Pacifier: High-Throughput, Reliable Multicast Without "Crying Babies" in Wireless Mesh Networks

Dimitrios Koutsonikolas, *Member, IEEE*, Y. Charlie Hu, *Senior Member, IEEE*, and Chih-Chun Wang, *Member, IEEE*

***Abstract*—In contrast to unicast routing, high-throughput reliable multicast routing in wireless mesh networks (WMNs) has received little attention. There are two primary challenges to supporting high-throughput, reliable multicast in WMNs. The first is no different from unicast: wireless links are inherently lossy due to varying channel conditions and interference. The second, known as the "crying baby" problem, is unique to multicast: the multicast source may have varying throughput to different multicast receivers, and hence trying to satisfy the reliability requirement for poorly connected receivers can potentially result in performance degradation for the rest of the receivers.**

In this paper, we propose *Pacifier*, a new high-throughput reliable multicast protocol for WMNs. *Pacifier* seamlessly integrates four building blocks, namely, *tree-based opportunistic routing, intra-flow network coding, source rate limiting, and round-robin batching,* to support high-throughput, reliable multicast routing in WMNs, while at the same time effectively addresses the "crying baby" problem. Our experiments on a 22-node 802.11 WMN testbed show that *Pacifier* increases the average throughput over a state-of-the-art reliable network coding-based protocol MORE by up to 144%, while at the same time it solves the "crying baby" problem by improving the throughput of well-connected receivers by up to a factor of 14.

***Index Terms*—reliable multicast; wireless mesh networks (WMNs); network coding.**

I. Introduction

WIRELESS mesh networks (WMNs) are increasingly being deployed for providing cheap, low maintenance Internet access (e.g. [1], [2], [3]). These networks have statically deployed mesh routers that are not energy constrained, and hence the main design challenge is to improve applications' performance, in particular, to provide high throughput and reliability in network access. Indeed, recent years have witnessed numerous "exotic" protocols that aim to improve the throughput and reliability of unicast routing. These include opportunistic routing (OR) protocols (e.g., [4]), protocols that exploit inter-flow (e.g., [5]) or intra-flow (e.g., [6]) network coding, as well as lower layer protocols (e.g., [7]).

In contrast to unicast routing, high-throughput, reliable multicast routing has received relatively little attention. In contrast to IP multicast in the wired Internet, we believe multicast is and will be a fundamental mode of communication in wireless networks due to the wireless multicast advantage (WMA) [8]. The use of 802.11 broadcast based multicast instead of individual unicast sessions has been heavily studied recently in the community in the context of 802.11 WLANs (e.g., [9], [10]). Also, in [6], the proposed MORE protocol is designed to to be a reliable unicast and multicast protocol to support future applications in multihop WMNs.

We envision at least three important classes of applications of high-throughput, reliable multicast in the context of community WMNs: (1) multicasting WMN-related software (e.g., a router software update or a security patch) hosted on a node controlled by the WMN operator to the WMN routers. (2) multicasting community-related audio/video files (e.g., a local football match that was held the day before) stored in a local database to WMN clients. In both these scenarios, the WMN node hosting the content serves as the multicast root. (3) a large group of WMN users downloading a popular file from the Internet (e.g., a new version of Windows); in that case, a proxy at the WMN gateway serves as the multicast root. For all these applications, using multicast instead of individual multihop unicast sessions can lead to significant wireless bandwidth savings due to the WMA.

A common characteristic of all these applications, in contrast to live streaming (another popular class of multicast applications), is a strict requirement of **100% Packet Delivery Ratio (PDR)**, since every byte of the downloaded file has to be received by **all** the receivers. This requirement makes many of the reliable multicast protocols proposed in the past (e.g., [11], [12], [13], [14]) inappropriate, since they cannot guarantee 100% PDR. In addition, reliability for this class of applications cannot come at the cost of significantly reduced throughput, unlike in many military applications [15], since Internet users always desire fast downloads.

The fundamental challenge in achieving reliable multicast in WMNs is no different from that of reliable unicast – that wireless links are lossy. To overcome this, researchers have applied classic techniques such as Automatic Repeat reQuest (ARQ), Forward Error Correction (FEC), or combinations of the two. The majority of the works on reliable multicast in multihop wireless networks either are solely based on ARQ (e.g., [16], [17]) which suffer the feedback implosion problem, or combine ARQ with congestion control (e.g., [18], [13]). A recent work [19] studied the applicability of FEC and hybrid ARQ-FEC techniques, borrowed from the wired Internet, to WMNs, and showed that RMDP [20], a hybrid ARQ-FEC protocol, can achieve both reliability and high throughput.

This work was supported in part by NSF grants CNS-0626703 and CCF-0845968.

D. Koutsonikolas, Y. C. Hu and C. C. Wang are with the School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN, 47907 USA e-mail: (dkoutson@purdue.edu, ychu@purdue.edu, chihw@purdue.edu).

More recently, researchers have applied network coding (NC), a technique originally developed for the wireline Internet, to overcome the above challenge. [21] showed that the operation of mixing packets resembles the operation of rateless FEC codes. Actually, NC can be viewed as a technique equivalent to performing hop-by-hop FEC, without the delay penalty incurred by the decoding operations at each hop, that would be required by hop-by-hop FEC. In [22], the authors went one step further and showed that the reliability gain (expressed as the expected number of transmissions) of NC over end-to-end FEC for a wireless multicast tree of height h with link loss rate p is in the order of $\Theta((\frac{1}{1-p})^h)$.

Practical work that exploits the idea of utilizing NC for reliable multicast is still at a preliminary stage. MORE [6] is the *only practical* NC-based protocol that supports high-throughput, reliable multicast. It combines NC with OR, with the primary goal of removing the need for coordination required in opportunistic routing. However, the design of MORE also guarantees reliability, i.e., MORE is a routing protocol for *reliable file transfer*, for both unicast and multicast.

A second fundamental challenge in reliable multicast, which is unique to multicast, is the "crying baby" problem as first pointed out in [23] in the context of multicast in the Internet. If one receiver has a particularly poor connection, then trying to satisfy the reliability requirement for that receiver may result in performance degradation for the rest of the receivers. This problem also raises the interesting question of what is a suitable definition of overall performance metric if multiple receivers are allowed to achieve uneven throughput. Regardless, a major challenge in the design of high throughput, reliable multicast protocols is whether it is possible to develop a protocol that improves the throughput of well-connected receivers without worsening the already low throughput of poorly connected receivers.

In spite of its significance, the "crying baby" problem has been largely ignored by the majority of the wireless reliable multicast protocols proposed in the past. To our best knowledge, BMCC [13], a multicast protocol for mobile ad hoc networks, was the first protocol to consider the problem in the context of multihop wireless networks. BMCC allows a router to drop packets on the path towards the worst receiver, in order to prevent that receiver from holding back the rest of the receivers. This solution is not applicable in file-download applications where 100% PDR is required and hence dropping packets for some receivers is not an option. Essentially, the requirement for 100% PDR makes the problem much more challenging.

In this paper, we propose *Pacifier*, a high-throughput, reliable multicast protocol that systematically addresses the above two challenges for reliable file transfer applications. *Pacifier* seamlessly integrates four building blocks, namely, *tree-based OR, intra-flow NC, source rate limiting, and round-robin batching*, to support high-throughput, reliable multicast routing and at the same time solve the "crying baby" problem. First, *Pacifier* builds an efficient multicast tree traditionally used by multicast protocols and naturally leverages it for opportunistic overhearing. Second, *Pacifier* applies intra-flow, random linear NC to overcome packet loss over lossy links which avoids hop-by-hop feedback and the coordination of multicast tree forwarders in packet forwarding. Third, *Pacifier* applies rate limiting at the source, reducing the congestion level in the network. Fourth, *Pacifier* solves the "crying baby" problem by having the source send batches of packets in a round-robin fashion. This functionality allows *Pacifier* to improve the throughput of well-connected nodes drastically and often times of poorly connected nodes.

To evaluate *Pacifier*, we first compare its performance against MORE, using extensive realistic simulations. Our simulations use a realistic physical model, with random signal variations due to fading, take into account the additional packet header overhead introduced by the use of NC and OR, and are conducted over a variety of network topologies and multicast groups. Our simulation results show that *Pacifier* increases the average throughput of multicast receivers over MORE by 171%, while it solves the "crying baby" problem, by increasing the maximum throughput gain for well-connected receivers by up to 20x. Interestingly and importantly, *Pacifier* also improves the throughput of the "crying babies", i.e., the poorly connected receivers, by up to 4.5x.

Second, since *Pacifier* uses the same type of NC as MORE, and has the same memory requirements at the routers, hence, like MORE, it can be easily implemented on commodity hardware. To demonstrate this, we present an application-layer implementation of *Pacifier* and MORE on Linux and their performance evaluation on a 22-node 802.11 WMN testbed deployed in two academic buildings on the Purdue University campus. Our testbed results verify the simulation results showing that *Pacifier* increases the average multicast throughput over MORE by 83-114%, while the maximum throughput gain for well-connected receivers can be as high as 14x, and the maximum throughput gain for the "crying baby" itself can be as high as 5.4x.

In summary, we make the following contributions:

- We address the problem of high-throughput, reliable multicast for file download applications in deployed WMNs. We identify two challenges in supporting high-throughput, reliable multicast (i.e. 100% PDR) in deployed WMNs: high packet loss rates [24], and the "crying baby" problem. In particular, to our best knowledge, *this is the first work that addresses the well-known in the wired Internet "crying baby" problem in the context of file download applications in multihop WMNs*. The requirement for 100% PDR, posed by the nature of the application into consideration, makes the problem even more challenging, and existing solutions [13] not applicable.
- We propose *Pacifier*, the first practical multicast protocol that simultaneously addresses both challenges: it guarantees 100% PDR for all multicast receivers, while simultaneously solving the "crying baby" problem by offering significant throughput improvements for both well-connected and poorly-connected receivers over a state-of-the-art protocol.
- We present the design of *Pacifier* which seamlessly integrates four building blocks, namely, *tree-based OR, intra-flow NC, source rate limiting, and round-robin batching*.

