [9] do not have multicast extensions. Thus, we need to define how these protocols deal with multicast. For Srcr we adopt the same approach as wired multicast. Specifically, we find the shortest path from the source to each destination, using ETX as the metric. These paths create a tree rooted at the source. Srcr's multicast traffic is sent along the branches of this tree. In contrast, with ExOR, we want multicast traffic to exploit opportunistic receptions. We find the ExOR forwarders for each destination. The per-destination forwarders use the ExOR protocol to access the medium and coordinate their transmissions. In contrast to unicast ExOR, if the forwarders toward destination X opportunistically hears a packet by a forwarder in the forwarder list of destination Y, it exploits that opportunistic reception. Said differently, we allow opportunistic receptions across the forwarders of various destinations.

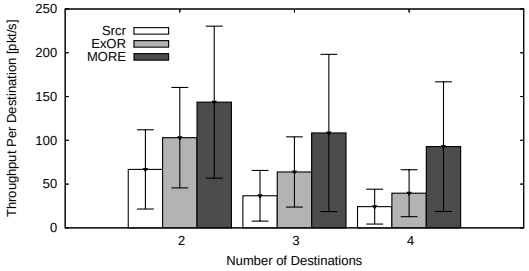


Figure 10—Multicast Throughput as a Function of the Number of Destinations for the Topology in Fig. 9. The figure shows the per-destination multicast throughput of MORE, ExOR, and Srcr. The thick bars show the average per-destination throughput taken over 40 runs with different nodes. The lines show the standard deviation.

Our results show that MORE's multicast throughput is significantly higher than both ExOR and Srcr. In particular, we experiment with the simple topology in Fig. 9, where the source multicasts a file to a varying number of destinations. Fig. 10 shows the average multicast throughput as a function of the number of destinations. The average is computed over 40 different instantiations of the topology in Fig 9, using nodes in our testbed. As expected, the per-destination average throughput decreases with increased number of destinations. Interestingly however, the figure shows that MORE's throughput gain increases with increased number of destinations. MORE has 35-200% throughput gain over ExOR and 100-300% gain over Srcr.

MORE's multicast throughput gain is higher than its unicast gain. This is because network coding fits naturally with multicast. Recall from the example in §2 that without network coding, a transmitter (whether the source or a forwarder) needs to retransmit the union of all packets lost by downstream nodes. In contrast, with coding it is enough to transmit just the number of packets missed at the downstream node that experienced the most packet loss.

Next, we run multicast over random topologies and multihop paths. We pick a source and 3 destinations randomly from the nodes in the testbed. We make the source multicast a file to the three destinations, using MORE, ExOR, and Srcr. We repeat the experiment for 40 different instantiations of the nodes, and plot the CDFs of the throughput. Fig. 11 confirms our prior results showing significant gain for MORE over both ExOR and Srcr. In this figure however the difference between MORE and ExOR

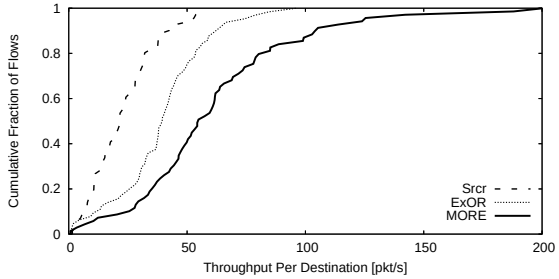


Figure 11—CDF of Multicast Throughput for 3 Destinations in a Random Topology. The figure shows the CDF of the per-destination multicast throughput of MORE, ExOR, and Srcr. For each run, a source and 3 destinations are picked randomly from among the nodes in the testbed.

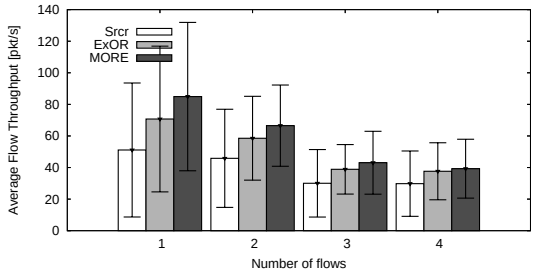


Figure 12—Multi-flows. The figure plots the per-flow average throughput in scenarios with multiple flows. Bar show the average of 40 random runs. Lines show the standard deviation.

is less pronounced than in Fig. 10. This is because the CDF uses random topologies with all nodes in the testbed potentially acting as forwarders. This increases the potential for opportunistic receptions and thus makes the relative gain from network coding look less apparent.

