Graduate School ETD Form 9
(Revised 12/07)

# PURDUE UNIVERSITY
## GRADUATE SCHOOL
## Thesis/Dissertation Acceptance

This is to certify that the thesis/dissertation prepared

By Dimitrios Koutsonikolas

Entitled `Exotic' Routing Protocol Design and Implementation for Wireless Mesh Networks

For the degree of Doctor of Philosophy

Is approved by the final examining committee:

| | |
|---|---|
| Y. C. Hu<br>Chair | |
| C. C. Wang | |
| S. G. Rao | |
| S. A. Fahmy | |

To the best of my knowledge and as understood by the student in the *Research Integrity and Copyright Disclaimer (Graduate School Form 20)*, this thesis/dissertation adheres to the provisions of Purdue University's "Policy on Integrity in Research" and the use of copyrighted material.

Approved by Major Professor(s): Y. C. Hu

Approved by: V. Balakrishnan 7/19/10
Head of the Graduate Program Date

Graduate School Form 20
(Revised 10/07)

# PURDUE UNIVERSITY
## GRADUATE SCHOOL

## Research Integrity and Copyright Disclaimer

Title of Thesis/Dissertation: `Exotic' Routing Protocol Design and Implementation for Wireless Mesh Networks

For the degree of Doctor of Philosophy

I certify that in the preparation of this thesis, I have observed the provisions of *Purdue University Executive Memorandum No. C-22,* September 6, 1991, *Policy on Integrity in Research.**

Further, I certify that this work is free of plagiarism and all materials appearing in this thesis/dissertation have been properly quoted and attributed.

I certify that all copyrighted material incorporated into this thesis/dissertation is in compliance with the United States' copyright law and that I have received written permission from the copyright owners for my use of their work, which is beyond the scope of the law. I agree to indemnify and save harmless Purdue University from any and all claims that may be asserted or that may arise from any copyright violation.

Dimitrios Koutsonikolas
Signature of Candidate

7/19/10
Date

*Located at http://www.purdue.edu/policies/pages/teach_res_outreach/c_22.html

# "EXOTIC" ROUTING PROTOCOL DESIGN AND IMPLEMENTATION FOR WIRELESS MESH NETWORKS

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Dimitrios Koutsonikolas

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

August 2010

Purdue University

West Lafayette, Indiana

Dedicated to my parents Stefanos Koutsonikolas and Eleni Koutsonikola, and my sister Natasa Koutsonikola, for their support, encouragement, and love. To the memory of my grandparents Dimitrios Koutsonikolas and Maria Koutsonikola.

## ACKNOWLEDGMENTS

I am forever indebted to my advisor Y. Charlie Hu for his constant guidance and mentorship in my research and for instilling the drive to pursue new ideas in me. By allowing me to be independent while steering work on the right course, he has made me confident in my abilities to succeed in research and handle new challenges in the future. I am also grateful to the members of my Advisory Committee; Sonia Fahmy, Sanjay Rao, and Chih-Chun Wang, for providing guidance and taking time out to oversee my research. I would like to thank especially Chih-Chun Wang for the excellent collaboration over the past three years and for helping me to improve my theoretical skills.

I would like to express special thanks to Theodoros Salonidis and Henrik Lundgren for helping me learn new things and become a better researcher during my internship at the Thomson Paris Research Lab with them. Working with theory and systems people together gave me valuable training for future endeavors. I would also like to thank everyone at the Thomson Paris Research Lab for their insights and valuable discussions on my work, and for making my internship a pleasurable experience.

I would like to express my sincere gratitude to all my collaborators – Saumitra Das, Himabindu Pucha, Sabyasachi Roy, Dimitrios Peroulis, Konstantina Papagiannaki, Pascal Le Guyadec, Irfan Sheriff, Yung-Hsiang Lu and others. Special thanks go to Saumitra Das who was almost like a second advisor to me at the beginning of my PhD, helping me to obtain all the necessary practical skills for doing successful research in wireless experimental networking. I especially enjoyed working closely with him those long days and nights deploying the MAP wireless testbed or running experiments on it.