While the design of *Pacifier* is based on the numerous principles and techniques developed over the past fifteen years in the field of reliable multicast, the novelty of *Pacifier* is in the use of NC to gracefully integrate all four building blocks to develop a full-fledged multicast protocol.

- We present extensive simulations with a realistic physical model showing that *Pacifier* offers significant throughput improvements over the state-of-the-art MORE, and, unlike MORE, it solves the "crying baby" problem. Starting with a basic version of *Pacifier* and adding one building block at a time, we show the additional benefits of each building block.
- We present an application-layer implementation of *Pacifier* and MORE and their evaluation on a 22-node 802.11 WMN testbed deployed in two academic buildings on the Purdue University campus. Our testbed experiments confirm the simulation findings.
- To facilitate further research of this subject, we have made the source code of our implementation publicly available at https://engineering.purdue.edu/MESH/pacifier/. The software has been downloaded by over 20 research groups from 6 countries since September 2009.

II. Related Work

In spite of the extensive research on reliable multicast in the wired Internet, which went through the development of ARQ-based schemes (e.g., [25], [23]), to FEC schemes (e.g., [26]), to hybrid ARQ-FEC schemes (e.g., [27], [20], [28]), to rateless codes [29], [30], [31], [32], the majority of the work on reliable multicast in multihop wireless networks have used the traditional ARQ techniques. A survey on reliable multicast protocols for ad hoc networks [33] classifies them into deterministic and probabilistic ones, depending on whether data delivery is fully reliable or not. Deterministic protocols (e.g., [16], [34], [35], [15], [36], [18]) provide deterministic guarantees for packet delivery ratio, but they can incur excessive high overhead and drastically reduced throughput. On the other hand, probabilistic protocols (e.g., [11], [12]) incur much less overhead compared to the former, but they do not offer hard delivery guarantees. Using rateless codes requires the source to continuously send packets, which can cause congestion in the bandwidth-limited wireless networks. Recently, [19] studied the applicability of FEC and hybrid ARQ-FEC techniques, borrowed from the wired Internet, to WMNs, and showed that RMDP [20], a hybrid ARQ-FEC protocol, can provide both reliability and high throughput.

Most recently, intra-flow network coding has been proposed as a whole new approach to reliable routing. NC in theory is equivalent to hop-by-hop FEC [21], [22], and hence the maximum amount of redundancy injected from any node in the network is determined by the lossiest link of the tree, and not by the lossiest path from the source to any receiver, unlike in end-to-end FEC. However, hop-by-hop FEC/NC also has its practical drawbacks; it requires buffering packets at each node for decoding/re-encoding (in case of FEC) or only re-encoding (in case of NC). Due to the constraints on the buffer size and on packet delay, NC needs to send packets in batches, i.e., the source needs to wait till a batch is received by all receivers before proceeding to the next batch. This introduces the "crying baby" problem, where the poorly connected receivers slow down the completion time of well-connected receivers.

To our best knowledge, MORE is the only NC-based protocol for high-throughput, reliable multicast routing (though it is also for unicast). Due to its significance, and since we will compare *Pacifier* against it in our evaluation, we present a brief overview of MORE in Section II-A. To our knowledge, the only other practical NC-based multicast protocol is CodeCast [14], which exploits NC for *improving* but not *guaranteeing* reliability in multimedia multicast applications in mobile ad hoc networks.

A. Overview of MORE

MORE [6] is an OR protocol for reliable file transfer. MORE is implemented as a shim between the IP and the 802.11 MAC layer. In the following, we describe the main functions of MORE, focusing on its multicast operation. We briefly review its two major features: forwarding node (FN) selection and packet batching.

FN selection. MORE uses the ETX metric [37], based on loss rate measurements, to select the possible FNs. ETX is equal to the expected number of transmissions required to successfully transmit a packet from the source to a destination. For each destination the source includes in the FN list the nodes whose ETX distance to that destination is shorter than the source's distance. Also, for each FN the source includes a *TX_credit* in the FN list. The *TX_credit* is the expected number of transmissions a node should make for every packet it receives from a node farther from a destination in the ETX metric, in order to ensure that at least one node closer to the destination will receive the packet.

The algorithm for FN selection and TX_credit calculation is run at the source. The algorithm starts by assuming that *every* node is a candidate FN for a source-destination pair and calculates the expected number of transmissions this node would make. It then prunes nodes that are expected to perform less than 10% of the total transmissions and assigns TX_credits to the remaining ones, which form a belt of FNs that connect the source to the destination. The algorithm is repeated for each destination; in the end the belts formed for each destination are merged into the final FN set. If an FN belongs to more than one belts, (i.e., for more than one destination), the algorithm calculates a different expected number of transmissions for each of the belts it belongs to. Its final TX_credit is then calculated using the maximum number of transmissions among these belts.

Batching and Coded Packet Forwarding. In MORE, the source breaks a file into batches of k packets. Whenever the MAC is ready to send a packet, the source creates a random linear combination of the k packets of the current batch and broadcasts the encoded packet. Each packet is augmented with its code vector, the batch ID, the source and destination IP addresses and the list of FNs for that multicast, with their TX_credits.

Packets are broadcast at the MAC layer, and hence they can be received by all nodes in the neighborhood. When a node hears a packet, it checks if it is in the packet's FN list. If so, the node checks if the packet is *linearly independent* with all the packets belonging to the same batch that it has already received. Such packets are called *innovative packets* and are stored in a buffer. Non-innovative packets are discarded. Every time a node receives a packet from an upstream node, it increments its *credit_counter* by its assigned TX_credit included in the packet header. If its *credit_counter* is positive, whenever the MAC is ready to send a packet, the node creates a linear combination of the innovative packets it has received so far and broadcasts it. Broadcasting a packet decrements the *credit_counter* by one unit.

A multicast receiver decodes a batch once it collects k innovative packets from that batch. It then sends an ACK back to the source along the shortest ETX path in a reliable manner. The source keeps sending packets from the same batch until all receivers have decoded and acknowledged the current batch; it then proceeds to the next batch. Whenever a receiver acknowledges the current batch, the source removes the FNs responsible for forwarding packets only towards that receiver and recalculates the credits for the remaining FNs, using the maximum number of transmissions taken only over FN belts to receivers that have not yet acknowledged the batch.

III. *Pacifier* DESIGN

The design of *Pacifier* addresses several weaknesses of MORE. In particular, the belt-based forwarding in MORE can be inefficient for multiple receivers, MORE lacks source rate limiting which can lead to congestion in data dissemination, and MORE suffers the "crying baby" problem.

For clarity, we present the design of *Pacifier* in several steps. We first present a basic version of *Pacifier*, which consists of several building blocks: tree-based opportunistic multicast routing, batching and NC-based forwarding, and credit calculation. The basic version guarantees reliability and already increases throughput compared to MORE, but does not solve the "crying baby" problem. We then present two optimizations: source rate limiting which reduces congestion and further improves the throughput, and round-robin batching, which solves the "crying baby" problem.

A. Tree-based Opportunistic Routing

We argue that the use of OR in the form used in MORE is an overkill for multicast and it can lead to congestion, for two reasons. First, even for a single destination, congestion can occur if too many nodes act as FNs, or if the FNs are far from each other and they cannot overhear each other's transmissions [38]. The situation is worsened when the number of flows increases, since almost all nodes in the network may end up acting as FNs. Such performance degradation was observed in the evaluation of MORE in [6] for many unicast flows; the situation for many hypothetical unicast flows is not very different from a source to many multicast receivers. Second, the benefit of overhearing of broadcast transmissions, which is exploited by OR in MORE, is naturally exploited in a fixed multicast tree, where the use of broadcast allows nodes to receive packets not only from their parent in the multicast tree, but also from ancestors or siblings, essentially transforming the tree into a mesh. We note this property of opportunistic reception of broadcast transmissions has been previously exploited in the design of some of the first multicast protocols for multihop wireless networks (e.g., ODMRP [39]), for improving the PDR.

The above observation motivates a simple multicast-tree based OR design. Specifically, *Pacifier* starts by building a multicast tree to connect the source to all multicast receivers. The tree is a shortest-ETX tree, constructed at the source by taking the union of all the shortest-ETX paths from the source to the receivers, which in turn are based on periodic loss rate measurements.The multicast tree is reconstructed at the source every time the number of active receivers changes.

1) Batching and Coded Forwarding: As in MORE, the source and the FNs in *Pacifier* use *intra-flow* random linear NC. The source breaks a file into small batches of packets and sends random linear combinations of the packets belonging to the current batch. Intermediate FNs store all the innovative packets of the batch and also send random linear combinations of them. We selected a batch size of $k = 32$ packets, same as in [6], [40]. The random coefficients for each linear combination are selected from a Galois Field of size 2^8, again same as in [6]. When a receiver receives any k linearly independent coded packets of a batch, it decodes the batch and sends an ACK back to the source along the shortest ETX path in a reliable manner.

To achieve reliability, this basic version of *Pacifier* uses the following *batch termination scheme*: the source keeps transmitting packets from the same batch, until *all* the receivers acknowledge this batch. Such a transmission scheme however introduces the "crying baby" problem as the completion time of each batch is limited by that of the worst receiver.

2) How many packets does an FN send?: Despite the use of a multicast tree for data forwarding, the use of 802.11 broadcast effectively enables opportunistic routing, i.e., a node can opportunistically receive packets from nodes other than its parent in the multicast tree. If a node forwards every packet it receives, a receiver could potentially receive each packet originated from the source multiple times. To avoid unnecessary transmissions, we need to carefully analyze *how many (coded) packets an FN should send upon receiving a data packet.*

Our solution is inspired by the approach used in MORE, and is based on the notion of TX_credits. Since, in practice, an FN should be triggered to transmit only when it receives a packet, we derive the number of transmissions each FN needs to make for every packet it receives. We define this number as the TX_credit for that FN. Thus, in *Pacifier*, an FN node j keeps a credit counter. When it receives a packet from an *upstream* node (defined below), 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. We note that opportunistic reception of data packets is always allowed, even from downstream nodes. The credit calculation is on how many packets to be transmitted by the FN upon

receiving a data packet from an upstream node.