8.6 Multiple Flows

One may also ask how MORE performs in the presence of multiple flows. Further, since the ExOR paper does not show any results for concurrent flows, this question is still open for ExOR as well. We run 40 multi-flow experiments with random choice of source-destination pairs, and repeat each run for the three protocols.

Fig. 12 shows the average per-flow throughput as a function of the number of concurrent flows, for the three protocols. Both MORE and ExOR achieve higher throughput than Srcr. The throughput gains of opportunistic routing, however, are lower than for a single flow. This highlights an inherent property of opportunistic routing; it exploits opportunistic receptions to boost the throughput, but it does not increase the capacity of the network. The 802.11 bit rate decides the maximum number of transmissions that can be made in a time unit. As the number of flows in the network increases, each node starts playing two roles; it is a forwarder on the best path for some flow, and a forwarder off the best path for another flow. If the driver polls the node to send a packet, it is better to send

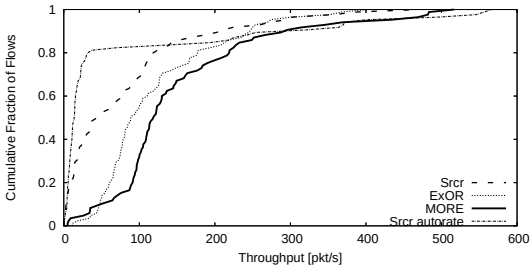


Figure 13—Opportunistic Routing Against Srcr with Autorate. The figure compares the throughput of MORE and ExOR running at 11Mb/s against that of Srcr with autorate. MORE and ExOR preserve their throughput gains over Srcr.

a packet from the flow for which the node is on the best path. This is because the links on the best path usually have higher delivery probability. Since the medium is congested and the number of transmissions is bounded, it is better to transmit over the higher quality links.

Also, the gap between MORE and ExOR decreases with multiple flows. Multiple flows increase congestion inside the network. Although a single ExOR flow may underutilize the medium because it is unable to exploit spatial reuse, the congestion arising from the increased number of flows covers this issue. When one ExOR flow becomes unnecessarily idle, another flow can proceed.

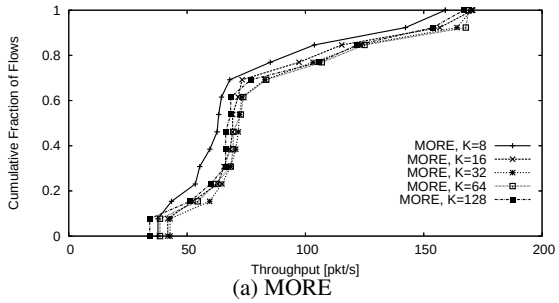
Although the benefits of opportunistic routing decrease with a congested network, it continues to do better than best path routing. Indeed, it smoothly degenerates to the behavior of traditional routing.

Finally, this section highlights the differences between inter-flow and intra-flow network coding. Katti et al. [23] show that the throughput gain of COPE, an inter-flow network coding protocol, increases with an increased number of flows. But, COPE does not apply to unidirectional traffic and cannot deal with dead spots. Thus, inter-flow and intra-flow network coding complement each other. A design that incorporates both MORE and COPE is a natural next step.

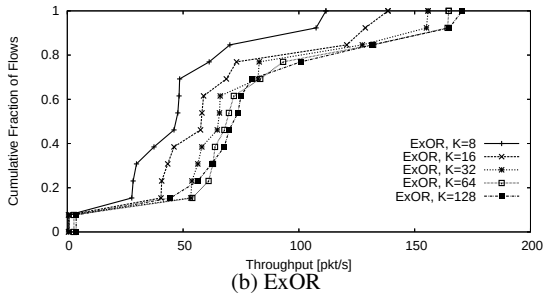
8.7 Autorate

Current 802.11 allows a sender node to change the bit rate automatically, depending on the quality of the link to the recipient. One may wonder whether such adaptation would improve the throughput of Srcr and nullify the gains of opportunistic routing. Thus, in this section, we allow Srcr to exploit the autorate feature in the MadWifi driver [28], which uses the Onoe bit-rate selection algorithm [8].

Opportunistic routing does not have the concept of a link, it broadcasts every packet to many potential nexthops. Thus, current autorate algorithms are not suitable for opportunistic routing. The problem of autorate control for opportunistic routing is still open. Therefore, in our experiments, we compare Srcr with autorate to opportunis-



(a) MORE



(b) ExOR

Figure 14—Impact of Batch Size. The figure shows the CDF of the throughput taken over 40 random node pairs. It shows that MORE is less sensitive to the batch size than ExOR.

tic routing (MORE and ExOR) with a fixed bit-rate of 11 Mb/s.