I would also like to thank all the fellow researchers and friends in the Distributed Systems and Networking Lab and outside – Chris Gniady, Rongmei Zhang, Zheng

Zhang, Sabyasachi Roy, Ali Jafri, Ali Butt, Abhinav Pathak, Bowen Zhou, Di Xie, Soumyadeep Banerjee, Hamza Bin Sohail, Ruben Torres, V.J Venkataramanan, Georgios Karakonstantis, Georgios Panagopoulos, Miltos Alamaniotis, and others, for our fruitful discussions, for their help and support, and for making this long journey towards the PhD more pleasurable.

Finally, thanks are due to the people of the Electrical and Computer Engineering Graduate Office, especially Matt Golden, who helped and guided me through all the steps and requirements of the Purdue ECE PhD program.

# TABLE OF CONTENTS

Page

## LIST OF TABLES

LIST OF FIGURES

Figure Page

## ABSTRACT

Koutsonikolas, Dimitrios. Ph.D., Purdue University, August 2010. “Exotic” Routing Protocol Design and Implementation for Wireless Mesh Networks. Major Professor: Y.Charlie Hu.

Wireless mesh networks (WMNs) are increasingly being deployed for providing cheap Internet access. A main challenge in city-wide WMNs is to deal with the poor link quality due to urban structures and interference. To combat the high loss rates and satisfy the ever-growing demands for high-throughput Internet access, researchers have recently proposed a new class of “exotic” routing protocols exploiting techniques such as opportunistic routing or network coding, that promise a dramatic increase in overall network throughput. Nonetheless, the design of practical protocols exploiting such techniques still faces several challenges that often prevent them from realizing the maximum possible gains.

This thesis presents the design, prototyping, and evaluation of two novel “exotic” routing protocols that address fundamental limitations of state-of-the-art protocols for multicast and unicast routing, respectively. The first part of the thesis presents *Pacifier*, the first reliable multicast protocol for WMNs that efficiently addresses the well-known (from the Internet) “crying baby” problem, i.e., when the presence of a poorly connected multicast receiver results in performance degradation for the rest of the receivers. The second part of the thesis presents a novel solution to the main challenge in network coding based opportunistic routing protocols, i.e., determining how many coded packets each forwarding node should transmit, through the design of CCACK. Finally, in the third part of the thesis, we review the current diverse practices in the evaluation methodology of this new class of “exotic” routing protocols, examine their strengths and their weaknesses, and make recommendations for more consistent and meaningful evaluation methodologies.

# 1. INTRODUCTION

Wireless mesh networks [1] (WMNs) have recently gained significant popularity as an alternative technology for providing cheap, low maintenance Internet access. This type of networks, also known as community networks, are characterized by static mesh routers connected by wireless links to each other and to a few gateway nodes. These few gateway nodes have wired connectivity to the Internet. The WMN routers and the gateway nodes effectively form a multihop wireless access backbone. A typical mesh network is depicted in Figure 1.1 in which mesh routers are deployed inside or on rooftops of homes.

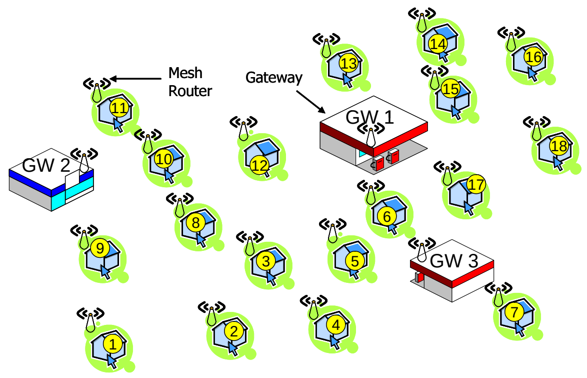


Fig. 1.1. Schematic of a wireless typical mesh network.

These mesh routers connect to each other over wireless links so that each home can reach one of the three Internet gateways either directly or via multi-hop packet forwarding among the mesh routers. Mesh routers are similar to cheap commodity access point devices that are widely available. Typically home users connect to the mesh router on a separate radio interface or via wired Ethernet.

WMNs typically have low deployment cost and low maintenance overhead, high data rates, and are not energy constrained. Because of these properties, the deployment and use of WMNs has recently increased significantly and several cities have planned and/or deployed WMNs [2–11]. In addition to broadband Internet access, WMNs are also an attractive option for enterprise wireless offices [12] in which the entire office communication (printing, servers, voice communication, etc.) takes place over wireless links.