In the analysis, we focus on disseminating one data packet from the root down the multicast tree. Our analysis is based on the simple principle that in disseminating a packet from the root, each FN in the multicast tree should ensure that each of its child nodes receives the packet *at least once*. Note this principle slows down a parent node to wait for the worst child and creates the "crying baby" problem at each FN, but is consistent with the batch termination scheme of this basic version of *Pacifier*.

We assume an FN j sends packets after receiving from any nodes with lower ETX distance from the root to them, i.e., j's upstream nodes. These nodes are likely to receive packets from the root before j.[1] We also assume that wireless receptions at different nodes are independent, an assumption that is supported by prior measurements [41].

Let N be the number of FNs in the multicast tree rooted at s. Let ϵ_{ij} denote the loss probability in sending a packet from node i to node j. Let z_j denote the expected number of transmissions that FN j must make in disseminating one packet (from the root) down the multicast tree. Let $C(j)$ denote the set of child nodes of j in the multicast tree, and $A(j)$ denote the set of j's upstream nodes.

The expected number of packets that j receives from ancestor nodes is $\sum_{i\in A(j)} z_i(1-\epsilon_{ij})$. Recall j's objective is to make sure each of its child nodes receives at least *one* packet. Since each child node $k \in C(j)$ has already overheard $\sum_{i\in A(j)} z_i(1-\epsilon_{ik})$ from node j's ancestors, the amount of packets node j actually needs to forward for child k is:

$$L_{jk} = min(\sum_{i\in A(j)} z_i(1-\epsilon_{ij}), 1) - \sum_{i\in A(j)} z_i(1-\epsilon_{ik}) \quad (1)$$

The min operation ensures that j does not forward the same packet more than once, in case it receives it from more than one FNs. Note for the source node s, $L_{sk} = 1$ for all $k \in C(s)$.

Since the expected number of times node j has to transmit a packet to ensure that its child k will receive one packet is $\frac{1}{1-\epsilon_{jk}}$, the expected number of transmissions of j for child k to receive L_{jk} is:

$$z_{jk} = \frac{L_{jk}}{1-\epsilon_{jk}} \quad (2)$$

Since packets are broadcast, they can be received by more than one child nodes at a time. Hence, the expected number of transmissions node j has to make to ensure that each child node has *one* packet is:

$$z_j = max_{k\in C(j)} z_{jk} \quad (3)$$

z_j and L_{jk} are inter-dependent, and can be calculated recursively in $O(N^2)$ operations, i.e., by traversing the FNs in the increasing order of their ETX values from the source. Since the order of FNs is well-defined, there are no loops in the credit calculation.

[1]In contrast, MORE's credit calculation was based on the ordering of FNs according to their ETX distance to the destination node. It is unclear that nodes with larger ETX distance to the destination will receive the packet from the root sooner.

For each data packet the source sends down the multicast tree (which may require multiple transmissions), FN j receives $\sum_{i\in A(j)} z_i(1-\epsilon_{ij})$. Thus, the TX_credit of node j is:

$$\text{TX\_credit}_j = \frac{z_j}{\sum_{i\in A(j)} z_i(1-\epsilon_{ij})} \quad (4)$$

A fundamental difference between the TX_credit calculation in MORE and in *Pacifier* is that the latter decouples the credit calculation from the routing process. Indeed, in *Pacifier*, we first build a multicast tree and then calculate the TX_credits only for those FNs that are part of the tree. In contrast, in MORE, FN selection and TX_credit calculation are tighly coupled; TX_credits are calculated for the whole network and then some FNs are pruned based on this calculation. As we will show in Section IV, this decoupling in *Pacifier* significantly improves the efficiency of both procedures.

B. Source Rate Limiting

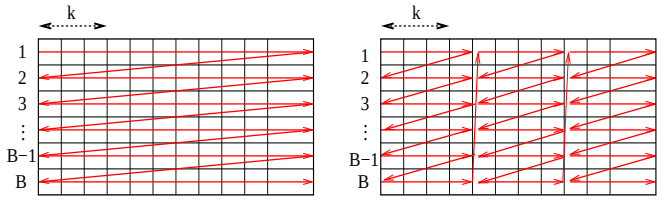
Recent studies have shown the importance of adding rate control to NC-based unicast routing protocols, which exploit MAC layer broadcast [40], [42], [38], [43]. However, end-to-end rate control in multicast is much more complex than in unicast, and there is no widely accepted solution so far. In the version of *Pacifier* presented so far, the use of TX_credits implements a form of rate control at which each intermediate FN injects packets into the network. However, the source can potentially send out all the packets in a batch unpaced.

To add rate control to the source, we exploit the broadcast nature of the wireless medium and apply a simple form of backpressure-based rate limiting, inspired by BMCC [13]. The basic idea is to have the source wait until it overhears its child nodes forward the previous packet it sent before it transmits the next packet. Since the number of transmissions by the source z_s has already factored in packet losses to its child nodes, the source does not need to worry about losses of individual transmissions, i.e., it does not need to wait until all its child nodes forward each packet it sends out. In fact, it is not even sure that every of its transmissions will trigger a transmission at each of its child nodes, as some nodes may have negative credit counters. Instead, the source waits until it overhears a transmission from *any* of its child nodes or until a timeout before it sends the next packet in the batch.

The work in [13] does not discuss how to set the timeout. In [44], the authors suggested a heuristic timeout of $3\times T_p$ for the backpressure-based unicast version of BMCC, where T_p is the transmission time of one data packet, which depends on the packet size and the MAC data rate. The factor of 3 is to account for the contention time preceding each transmission. Following the same reasoning, in *Pacifier*, we set the timeout to $\sum_{j\in C(s)} \text{TX\_credit}_j \times 8 \times T_p$. This choice for the timeout reflects the fact that in *Pacifier* a transmission from the source will trigger on average $\sum_{j\in C(s)} \text{TX\_credit}_j$ transmissions from its child nodes, which in the worst case can be sequential, and also the fact that in multicast contention near the source is in general higher.

C. Solving the "Crying Baby" Problem

In MORE, the source keeps transmitting packets from the same batch until all the receivers acknowledge that batch, as



(a) Sequential batch transmission in MORE. Each batch is acknowledged by *all* the receivers before the source moves to the next batch.

(b) Round-robin batch transmission in *Pacifier*. The source moves to the next batch every time *one* receiver acknowledges the current batch.

Fig. 1. Two different ways of transmitting B batches of k original packets each: sequential (as in MORE), and round-robin (as in *Pacifier*). For better visualization, we assume here (not true in the actual operations of the protocols) that the same total amount of redundancy is required to be sent for each batch.

shown in Figure 1(a). This policy makes the protocol susceptible to the "crying baby" problem, since if the connection to one receiver is poor, that receiver can slow down the rest of the receivers. The basic version of *Pacifier* we have described so far suffers from the same problem.

As an example, assume the source multicasts a file consisting of B batches to two receivers $R1$ and $R2$. $R1$ can download a batch (i.e., collect the required number of linearly independent coded packets and decode the batch) in time T, and $R2$ can download a batch in time $T' >> T$. With MORE, $R1$ will remain idle for time $T'-T$ after decoding each batch and for a total time of $B \cdot (T' - T)$, and both receivers will finally complete the file download in time $B \cdot T'$. We want a solution that would allow the well-connected receiver $R1$ to complete the download in time $B \cdot T$ and the poorly-connected receiver $R2$ (i.e., the crying baby) in time no more than $B \cdot T'$.

Note the problem would not exist if the whole file could be encoded into one batch. However, such an approach is not realistic due to the prohibitively high computational overhead (associated with coding operations), header overhead (from including the random coding coefficients in packet headers) and memory requirement at the intermediate routers. In the following, we describe a practical solution to the problem, which requires no more memory than MORE or our basic version, i.e., FNs still maintain only one batch at a time in their memory.

In the proposed scheme, the source iteratively sends the batches of a file in a *round-robin* fashion, for as many rounds as required, until it has received ACKs for all batches from all the receivers, as shown in Figure 1(b). In detail, the source maintains a counter C_{s_i} for each batch i which is equal to the number of remaining packets the source has to transmit for that batch. The counter for batch i is initialized as $C_{s_i} = z_s \times k$, where z_s is calculated in Equation (3) and k is the batch size, and it is decremented every time a packet from batch i is transmitted. Each intermediate FN forwards coded packets according to its TX_credit, and only buffers packets belonging to the current batch; when it receives the first packet from a new batch, it flushes its buffer and starts buffering packets from the new batch.

The source determines when to switch to work on the next batch as follows. It sends packets from batch i until either (1) C_{s_i} reaches zero or (2) it receives from *one* receiver acknowledging completion of this batch; it then moves to the next batch for which there are still receivers that have not acknowledged it. When the source finishes with the last batch B, it starts the next round by going back to the first batch for which it has not received ACKs from all receivers. For each such batch it revisits, it recalculates the multicast tree (i.e., the FNs) and the TX_credit values for the FNs based on the receivers that have not sent ACKs and resets $C_{s_i} = z_s \times k$ using the newly calculated z_s.

The proposed approach effectively addresses the "crying baby" problem. Back to our example, the source now stays at each batch only for time T, i.e., until it receives an ACK from receiver $R1$. After a total time of $B \cdot T$, $R1$ completes the whole file and leaves the multicast tree. The user can now disconnect from the Internet and watch the movie or install the software he/she just downloaded, or join another multicast group and download a different file, or start a new unicast session (e.g., browsing the web). The source recalculates the multicast tree, keeping only those FNs responsible for $R2$, and starts a second round, spending an additional $T' - T$ time at each batch. At the end of the second round, $R2$ also completes the download and the total time for $R2$ is $B \cdot T + B \cdot (T' - T) = B \cdot T'$.