Fig. 13 shows CDFs of the throughputs of the various protocols. The figure shows that MORE and ExOR preserve their superiority to Srcr, even when the latter is combined with automatic rate selection. Paths with low throughput in traditional routing once again show the largest gains. Such paths have low quality links irrespective of the bit-rate used, therefore autorate selection does not help these paths.

Interestingly, the figure also shows that autorate does not necessarily perform better than fixing the bit-rate at the maximum value. This has been noted by prior work [34] and attributed to the autorate confusing collision drops from error drops and unnecessarily reducing the bit-rate.

8.8 Batch Size

We explore the performance of MORE and ExOR for various batch size. Fig. 14 plots the throughput for batch sizes of 8, 16, 32, 64, and 128. It shows that ExOR's performance with small batches of 8 packets is significantly worse than large batches. In contrast, MORE is highly insensitive to different batch sizes.

In both ExOR and MORE, the overhead increases with reduced batch size. ExOR nodes exchange control packets whenever they transmit a batch. Increasing the batch size allows ExOR to amortize the control traffic and reduces the chance of spurious transmissions. MORE may make a few spurious transmissions between the time the destination decodes a batch and when the source and forwarders stop transmitting packets from that batch. A bigger batch size allows MORE to amortize the cost of these spurious transmissions over a larger number of packets, increasing the overall throughput.

Insensitivity to batch sizes allows MORE to vary the batch size to accommodate different transfer sizes. We expect that for any transfer size larger than 7-10 packets (i.e., a batch larger than 7-10 packets), MORE will show significant advantages. Shorter transfers can be sent using traditional routing. Note that MORE benignly co-exists with traditional routing, which it uses to deliver its ACKs.

Operation	Avg. Time [μs]	Std. Dev. [μs]
Independence check	10	5
Coding at the source	270	15
Decoding	260	150

Table 2—Average computational cost of packet operations in MORE. The numbers for $K = 32$ and 1500B packets are measured on a low-end Celeron machine clocked at 800MHz with 128KiB cache. Note that the coding cost is highest at the source because it has to code all K packets together. The coding cost at a forwarder depends on the number of innovative packets it has received, and is always bounded by the coding cost at the source.

8.9 MORE's Overhead

Finally, we would like to estimate MORE's overhead and itssuitabilityy for deployment in mesh networks like Roofnet [4] and community wireless networks [32, 6].

(a) Coding Overhead: In MORE, the cost of coding/decoding packets is incurred mainly when the packet has to be multiplied by a random number (in a finite field of size 2^8). To optimize this operation, our implementation reduces the cost by using a 64KiB lookup-table indexed by pairs of 8 bits. The lookup table caches results of all possible multiplications, so multiplying any byte of a packet with a random number is simply a fast lookup.

Table 2 provides micro benchmarks for coding and decoding in MORE. The measurements are taken on a low-end Celeron 800MHz machine. The benchmarks show that coding and decoding have roughly equal cost. They require on average K finite-field multiplications per byte, where K is the batch size. This ties the choice of K with the maximum achievable throughput. In our setting $K = 32$ and coding takes on average $270\mu s$ per 1500B packet. This limits the effective throughput to 44 Mb/s, which is higher than the effective bit rate of current wireless mesh networks [20].

(b) Memory Overhead: In MORE like in ExOR, routers do not keep an output queue. Instead, they store the current batch from each flow. This per-flow state is dominated by the storage required to buffer innovative packets from the current batch, which is bounded by $K = 32$ packets. Additionally, as stated above, MORE nodes keep a 64KiB lookup-table. Given that the number of concurrent flows in a mesh network is relatively small, we believe MORE's

memory overhead is acceptable.

(c) Header Overhead: MORE's header in our current implementation is bounded by 70 bytes because we bound the number of forwarders to 10. Certain values in the header are compressed to increase efficiency. For example, since routers only keep the current batch, we can represent batch IDs using a few bits. Similarly, we compress the node ID in the forwarder list to one byte, which is a hash of its IP. This works because only nodes whose ETX to the destination is smaller than the source are allowed to participate in forwarding. For 1500B packets, the header overhead is less than 5%. Note that our throughput numbers are computed over the delivered data, and thus they already account for header overhead.

9 CONCLUSION

Opportunistic routing and network coding are two powerful ideas which may at first sight appear unrelated. Our work combines these ideas in a natural fashion to provide opportunistic routing *without* node coordination. We design a practical system, MORE, that plugs random network coding into the current network stack, exploits the opportunism inherent in the wireless medium, and provides significant performance gains. Field tests on a 20-node wireless testbed show that MORE provides both unicast and multicast traffic with significantly higher throughput than both traditional routing and prior work on opportunistic routing.