However, despite their attractive properties, WMNs face a fundamental challenge to be successful: **performance**. Unlike traditional mobile ad hoc networks (MANETs), the routers in WMNs are static; thus, topology changes due to mobility or power consumption are not a concern in these networks. In addition, a WMN aims to be a last-mile technology. Thus, in order to become a viable solution, it has to compete with existing broadband technologies, such as cable and DSL, and offer performance comparable to the performance offered by them. As a consequence, the main design goal for WMNs has been shifted from maintaining connectivity among routers and power saving to improving applications' performance, in particular providing (primarily) high-throughput and reliability in network access.

A fundamental factor that can result in poor performance in WMNs is the poor link quality due to urban structures and interference, both internal (among flows in the WMN competing for the wireless medium via the 802.11 CSMA protocol) and external (from other 802.11 networks or other sources of interference, e.g., microwaves or cordless phones). For example, 50% of the operational links in Roofnet [2] have loss rates higher than 30% [13]. Multihop routing makes the problem even more challenging since the cumulative end-to-end loss rates are much higher. In addition, a third type of interference, self-interference (among nodes that are part of the same routing path carrying traffic for a given flow), further exacerbates the situation. As a consequence, routing protocol design in WMNs has to overcome these challenges to meet the goal of high performance.

## 1.1 Evolution of wireless routing protocol design

In this section we give a brief overview of the evolution of unicast and multicast routing protocol design for multihop wireless network, which is summarized in Figure 1.2. This overview reveals that despite significant innovations in routing protocol design, there is still much to be desired in terms of performance. The discussion in this section motivates our contributions in Section 1.3.

| | | **WMN Routing Protocols** | |
|---|---|---|---|
| | | **Unicast** | **Multicast** |
| **Traditional Routing** | **Ad Hoc Era** | DSDV ('94), DSR ('96), AODV ('97) | ODMRP ('99), MAODV ('99) |
| | **Mesh Era** | ETX ('03), ETT ('04), ... | METX, SPP ('05) |
| **"Exotic" Routing** | | ExOR ('05) | Pacifier ('09) |
| | | MORE ('07) | MORE ('07) |
| | | CCACK ('10) | |

Fig. 1.2. Evolution of unicast and multicast routing protocol design for multihop wireless networks. The white area denotes existing work and the shaded area denotes our contributions.

In the ad hoc networking era, the primary challenge faced by routing protocols, both unicast (e.g., DSR [14], AODV [15]) and multicast (e.g., ODMRP [16], ADMR [17]) was to deal with frequent route breaks due to host mobility in a dynamic mobile environment. Accordingly, most research efforts were focused on designing efficient route discovery/repair schemes to discover or repair routes with minimum overhead. The routing process itself was simple; once a route from the source to a destination was known, each hop along the route simply transmitted the packet to the next hop via 802.11 unicast. These protocols relied on 802.11 unicast (with its built-in ACK-based local recovery scheme and exponential backoff) to deal with packet loss due to channel errors or collisions.

Transition to WMNs changed the design goals of routing protocols. As we have already mentioned, in a WMN, routers are static and hence route changes due to mobility are not a concern anymore. In addition, WMNs are proposed for urban usage (and not for military or rescue applications). The main performance metric is now *throughput*, often times even at the cost of increased control overhead.

The first major effort towards the new design goal was on designing link-quality path metrics that replaced the commonly used shortest-path metric. These metrics (e.g., ETX [18], ETT [19]) characterize link quality using information from the MAC layer and are then used to assist route selection in finding a high-throughput end-to-end path. The use of these new metrics was shown to improve the throughput of routing protocols in mesh testbeds (e.g. [18, 19]). The protocols using these link-quality metrics still followed the layering principle: the routing layer finds a good route, and 802.11 MAC is used to deliver packets hop by hop.

Seeking further throughput improvement, researchers looked into new, "exotic" techniques, which took two important departures from traditional routing. First, they abandoned the notion of the wireless link, by exploiting *wireless broadcast* at the MAC layer. Second, they largely abandoned the layering principle, adopting a *cross-layer* approach.

**Opportunistic routing.** The first such technique was opportunistic routing as demonstrated in the ExOR protocol [20].

Traditional routing protocols (independent of the routing metric they used) were built on top of 802.11 unicast. The rule for routing was simple: first determine the next hop, then let the MAC ensure that the packet is indeed delivered to the pre-determined next hop. As shown, this rule was a limiting factor for achieving high throughput.