In practice, things are a bit more complicated and the batch switching policy is critical to achieving high throughput for both well- and poorly connected receivers. On one hand, if one receiver has already acknowledged the current batch before the source sends all the scheduled packets for that batch, not moving to the next batch at the source will reduce the throughput of that receiver. On the other hand, after all the scheduled packets have been sent out, allowing the source to move to the next batch only when it receives an ACK from one receiver can be inefficient, as ACKs may delay to reach the source, due to congestion, or because they have to traverse several hops to reach the source. This could result in redundant packet transmissions from the current batch at the source while waiting for an ACK. Instead, immediately moving to the next batch after finishing the scheduled packets, ensures that the "network pipe" is always filled with useful packets, and delayed ACKs will have little impact on the performance. Our evaluation in Sections IV-B, V-D shows that switching batch when *either* of the two conditions is satisfied results in significant throughput improvements over MORE for both well-connected receivers and the worst ones (i.e., the "crying babies"). We note previously [45] also noticed this "stop-and-wait" policy (also used in MORE) can result in significantly low throughput, in the context of unicast, as the network scales.

A round-robin batching scheme was also used in Fcast [28], an FEC-based protocol for the wired Internet. However, there are two major differences between the two protocols. First, the use of NC in *Pacifier* eliminates duplicate packet transmissions; the source *sends different random combinations of the original k packets in every round*. In contrast, in Fcast, each batch of k packets is pre-encoded to produce a fixed number of $n > k$ packets (typical values for k, n, are 32 and

255, respectively). Once the source finishes the transmission of all n encoded packets from each of the B batches, it starts a new round where it *retransmits again the same* n *encoded packets* for each batch. Under high packet loss rates, observed in WMNs [19], this policy may result in some receivers receiving duplicate packets and may further delay decoding. While rateless codes (e.g., [46]) can indeed help to avoid duplication as in these codes n is not limited, a second difference still remains. In *Pacifier*, both the source and the FNs perform coding operations, which allows the protocol to exploit the benefits of OR without the burden of coordination overhead. In contrast, in Fcast (even with a rateless code), only the source performs coding operations. This limits the efficiency of OR with Fcast, since FNs need a coordination mechanism [4] to avoid duplicate transmissions.

1) Adjusting TX_credit Calculation: In the basic version of *Pacifier* (Section III-A2), we defined the TX_credit of an FN as the expected number of packets it has to transmit for every packet it receives from its upstream nodes, in order to ensure that *all* of its child nodes will receive one packet. This definition is consistent with the batch termination scheme of the basic scheme, i.e., the source completes a batch when it receives ACKs from all receivers. However, it is inconsistent with the round-robin batching scheme, which aims to prevent poorly connected receivers from slowing down well-connected receivers. Hence under the round-robin batching, we adjust the definition of TX_credit of an FN to be the expected number of packets it has to transmit for every packet it receives from its upstream nodes, in order to ensure that *at least one* of its child nodes will receive one packet. To realize this change, we simply change the max operator to min in Equation (3). We note this new definition is also consistent with the policy of moving to the next batch whenever any receiver acknowledges the current batch.

2) Intricacies in TX_credit Calculation: There is a subtlety in the above adjustment to the TX_credit calculation under the round-robin batching scheme, i.e., changing the max operator to min in Equation (3). The derivation of Equation (3) is based on expected number of opportunistic packet receptions (based on the ETX measurements). However, in the actual dissemination of any given batch i, it is possible that the actual packet reception is below or above the expected value. In the later case, the best receiver will successfully receive all packets for that batch, and it is the correct thing to do for the source to move on to the next batch. However, in the former case, the best receiver could be a few packets short of receiving the whole batch i, and hence if the source moves on to the next batch, even the best receiver has to wait for a whole round before the source transmits again packets from batch i. On the other hand, if we had let the source send some additional packets to those predicted by Equation (3), there is a good chance that the best receiver would have finished in the current round; this would increase the throughput of the best receiver. The challenge here is that it is unknown beforehand whether the opportunistic reception in any particular batch is above or below the expectation, and hence those extra packets sent by the source for a batch can potentially elongate each batch and reduce the throughput of the best receiver.

To facilitate studying the above subtlety in the TX_credit calculation under the round-robin batching scheme, we introduce a tunable knob in Equation (3). Essentially, we define the expected number of transmissions node j makes to its child nodes as $z_j = min_{k \in C(j)} z_{jk} + knob * (max_{k \in C(j)} z_{jk} - min_{k \in C(j)} z_{jk})$. Setting $knob$ to 1 changes the objective to ensuring all child nodes receive a packet at least once, while setting $knob$ to 0 changes the objective to ensuring at least one child node receives a packet at least once. In Section IV-B5, we evaluate the impact of this knob by comparing the performance of *Pacifier* under different values of $knob$.

IV. Simulation Studies

We first evaluate the performance of *Pacifier* by comparing it against MORE using extensive simulations. The use of a simulator allowed us to evaluate the performance of the two protocols in large networks, using a diverse set of topologies, which are difficult to create in a testbed. We note *Pacifier* uses the same type of NC and has the same memory requirements and the same fields in the packet header as MORE,[2] and hence it can be easily implemented in practice. We present an implementation study of *Pacifier* in the next section.

A. Evaluation Methodology

Simulation Setup. We used the Glomosim simulator [47], a widely used wireless network simulator with a detailed and accurate physical signal propagation model. Glomosim simulations take into account the packet header overhead introduced by each layer of the networking stack, and also the additional overhead introduced by MORE or *Pacifier*. For the implementation of MORE, we followed the details in [6].

We simulated a network of 50 static nodes placed randomly in a $1000m \times 1000m$ area. The average radio propagation range was 250m, the average sensing range was 460m, and the channel capacity was 2Mbps. The *TwoRay* propagation model was used. To make the simulations realistic, we added fading in our experiments. The Rayleigh model was used, as it is appropriate for WMN environments with many large reflectors, e.g., walls, trees, and buildings, where the sender and the receiver are not in Line-of-Sight of each other. Because of fading, the probability for a node to hear/sense another node decreases with the distance and there is no clear cut off. For example, at a distance of 250m, the probability of hearing a neighbor node is very low. Although sometimes nodes can hear each other even in distances larger than 250m, in most cases, link quality is very low for distances larger than 150m.

We simulated each protocol on 10 different randomly generated topologies (scenarios), i.e., placement of the 50 nodes. For each scenario, we randomly generated a multicast group consisting of 1 source and 9 receivers. The source sent a 12MB file, consisting of 1500-byte packets, transmitting at the maximum rate allowed by the MAC (in case of MORE) or by the MAC and the backpressure-based rate limiting (in case of *Pacifier*). We present the result for each scenario and the average result over all 10 scenarios.

[2] *Pacifier* only includes the list of FN nodes in the header, sorted in increasing ETX distance from the source. It does not require information about the edges of the tree.

TABLE I
VERSIONS OF MORE AND *Pacifier* EVALUATED IN OUR STUDY. ALL VERSIONS INCLUDE INTRA-FLOW NC.

Name	Description
MORE	MORE [6] optimized with scenario-specific pruning threshold
TREE	Tree-based OR
TREE+RL	Tree-based OR, source rate limiting
TREE+RL+RRB (*Pacifier*)	Tree-based OR, source rate limiting, and round-robin batching

Following the methodology in [6], [4], we implemented an ETX measurement module in Glomosim which was run for 10 minutes prior to the file transfer for each scenario to compute pairwise delivery probabilities. During these 10 minutes, each node broadcasts a 1500-byte packet every second, and keeps track of the packets it receives from its neighbors. At the end of the 10-minute duration, all the measurements are distributed to all the nodes. The source uses these measurements to compute the forwarding lists and the transmission credits for the two protocols. There was no overhead due to loss rate measurements during the file transfer.

Evaluation Metrics. We used the following metrics:

Average Throughput: The file size (in bytes) divided by the total time required for a receiver to collect the necessary number of packets for decoding, averaged over all receivers.

Total number of data packet transmissions:[3] The total number of data packets broadcast by the source and the FNs.

Source Redundancy: The total number of encoded data packets sent by the source divided by the file size. It gives an estimate of the redundancy injected in the network by the source.

Download completion time: The total time required for a receiver to collect the necessary amount of coded packets to decode all the batches and recover the complete file.

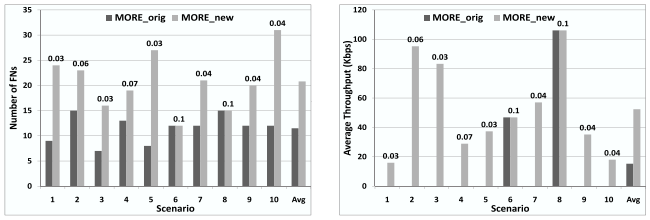
Note that we did not use the PDR as a metric, since both protocols *guarantee* 100% PDR.

B. Simulation Results

We start by optimizing MORE's pruning strategy as the default strategy appears to cause frequent network partition. We then proceed to evaluate the incremental performance benefit of *Pacifier*'s major components, i.e., the basic version, adding source rate limiting, and adding round-robin batching. Table I summarizes the different versions of MORE and *Pacifier* evaluated.

1) Fixing MORE's pruning threshold: Recall from Section II-A that MORE prunes FNs that are expected to perform less than 10% of the total number of transmissions. We found using such a pruning threshold can result in disconnection of some receivers. The probability for this to happen naturally increases with the network size, since the larger the number of nodes acting as FNs, the smaller the expected number of transmissions each of them has to make. Recall also that in MORE, the source proceeds to the next batch only when all receivers acknowledge the current batch. When a receiver is disconnected, the source will never leave the first batch, and all the receivers will receive zero throughput.

[3]The number of control packets (ACKs) is the same for both MORE and *Pacifier*, equal to $N \times B$, where N is the number of receivers and B is the number of batches the file is broken into.



(a) Number of FNs (b) Throughput

Fig. 2. Number of FNs and throughput with default pruning threshold (MORE_orig) and the largest pruning threshold that does not cause any disconnection (MORE_new), for 10 different scenarios. For MORE_new, the labels above the bars show the pruning threshold used for each scenario.

One solution to the problem is to use a much lower pruning threshold than 0.1. However, using a very low threshold can lead to too many FNs in dense WMNs which increases the contention for the channel. To be fair in our evaluation and not cause performance degradation for MORE, we used the following approach, which favors MORE, instead of a common threshold for all 10 scenarios: for each scenario, we repeated the simulation for different values of the pruning threshold α, starting with the default value of 0.1, and lowering it by 0.01 until no receiver was disconnected. This last value was the one we used for the comparison against *Pacifier*.