REFERENCES

[1] Calling p2p: Peer-to-peer networks coming to a phone near you, 2005. http://www.econtentmag.com.

[2] Digiweb offers wireless IPTV in Ireland, 2005. http://www.dtg.org.uk/news/.

[3] Ruckus to announce wireless, IPTV deals with 15 telcos, 2006. http://www.eweek.com/article2/0,1895,1989290,00.asp.

[4] D. Aguayo, J. Bicket, S. Biswas, G. Judd, and R. Morris. Link-level measurements from an 802.11b mesh network. In *SIGCOMM*, 2004.

[5] R. Ahlswede, N. Cai, S. R. Li, and R. W. Yeung. Network Information Flow. In *IEEE Trans. on Information Theory*, Jul 2000.

[6] Bay area wireless user group. http://www.bawug.org.

[7] P. Bhagwat, B. Raman, and D. Sanghi. Turning 802.11 inside-out. In *HotNets*, 2003.

[8] J. Bicket. Bit-rate selection in wireless networks. *M.S. Thesis*, 2005.

[9] J. Bicket, D. Aguayo, S. Biswas, and R. Morris. Architecture and evaluation of an unplanned 802.11b mesh network. In *MOBICOM*, 2005.

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

[11] N. Cai and R. W. Yeung. Secure Network Coding. In *ISIT*, 2002.

[12] D. S. J. De Couto, D. Aguayo, J. Bicket, and R. Morris. A high-throughput path metric for multi-hop wireless routing. In *MOBICOM*, 2003.

[13] J. E. Gentle. *Numerical Linear Algebra for Applications in Statistics*. Springer–Verlag, 1998.

[14] C. Gkantsidis and P. Rodriguez. Network Coding for Large Scale Content Distribution. In *INFOCOM*, 2005.

[15] T. Ho, M. Médard, J. Shi, M. Effros, and D. Karger. On randomized network coding. In *Allerton*, 2003.

[16] D. T. J. N. Laneman and G. Wornell. Cooperative diversity in wireless networks: Efficient protocols and outage behavior. *IEEE Trans. on Information Theory*, Dec 2004.

[17] S. Jaggi, M. Langberg, S. Katti, T. Ho, D. Katabi, and M. Médard. Resilient Network Coding In The Presence of Byzantine Adversaries. In *INFOCOM*, 2007.

[18] S. Jaggi, P. Sanders, P. A. Chou, M. Effros, S. Egner, K. Jain, and L. Tolhuizen. Polynomial time algorithms for multicast network code construction. *IEEE Trans. on Information Theory*, 2003.

[19] A. Jiang. Network Coding for Joing Storage and Transmission with Minimum Cost. In *ISIT*, 2006.

[20] A. Kamerman and G. Aben. Net throughput with IEEE 802.11 wireless LANs. In *WCNC*, 2000.

[21] A. Kamra, V. Misra, J. Feldman, and D. Rubenstein. Growth Codes: Maximizing Sensor Network Data Persistence. In *SIGCOMM*, 2006.

[22] S. Katti, D. Katabi, W. Hu, H. S. Rahul, and M. Médard. The importance of being opportunistic: Practical network coding for wireless environments. In *Allerton*, 2005.

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

[24] R. Koetter and M. Médard. An algebraic approach to network coding. *IEEE/ACM Trans. on Networking*, 2003.

[25] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. The click modular router. *ACM Trans. on Computer Systems*, Aug 2000.

[26] S.-Y. R. Li, R. W. Yeung, and N. Cai. Linear network coding. *IEEE Trans. on Information Theory*, Feb 2003.

[27] D. S. Lun, M. Médard, and R. Koetter. Efficient operation of wireless packet networks using network coding. In *IWCT*, 2005.

[28] MADWiFi: Multiband Atheros Driver for WiFi. http://madwifi.org.

[29] A. K. Miu, H. Balakrishnan, and C. E. Koksal. Improving loss resilience with multi-radio diversity in wireless networks. In *MOBICOM*, 2005.

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

[31] C. Reis, R. Mahajan, M. Rodrig, D. Wetherall, and J. Zahorjan. Measurement-based models of delivery and interference. In *SIGCOMM*, 2006.

[32] Seattle wireless. http://www.seattlewireless.net.

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

[34] S. H. Y. Wong, S. Lu, H. Yang, and V. Bharghavan. Robust rate adaptation for 802.11 wireless networks. In *MOBICOM*, 2006.