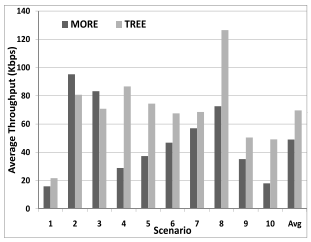
Figure 2 shows the number of FNs, and the throughput, with the default threshold of 0.1 (MORE_orig), and with the best threshold for each scenario (MORE_new), in each of the 10 scenarios. Figure 2(a) shows that using the default threshold resulted in a very low number of FNs; on average only 11.2 FNs were used in a network of 50 nodes. However, this low number of FNs caused disconnection of at least one receiver and resulted in zero throughput in 8 out of 10 scenarios, as shown in Figure 2(b). For the 10 scenarios studied, the largest pruning threshold that does not cause any disconnection varies from 0.1 to 0.03.

In the following, we compare various versions of *Pacifier* to MORE_new. For simplicity, we will call it MORE.

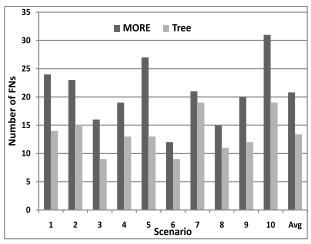
2) Impact of tree-based OR: We start the evaluation of *Pacifier* by examining the impact of its tree-based OR, by comparing the basic version of *Pacifier* (TREE), with MORE. The only difference between the protocols is the algorithm used for selecting FNs and assigning TX_credits to them. The results for 10 different scenarios are shown in Figure 3.

Figure 3(a) shows TREE achieves higher throughput than MORE in 8 out of 10 scenarios. The gain ranges from 20% (Scenario 7) up to 199% (Scenario 4), with an average throughput gain over all 10 scenarios equal to 42%. Only in two scenarios (2 and 3), there is a small throughput reduction with TREE, about 16%.

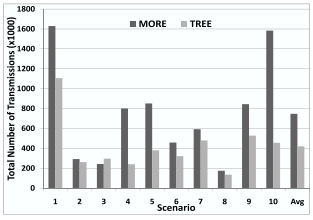
The higher throughput achieved by TREE compared to MORE can be explained by the fewer FNs and lower total number of transmissions in the former compared to the latter. In particular, Figure 3(b) shows that the use of a tree instead of a union of belts results in on average 36% fewer FNs in TREE than in MORE. Note that in some cases, TREE uses equal or



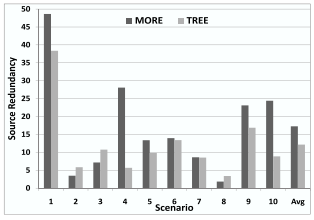
(a) Throughput



(b) Number of FNs

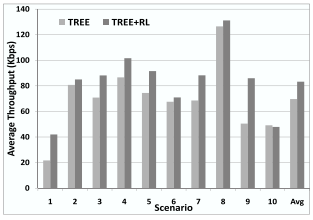


(c) Total # of Transmissions

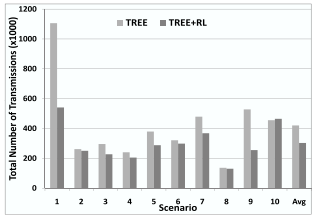


(d) Source Redundancy

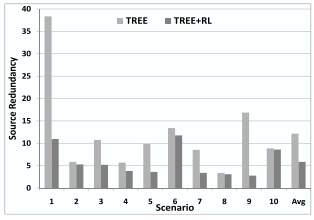
Fig. 3. Throughput, number of FNs, total number of transmissions, and source redundancy with MORE and TREE for 10 different scenarios.



(a) Throughput



(b) Total # of Transmissions



(c) Source Redundancy

Fig. 4. Throughput, total number of transmissions, and source redundancy with MORE and TREE+RL for 10 different scenarios.

even fewer FNs compared to MORE with the default pruning threshold (e.g., in Scenarios 2, 4, 6). However, the FN selection algorithm ensures that no receiver is disconnected from the source, unlike in MORE, since there is no random, threshold-based pruning. Figure 3(c) shows the use of a tree combined with the new algorithm for TX_credit calculation results in on average 44% reduction in the total number of transmissions in TREE, compared to MORE. Finally, Figure 3(d) shows MORE has a high source redundancy; the source sends on average 17 times the file size. TREE reduces the average source redundancy to 12. The difference in source redundancy suggests TREE is more efficient in selecting FNs and more accurate in calculating the TX_credit values for the FNs.

3) Impact of source rate limiting: We next evaluate the impact of backpressure-based rate limiting at the source, as implemented in the TREE+RL version of *Pacifier*. Figure 4(a) shows that the use of rate limiting at the source improves the throughput by 5% (Scenario 6) to 94% (Scenario 1), with an average of 20%, compared to TREE. Figure 4(c) shows that TREE+RL on averages reduces the source redundancy to 5.84, a 52% reduction compared to the value of 12.15 for TREE. The reduction in the source redundancy in turn reduces the total number of transmissions by 28% on average, as shown in Figure 4(b). We found that this reduction comes not only from the contribution of the source but also from the majority of the FNs. This confirms that, by pacing the source's transmissions, the source's children and grandchildren get better chances to successfully transmit packets and make progress down the tree.

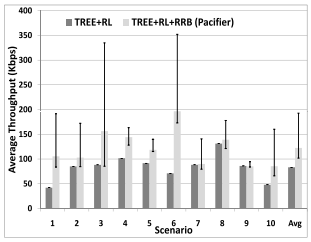


Fig. 5. Throughput with TREE+RL and TREE+RL+RRB (*Pacifier*) for 10 different scenarios. The error bars show throughput of the best and the worst receiver.

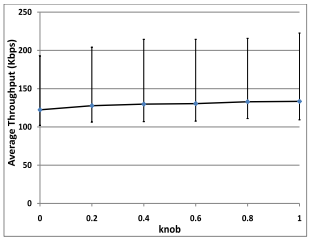
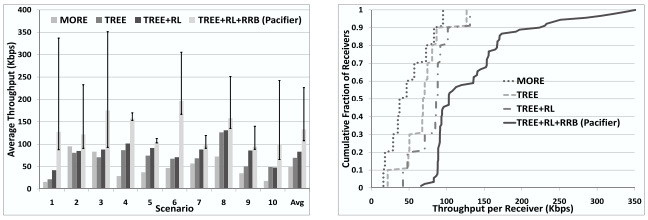


Fig. 6. Average throughput with *Pacifier* as a function of $knob$, over 10 scenarios. The error bars show average max and min values over the 10 scenarios.

4) Solving the "crying baby" problem: The above results have shown that TREE and TREE+RL already offer significant throughput improvement over MORE. However, these two versions of *Pacifier* still suffer from the "crying baby" problem. We next evaluate the effectiveness of round-robin batching on solving the "crying baby" problem, by comparing TREE+RL+RRB (the complete *Pacifier* protocol) with TREE+RL.

Figure 5 shows the average throughput achieved with TREE+RL+RRB and TREE+RL in each of the 10 scenarios, as well as the throughput of the best and the worst receiver (top and bottom of error bars) in each scenario under TREE+RL+RRB. We make three observations. First, with TREE+RL, which uses sequential batch transmission, all 9 receivers in each scenario achieve the same throughput, which is determined by the worst receiver. In contrast, with TREE+RL+RRB, well-connected receivers get much higher throughput than the average, as shown by the large gap between the top of the error bars and the average in most scenarios. Averaging over 10 scenarios, the best receiver achieves 58% higher throughput than the average throughput by all receivers. Second, allowing receivers to proceed independently in TREE+RL+RRB also increases the average throughput by 47% on average over all 10 scenarios, compared to TREE+RL. Third, importantly, the throughput improvement for the best receivers comes at almost no penalty to the worst receivers. In particular, compared to with TREE+RL, the throughput of the worst receiver with TREE+RL+RRB gets slightly worse in 3 scenarios (Scenario 7, 8, and 9 by 10%, 7%, and 3%, respectively), remains unaffected in 2 scenarios (Scenarios 2 and 3), and increases by 26%-146% for the remaining 5 scenarios.



(a) Average, max, and min throughput with each protocol for each of the 10 scenarios.

(b) CDF of the 90 throughput measurements obtained with each protocol for 10 scenarios with 9 receivers each.

Fig. 7. Overall throughput comparison of MORE, TREE, TREE+RL, and TREE+RL+RRB (*Pacifier*).

5) Tuning the knob in TX_credit Calculation: Finally, we study the intricacies in calculating TX_credit values by varying the $knob$ value introduced in Section III-C2. We vary the value of $knob$ from 0 (the version evaluated in Section IV-B4) to 1. Intuitively, as $knob$ increases, the throughput of the best receiver is expected to decrease and the throughput of the worst receiver is expected to increase, since we spend more time on each batch in every round.

Figure 6 shows the average, max, and min throughput with *Pacifier*, as $knob$ varies from 0 to 1. Every point is the average over 10 scenarios. $knob = 1$ improves the min throughput and maximizes the average, and, somewhat surprisingly, the max throughput as well. On the other hand, $knob = 0$ achieves the lowest max, average, and min throughput, compared to all the other $knob$ values. This confirms our speculation in Section III-C that setting $knob = 0$ may not give the best result as the TX_credit calculation is fundamentally based on the expected opportunistic receptions, and a lower than expected number of receptions in any given batch can cause the best receiver to be a few packets short of decoding a batch and wait for a whole round. In the remaining of the paper, we use $knob = 1$.

6) Overall Comparison: Figure 7(a) summarizes the average, maximum and minimum throughput comparison among MORE, TREE, TREE+RL, and TREE+RL+RRB (*Pacifier*), where TREE+RL+RRB used a $knob$ value of 1. We observe that on average, *Pacifier* outperforms TREE+RL, TREE, and MORE by 60%, 90%, and 171%, respectively. In addition, *Pacifier* allows well-connected receivers to achieve much higher throughput, which can be up to 20x higher than with MORE (for scenario 1), and also improves throughput of the worst receiver in all 10 scenarios, compared to the other 3 protocols.

Figure 7(b) depicts the same results in a different way. It plots the CDF of the 90 throughput values obtained from 10 scenarios with 9 receivers each, for the four protocols. In this figure, the CDFs for MORE, TREE, and TREE+RL have a staircase form, since for each scenario, all 9 receivers get roughly the same throughput (equal to that of the worst receiver) due to the "crying baby" problem. In contrast, with *Pacifier*, receivers finish independently of each other and the CDF has a continuous form. In the median case, *Pacifier* outperforms TREE+RL, TREE, and MORE by 20%, 49%, and 178%, respectively.

The benefit of *Pacifier* becomes more prominent if we look at the two ends of the CDF. *Pacifier* solves the "crying baby" problem by allowing good receivers to achieve very high throughput. The 90th percentile is 223Kbps for *Pacifier*, 70%, higher than with TREE+RL, 77% higher than with TREE, and 159% higher than with MORE. If we look at the 10th percentile, i.e., the worst receivers, we observe that *Pacifier* outperforms TREE+RL, TREE, and MORE by 80%, 300%, and 450%, respectively. This shows again that *Pacifier* not only solves the "crying baby" problem, it also simultaneously offers a significant improvement to the performance of the "crying baby" itself.

V. Protocol Implementation and Testbed Evaluation

In this section, we describe an implementation of *Pacifier* on a WMN testbed and present experimental results comparing *Pacifier* and MORE.

A. Testbed description

Our testbed, Mesh@Purdue (MAP) [48], currently consists of 22 mesh routers (small form factor desktops) deployed on two floors of two academic buildings on the Purdue University campus. Each router has an Atheros 5212 based 802.11a/b/g wireless radio operating in b ad hoc mode and attached to a 2dBi rubber duck omnidirectional antenna with a low loss pigtail. Each mesh router runs Mandrake Linux 10.1 (kernel 2.6.8-12) and the open-source *madwifi* driver [49] is used to enable the wireless cards. IP addresses are statically assigned. The testbed deployment environment is not wireless-friendly, having floor-to-ceiling office walls, as well as laboratories with structures that limit the propagation of wireless signals and create multipath fading.

B. Implementation details

NC-based wireless protocols (e.g., [6], [7]) are typically implemented as a shim between the IP and the MAC layer, i.e., at layer 2.5. Here, for ease of debugging, deployment, and evaluation, we implemented *Pacifier* at the application layer, using broadcast sockets, on Mandrake Linux 10.1 (kernel 2.6.8-12). For a fair comparison, we also implemented MORE at the application layer, following all the details in [6].[4] Although such an implementation unavoidably results in some performance degradation, compared to an implementation closer to the MAC layer, from crossing the kernel-user boundary, we note that this degradation is expected to be similar for both protocols, since they use the same type of network coding, they have the same memory requirements at the routers, and the same header fields.

Our implementation handles only synthetic traffic, i.e. data packets are generated within the MORE or *Pacifier* application, similarly as the implementation in [50], in which packets are generated within Click. The layer-2.5 header of MORE or *Pacifier* is part of the application layer packet payload. The source initially generates k random payloads for the current batch and mixes them every time it wants to transmit a packet.

[4]The publicly available implementation of MORE [50] using the Click modular router from the authors of [6] currently supports only unicast.

It then appends the MORE or *Pacifier* header and delivers the resulting packet to the IP layer, which in turn delivers the packet to the MAC for transmission. Packets are broadcast at the MAC layer, and every neighbor node can hear them. When a node receives a packet, it extracts and processes the protocol-specific header from the payload; if the node is an FN (i.e., it finds its ID[5] in the FN list in the header), it also uses the coding coefficients (also included in the header) to check for linear independence. If the received packet is innovative, the rest of the payload is stored for future mixing (if the node is an FN) or for decoding (if the node is a multicast receiver).

1) Dealing with queue sizes: In an ideal implementation at layer 2.5, a node running either MORE or *Pacifier* transmits a packet when (1) *the 802.11 MAC allows* and (2) *the credit counter is positive*. A layer-2.5 implementation [6] does not queue packets in the wireless card. Instead, innovative packets for the current batch are stored at a buffer. A pre-coded packet is always available awaiting for transmission. If another innovative packet is received before the pre-coded packet is transmitted, the pre-coded packet is updated by multiplying the newly received packet with a random number and adding it to the pre-coded packet. This approach ensures that every transmitted packet includes information from all the received innovative packets, including the most recent ones.

In our application layer implementation, we cannot get any feedback from the MAC, and hence, we have no control over the time a packet is transmitted. Instead, the application delivers packets to the IP when only the second condition holds and there is enough space in the socket buffer; from the IP layer, the packets are delivered to the wireless driver stored at the card's queue for transmission at a later time.

Since we have no control over a packet, once it leaves the application layer, we cannot update the packets buffered at the socket buffer or awaiting for transmission at the card's queue, if a new innovative packet is received. This inefficiency can have a significant impact on the performance of the two protocols. If a packet is queued either at the IP or at the MAC layer for a long time, it may not contain information from all the received packets so far. Even worse, the downstream nodes may have already received enough packets from the current batch, in which case the enqueued packets should not be transmitted at all. This is true in particular at the source which may create packets at a rate faster than the (actual) MAC's transmission rate. To avoid this problem with application-level implementation, we limit the socket buffer size to one packet and the card's queue length to three packets, so as to limit the time from the moment a packet is created at the application layer till the moment the packet is actually transmitted.

2) Dealing with end-to-end ACKs: In both protocols, a multicast receiver sends an end-to-end ACK back to the source every time it decodes a batch. It is critical for the performance of the protocols that these ACKs are propagated to the source in a fast and reliable way. In particular in MORE, loss of an ACK breaks the operation of the protocol, since the source only moves to the next batch when all receivers acknowledge the current batch. Similarly, delayed ACKs cause throughput degradation, since the source again cannot quickly move to the next batch. In *Pacifier*, the first problem does not exist, since even if no ACK is received for batch i, the source will eventually move to the next batch when the C_{s_i} counter reaches zero (Section III-C). However, delayed or lost ACKs can again significantly affect performance if the source unnecessarily spends time on batches that have already been decoded by all the receivers.

[5]To reduce the header overhead, we used 1-byte IDs instead of 4-byte IP addresses.

ACK reliability. To provide reliability, the ACKs in MORE are *unicast* at the MAC layer. In contrast to 802.11 broadcast mode, 802.11 unicast mode provides a reliability mechanism through acknowledgments and retransmissions. Unfortunately, there is an upper limit to the number of times a packet can be retransmitted at the MAC layer. For our Atheros wireless cards, this limit is 11. In our experiments, we found that 11 retransmissions were not always enough to deliver the packet to the next hop (especially under heavy traffic). Since this particular card does not allow changing this limit through *iwconfig*, we had to implement a simple but efficient reliability scheme at the application layer.

In our scheme, every node maintains an ACK cache, where it caches every ACK it transmitted, along with some meta data (the next hop of the path towards the source, the multicast group, the batch acknowledged by the ACK, and its status – "ACKed" or "not ACKed"). Nodes also remember the last ACK they forwarded for each multicast group. Every time a node transmits a data packet, it piggybacks information about the last ACK it received. This serves as an acknowledgment for the ACK to the ACK's previous hop. When the previous hop overhears a data packet acknowledging the ACK, it marks it as "ACKed" in the ACK cache. A node retransmits an ACK when (i) it overhears M packets from the ACK's next hop that do not acknowledge the ACK, or (ii) it overhears N packets from any node other than the ACK's next hop. We experimented with different values of M and N and finally selected $M = 10$, $N = 20$.

Fast ACK propagation. Similar to in [6], ACKs are sent to the source over the shortest ETX path to ensure quick propagation. In addition, in [6], ACKs are prioritized over data transmissions. In addition to ensuring fast ACK propagation, prioritizing ACKs over data packets is critical in our application layer implementation for one more reason. Since we have no control over a packet once it leaves the application layer, we have to guarantee that an ACK packet will never be dropped if the card's queue is full of data packets.

To implement ACK priority over data packets in our application layer implementation, we leveraged the TOS bits ("TOS field") of the IP header, which can be set using *setsockopt* at the application layer, and the priority properties in Linux routing [51]. The basic queuing discipline in Linux, *pfifo_fast*, is a three-band first-in, first-out queue. Packets are enqueued in the three bands based on their TOS bits. We set the TOS bits of the ACKs to 1010, which corresponds to "minimum delay + maximum reliability" (or "mr+md") and enqueues the ACKs in the highest priority band.

In addition to the two protocols, we also implemented an

ETX measurement module, same as the one we used in our simulations (described in Section IV-A). The source code for the two protocols and the ETX module together is over 9000 lines of C code.

C. Experimental setup

In the implementation of the two protocols we used the same parameters as in our simulation study in Section IV. In all the experiments, the bitrate of the wireless cards was set to 2Mbps and the transmission power to 16dBm. We disabled RTS/CTS for unicast frames as most operational networks do. With these settings, the length of the shortest ETX paths between different nodes is 1-6 hops in length, and the loss rates of the links vary from 0% to 88%, with an average value of 29%.

We ran each protocol on 10 different scenarios (i.e., selection of source and multicast group members). In each scenario, 1 source and 4 receivers were randomly selected among the 22 nodes of our testbed. In each scenario, we first ran the ETX module for 10 minutes to collect the pairwise loss rates and ETX metric for each link of our testbed, and then we ran the two protocols, MORE and *Pacifier*, in sequence. With both protocols, the source sent a 2.3MB file consisting of 1460-byte packets. Since the quality of some links of our testbed varies substantially from day to day in a week, we repeated the experiments for the same 10 scenarios on 4 different days (one weekend and two weekdays). Due to space limitation, we present the results for the first and the fourth day (the results for the other two days were similar).

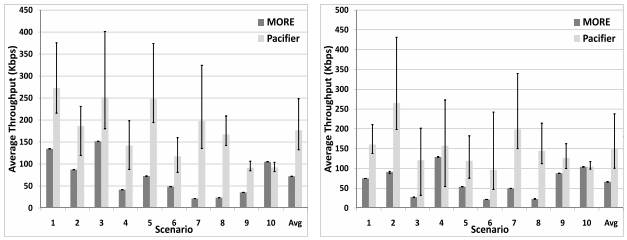
D. Experimental results

Figures 8(a), 8(b) show the average throughput achieved with MORE and *Pacifier* in each of the 10 scenarios, as well as the throughput of the best and the worst receiver (top and bottom of error bars) in each scenario, on 2 different days.

Similar to the simulation results, we observe that *Pacifier* outperforms MORE in 9 out of 10 scenarios on both days. The average throughput improvement over all 10 scenarios ranges between 83% (for Day 4) and 144% (for Day 1). This is somewhat lower than the corresponding simulation result (171% in Figure 7(a)). The reason is that the size of our testbed is much smaller than the simulated networks, and hence path diversity is not as large, and the "crying baby" problem is not as severe, as in the simulations.

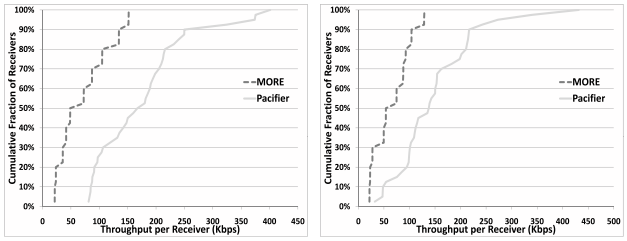
We observe again that *Pacifier* solves the "crying baby" problem, allowing well-connected receivers in each case to achieve throughputs much higher than the average value, while also improving throughput of the worst receivers in almost all scenarios. Averaging over 10 scenarios for each day, the throughput of the best receiver with *Pacifier* is 244% and 259% higher than with MORE, but, in some cases, it can be much higher, e.g., more than 8x in Scenario 8, Day4, and more than 14.4x in Scenario 7, Day1. Similarly, the average (over 10 scenarios) throughput of the worst receiver with *Pacifier* is on 83% and 53% higher than with MORE on each day, and the maximum improvement can be as high as 5.4x (higher than in the simulation results) in Scenario 7, Day 1.

Figures 9(a), 9(b) plot the CDF of the 40 throughput values obtained from the 10 scenarios with 4 receivers each, for the two protocols, on each of the 2 days. Similar to Figure 7(b) for



(a) Day 1 (b) Day 4

Fig. 8. Testbed throughput comparison of MORE and *Pacifier* in 10 different scenarios on 2 different days.



(a) Day 1 (b) Day 4

Fig. 9. CDFs of 40 testbed throughput measurements obtained with MORE and *Pacifier* for 10 scenarios with 4 receivers each on 2 different days.

the simulation results, we observe that the CDFs for MORE exhibit a staircase form. In contrast, with *Pacifier*, receivers finish independently of each other and the CDF always has a continuous form. *Pacifier* outperforms MORE on both days both in the median case, by 158 - 286%, and in the two ends of the CDFs – the 90th percentile is 85-128% higher and the 10th percentile is 128-294% higher with *Pacifier* than with MORE.

VI. Scope of Our Work

A. On the Choice of Transmission Rates

In both our simulation and our testbed evaluation, we used a low fixed MAC transmission rate of 2Mbps and disabled the rate adaptation mechanism of 802.11 for the following two technical reasons.

(i) Implementation related reasons: NC protocols are currently implemented as user-level programs and they cannot work with high data rates. The pioneering COPE [5] and MORE [6] protocols, implemented using the Click software router [52], were evaluated using 6Mbps and 5.5Mbps, respectively. In the publicly available implementation of MORE [50], it is stated that "it is highly recommended that no bit-rates faster than 6Mbps are used for comparison". Our implementation at the application layer posed even greater challenges with respect to timing constraints and we had to limit our testing to the bitrate of 2Mbps. Note that the second generation of NC/OR protocols exploiting PHY-MAC cross-layer interactions have been evaluated using even lower bitrates (a few hundreds of Kbps) due to hardware constraints (e.g., [53]).

(ii) Reasons related to the 802.11 standard: NC protocols are built on top of 802.11 broadcast; in contrast to unicast, 802.11 broadcast uses no rate adaptation. Integrating rate adaptation with broadcast is by itself a very interesting and at the same time extremely challenging open research problem, due to the speed vs. range tradeoff (a higher bitrate may reduce

the number of neighboring nodes being able to overhear a transmission, limiting the gain from OR). Preliminary efforts to integrate MORE with an offline rate selection algorithm have not produced satisfactory results [54]. The lack of an appropriate rate adaptation algorithm for broadcast has led many researchers to use a low fixed bitrate even in simulation studies of NC/OR protocols in order to exploit the Wireless Multicast Advantage (WMA) [8] (e.g., the very recent works [55], [56]).

B. On Link Loss Rate Measurement

In our evaluation of *Pacifier* and MORE, we followed the evaluation methodology of all state-of-the-art OR/NC protocols (e.g., [4], [6], [5]) and measured the link loss rates only once, prior to each experiment. In practice, a link state mechanism should be used to ensure that nodes periodically broadcast probes to their neighbors and the estimated loss rates are periodically but less frequently distributed to all the nodes. As [6] argues, such a link state mechanism is required in all state-of-the-art routing protocols, and the overhead this process incurs is not considered *Pacifier*-specific.

The use of stale loss measurements can affect two features of OR/NC protocols: topology control (i.e., the choice of FNs) and credit calculation. As pointed out in [4], the impact on the former is not significant. The performance of OR protocols is relatively insensitive to suboptimal choice of FNs, since a packet's actual path is determined by conditions at the time of transmission. This is the main benefit of OR.[6] However, the impact on the latter may be more serious, since inaccurate credit calculation may result in a large number of redundant transmissions leading to congestion. Addressing this problem, which can affect all OR/NC protocols, is not trivial, and is out of scope of this paper. In our recent work [57], we proposed a solution for unicast OR/NC protocols, which decouples credit calculation from loss rate measurements. We plan to extend this solution to multicast in our future work.

C. Generality of Our Results

The above limitations which prevent the use of higher data rates in the evaluation of NC-based OR protocols should not limit by any means the scope of our work and the generality of our results to the low rate 802.11b. On the contrary, the significance of our work will only be more pronounced when considering future high rate technologies (with fixed or varying rates). A core contribution of our work is to solve the classic "crying baby" problem in the new context of reliable multicast to a set of heterogeneous receivers in WMNs. As we move towards novel 802.11 technologies offering a larger selection of data rates (up to 54Mbps for 802.11a/g, up to 600Mbps for 802.11n), receiver heterogeneity will only increase, making the "crying baby" problem even more prominent and the need for our solution even greater.

Imagine an example scenario in an 802.11n network with two receivers: receiver $R1$ is directly connected to the multicast source over a 600Mbps link while receiver $R2$ is connected to the source over a 3-hop path including one poor link of 26Mbps. Even though the throughput of $R2$ is already much higher than the throughput achieved in an 802.11b network, there is no reason to limit $R1$'s throughput to $R2$'s level. Doing so would significantly underutilize the network and does not exploit the maximum benefits possible from the advanced PHY layer.

[6]In contrast, in single path routing, suboptimal choice of FNs may even result in a broken path.

VII. Conclusion

High-throughput, reliable multicast routing has many important applications in WMNs, such as software updates and video/audio file downloads. However, designing high-throughput, reliable multicast protocols faces two challenges: the inherent lossiness of wireless links and the "crying baby" problem. In this paper, we presented *Pacifier*, the first practical NC-based high-throughput, reliable multicast protocol for WMNs. *Pacifier* seamlessly integrates tree-based OR, intra-flow NC, source rate limiting, and round-robin batching, to achieve high throughput and solve the "crying baby" problem.

Our performance evaluation of *Pacifier* via extensive simulations and an implementation on a WMN testbed have shown *Pacifier* significantly outperforms the state-of-the-art MORE for various multicast scenarios. In particular, the experimental results on our 22-node WMN testbed show that *Pacifier* increases the average multicast throughput over MORE by 83-114%, while the maximum throughput gain for well-connected receivers is as high as 14x, and the maximum throughput gain for the "crying baby" itself is as high as 5.4x, compared to MORE.

Since the cumulative path loss rate in wireless multihop networks increases with the path length, multicast receiver heterogeneity is unavoidable in WMNs. In fact, the degree of heterogeneity is expected to increase as future WMNs scale in size. Our experience with designing *Pacifier* shows the importance of exploiting heterogeneity, rather than ignoring it. By treating heterogeneous receivers equally, MORE penalizes well-connected receivers, forcing them to achieve the same throughput as the worst receiver. In contrast, by exploiting heterogeneity, and prioritizing well-connected receivers over the "crying babies", *Pacifier* manages to achieve several-fold throughput improvement for well-connected receivers, without penalizing the poorly-connected ones; on the contrary, it often drastically improves throughput of the worst receivers.

References

[1] "MIT Roofnet," http://www.pdos.lcs.mit.edu/roofnet.

[2] "Seattle wireless," http://www.seattlewireless.net.

[3] "Technology For All (TFA)," http://tfa.rice.edu.

[4] S. Biswas and R. Morris, "ExOR: Opportunistic multi-hop routing for wireless networks," in *Proc of ACM SIGCOMM*, 2005.

[5] S. Katti, H. Rahul, W. Hu, D. Katabi, M. Medard, and J. Crowcroft, "XORs in the air: Practical wireless network coding," in *Proc. of ACM SIGCOMM*, August 2006.

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

[7] S. Katti, S. Gollakota, and D. Katabi, "Embracing wireless interference: Analog network coding," in *Proc. of ACM SIGCOMM*, 2007.

[8] M. Cagalj, J.-P. Hubaux, and C. Enz, "Minimum-energy broadcast in all-wireless networks: NP-Completeness and distribution," in *Proc. of ACM MobiCom*, September 2002.

[9] R. Chandra, S. Karanth, T. Moscibroda, V. Navda, J. Padhye, R. Ramjee, and L. Ravindranath, "Dircast: A practical and efficient wi-fi multicast system," in *Proc. of IEEE ICNP*, 2009.

[10] S. Sen, N. K. Madabhushi, and S. Banerjee, "Scalable wifi media delivery through adaptive broadcasts," in *Proc. of USENIX NSDI*, 2010.

[11] R. Chandra, V. Ramasubramaniam, and K. Birman, "Anonymous gossip: Improving multicast reliability in mobile ad hoc networks," in *Proc. of ICDCS*, 2001.

[12] J. Luo, P. Eugster, and J.-P. Hubaux, "Route driven gossip: Probabilistic reliable multicast in ad hoc networks," in *Proc. of IEEE Infocom*, 2003.

[13] B. Scheuermann, M. Transier, C. L. M. Mauve, and W. Effelsberg, "Backpressure multicast congestion control in mobile ad-hoc networks." in *Proc. of CoNEXT*, 2007.

[14] J. Park, M. Gerla, D. S. Lun, Y. Yi, and M. Medard, "Codecast: a network-coding-based ad hoc multicast protocol," *IEEE Wireless Communications*, vol. 13, no. 5, 2006.

[15] K. Tang, K. Obraczka, S.-J. Lee, and M. Gerla, "A reliable, congestion-controlled multicast transport protocol in multimedia multi-hop networks," in *Proc. of WPMC*, 2004.

[16] E. Pagani and G. Rossi, "Reliable broadcast in mobile multihop packet networks," in *Proc. of MobiCom*, 1997.

[17] A. Sobeih, H. Baraka, and A. Fahmy, "ReMHoc: A reliable multicast protocol for wireless mobile multihop ad hoc networks," in *IEEE Consumer Communications and Networking Conference (CCNC)*, 2004.

[18] V. Rajendran, Y. Yi, K. Obraczka, S.-J. Lee, K. Tang, and M. Gerla, "Combining source- and localized recovery to achieve reliable multicadt in multi-hop ad hoc networks," in *Proc. of Networking*, 2004.

[19] D. Koutsonikolas and Y. C. Hu, "The case for FEC-based reliable multicast in wireless mesh networks," in *Proc. of DSN*, 2007.

[20] L. Rizzo and L. Visicano, "RMDP: an FEC-based reliable multicast protocol for wireless environments," *Mobile Computing and Communications Review*, vol. 2, no. 2, 1998.

[21] D. Lun, M. Medard, and R. Koetter, "Efficient operation of wireless packet networks using network coding," in *Proc. of IWCT*, 2005.

[22] M. Ghaderi, D. Towsley, and J. Kurose, "Reliability Gain of Network Coding in Lossy Wireless Networks ," in *Proc. of IEEE INFOCOM*, 2008.

[23] H. W. Holbrook, S. K. Singhal, and D. R. Cheriton, "Log-based receiver-reliable multicast for distributed interactive simulation," in *Proc. of ACM SIGCOMM*, 1995.

[24] D. Aguayo, J. Bicket, S. Biswas, G. Judd, and R. Morris, "Link-level measurements from an 802.11b mesh network," in *Proc. of ACM SIGCOMM*, August 2004.

[25] S. Floyd, V. Jacobson, C.-G. Liu, S. McCanne, and L. Zhang, "A reliable framework for light-weight sessions and application level framing," *IEEE/ACM Transactions on Networking*, 1997.

[26] L. Rizzo, "Effective erasure codes for reliable computer communication protocols," *ACM Comp. Comm. Review*, vol. 27, no. 2, 1997.

[27] J. Nonnenmacher, E. Biersack, and D. Towsley, "Parity-based loss recovery for reliable multicast transmission," in *ACM SIGCOMM*, 1997.

[28] E. Schooler and J. Gemmel, "Using multicast FEC to solve the midnight madness problem," Technical Report, MSR-TR-97-25, Tech. Rep., 1997.

[29] M. Luby, "LT codes," in *Proc. of 43rd FoCS*, 2002.

[30] P. Maymounkov and D. Mazieres, "Rateless codes and big downloads," in *Proc. of IPTPS*, 2003.

[31] A. Shokrollahi, "Raptor codes," in *Proc. of IEEE International Symposium on Information Theory (ISIT)*, 2004.

[32] A. W. Eckford and W. Yu, "Rateless slepian-wolf codes," in *Proc. of 39th Asilomar Conference on Signals, Systems and Computers*, 2005.

[33] E. Vollset and P. Ezhilchelvan, "A survey of reliable broadcast protocols for mobile ad-hoc networks," University of Newcastle upon Tyne, Tech. Rep. CS-TR-792, 2003.

[34] S. Gupta and P.Srimani, "An adaptive protocol for reliable multicast in mobile multi-hop radio networks," in *Proc. of IEEE Workshop on Mobile Computing Systems and Applications*, 1999.

[35] T. Gopalsamy, M. Singhal, and P. Sadayappan, "A reliable multicast algorithm for mobile ad hoc networks," in *Proc. of ICDCS*, 2002.

[36] K. Tang, K. Obraczka, S.-J. Lee, and M. Gerla, "Reliable adaptive lightweight multicast protocol," in *Proc. of ICC*, 2004.

[37] D. S. J. D. Couto, D. Aguayo, J. C. Bicket, and R. Morris, "A high-throughput path metric for multi-hop wireless routing," in *Proc. of ACM MobiCom*, 2003.

[38] E. Rozner, J. Seshadri, Y. Mehta, and L. Qiu, "Simple opportunistic routing protocol for wireless mesh networks," in *Proc. of WiMesh*, 2006.

[39] S.-J. Lee, M. Gerla, and C.-C. Chiang, "On-Demand Multicast Routing Protocol," in *Proc. of IEEE WCNC*, September 1999.

[40] C. Gkantsidis, W. Hu, P. Key, B. Radunovic, S. Gheorghiu, and P. Rodriguez, "Multipath code casting for wireless mesh networks," in *Proc. of ACM CoNEXT*, 2007.

[41] C. Reis, R. Mahajan, M. Rodrig, D. Wetherall, and J. Zahorjan, "Measurement-based models of delivery and interference in static wireless networks," in *Proc. of ACM SIGCOMM*, 2006.

[42] Y. Li, L. Qiu, Y. Zhang, R. Mahajan, Z. Zhong, G. Deshpande, and E. Rozner, "Effects of interference on throughput of wireless mesh networks: Pathologies and a preliminary solution," in *Proc. of HotNets-VI*, 2007.

[43] D. Koutsonikolas, Y. C. Hu, and K. Papagiannaki, "How to evaluate exotic wireless routing protocols?" in *Proc. of ACM HotNets-VII*, 2008.

[44] B. Scheuermann, C. Lochert, and M. Mauve, "Implicit hop-by-hop congestion control in wireless multihop networks," *Elsevier Ad Hoc Networks*, vol. 6, no. 2, pp. 260–286, Apr. 2008.

[45] Y. Lin, B. Li, and B. Liang, "CodeOR: Opportunistic routing in wireless mesh networks with segmented network coding," in *Proc. of IEEE ICNP*, 2008.

[46] J. Byers, M. Luby, and M. Mitzenmacher, "A digital fountain approach to asynchronous reliable multicast," *Journal on selected areas in communications*, vol. 20, 2002.

[47] X. Zeng, R. Bagrodia, and M. Gerla, "Glomosim: A library for parallel simulation of large-scale wireless networks," in *Proc. of PADS Workshop*, May 1998.

[48] "http://www.engineering.purdue.edu/mesh."

[49] madwifi, "http://madwifi.org."

[50] "MORE source code," http://people.csail.mit.edu/szym/more.

[51] Linux Advanced Routing and Traffic Control, "http://lartc.org."

[52] R. Morris, E. Kohler, J. Jannoti, and M. F. Kaashoek, "The click modular router," in *Proc. of. SOSP*, 1999.

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

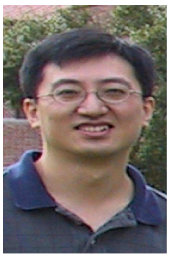
[54] M. Afanasyev and A. C. Snoeren, "The importance of being overheard: Throughput gains in wireless mesh neworks," in *Proc. of ACM SIGCOMM/USENIX IMC*, 2009.

[55] E. Rozner, M. K. Han, L. Qiu, and Y. Zhang, "Model-driven optimization of opportunistic routing," in *Proc. of ACM SIGMETRICS*, 2011.

[56] M. K. Han, A. Bhartia, L. Qiu, and E. Rozner, "Optimized overlay-based opportunistic routing," in *Proc. of ACM MobiHoc*, 2011.

[57] D. Koutsonikolas, C.-C. Wang, and Y. C. Hu, "Efficient network coding based opportunistic routing through cumulative coded acknowledgments," *IEEE/ACM Transactions on Networking*, vol. 19, no. 5, October 2011.

Dimitrios Koutsonikolas received the Ph.D. degree in Electrical and Computer Engineering from Purdue University, West Lafayette, IN, in 2010. He worked as a Post-Doctoral Research Associate at Purdue University from September to December 2010. He is currently an assistant professor of Computer Science and Engineering at the University at Buffalo, the State University of New York. His research interests are broadly on experimental wireless networking and mobile computing. He is a member of IEEE, ACM, and USENIX.



Y. Charlie Hu is a Professor of Electrical and Computer Engineering at Purdue University. He received his Ph.D. degree in Computer Science from Harvard University in 1997. From 1997 to 2001, he was a research scientist at Rice University. His research interests include operating systems, distributed systems, Internet measurement and routing analysis, and wireless networking. He has published over 130 papers in these areas. Dr. Hu received the NSF CAREER Award in 2003. He is a senior member of IEEE and an ACM distinguished scientist.

Chih-Chun Wang joined Purdue School of Electrical and Computer Engineering in 2006 as an Assistant Professor. He received the B.E. degree from National Taiwan University in 1999, and the M.S. and Ph.D. degrees in E.E. from Princeton University in 2002 and 2005, respectively. His current research interests are in the graph-theoretic and algorithmic analysis of iterative decoding and of network coding. His other research interests fall in the general areas of optimal control, information theory, detection theory, coding theory, iterative decoding algorithms, and network coding. He received the NSF CAREER Award in 2009.