Instead of having a decoupled MAC and routing layer, opportunistic routing exploits an inherent property of the wireless medium, its broadcast nature. Instead of first determining the next hop and then sending the packet to it, it broadcasts the packet so that all neighbors have the chance to hear it and assist in forwarding; among those that received the packet, the node closest to the destination forwards the packet. In practice, it is not beneficial if all nodes in the network participate in forwarding traffic for a single flow. Hence, existing opportunistic routing protocols typically construct a *belt* of forwarding nodes (FNs) for each flow and only members of the belt are allowed to forward packets.

Opportunistic routing provides significant throughput gains compared to traditional routing; however, it introduces a difficult challenge. Without any coordination, all members of the FN belt that hear a packet will attempt to forward it, creating duplicate transmissions, which waste bandwidth. To address this challenge, a coordination protocol needs to run among the nodes, so that they can determine which one should forward each packet. Coordination requires message exchange and it can be costly in terms of wireless bandwidth. To avoid control message exchange, ExOR imposes a strict global scheduler that works in rounds and allows only a single FN at a time to transmit, thus significantly limiting spatial reuse.

**Intra-flow network coding.** The next breakthrough that addressed this challenge in a simple and efficient manner, with minimal coordination was applying network coding [21] to multihop wireless networks, as was first demonstrated in the MORE protocol [22]. With network coding, the source sends random linear combinations of packets, and each router also randomly mixes packets it already has received before forwarding them. The destination can reconstruct a file consisting of $N$ packets if it receives $N$ linearly independent coded packets. Random mixing at each router ensures that with high probability different nodes that may have heard the same packet can still transmit linearly independent coded packets.

Network coding has one more positive effect. It resembles traditional Forward Error Correction (FEC) techniques, which offer reliability through redundancy, with the extra advantage that it is applied at every hop, and not end-to-end [23,24]. Hence, network coding completely eliminates the need for retransmissions of lost packets. Since each coded packet contains information about many packets, nodes can simply keep sending encoded packets until the destination receives sufficiently many packets that allow reconstruction of the original data.

## 1.2 New challenges

Initial practical routing protocols exploiting these "exotic" techniques were implemented and evaluated on small WMN testbeds; their evaluation promised several-fold throughput gains over traditional routing protocols. Nonetheless, a careful consideration reveals several limitations in the design of these protocols. For example, the lack of a rate control mechanism in these initial protocols causes a serious performance degradation as the number of flows in the network increases. As another example, the use of network coding introduces a new challenge: *How many coded packets should each forwarder transmit?* This challenge, if not efficiently addressed, can have a severe impact to the performance of network coding based opportunistic routing protocols. Consequently, such protocols often work particularly well for a specific scenario, but their performance may be far from satisfactory in different scenarios.

A second challenge lies in the evaluation methodology of this new class of protocols. In contrast to traditional routing protocols, there have been no clear guidelines that drive the evaluation of "exotic" routing protocols; often times each new protocol is evaluated with a different methodology. Even worse, several pitfalls in the evaluation methodology of this class of protocols often result in performance gains over traditional routing appearing higher than in reality.

Overall, the first generation of "exotic" routing protocols were a *proof of concept*, showing that "exotic" techniques such as opportunistic routing or network coding can indeed yield substantial performance improvements over traditional routing. Having understood the limitations of this first class of "exotic" protocols, the next step is to turn proof of concept to practical, robust, complete protocols.

Furthermore, the design of high-throughput routing protocols for WMNs has almost solely focused on unicast routing, while multicast has been largely ignored. Indeed, high-throughput routing metrics (ETX [18], ETT [19]) were proposed for unicast routing and they were integrated into unicast routing protocols, such as DSR [14] and DSDV [25]. Also, almost all the "exotic" routing protocols were only proposed for unicast routing. The only exception is MORE, which works both for unicast and multicast routing, although its design and evaluation were focused on unicast.

## 1.3 Research Contributions

In this thesis, we propose, design, and implement two *practical, robust* “exotic” routing protocols for WMNs that advance the state-of-the-art by addressing fundamental limitations of existing protocols.

- First, this thesis fills the gap in high-throughput multicast routing. We propose *Pacifier*, the first high throughput reliable multicast routing protocol for file download applications, that require 100% Packet Delivery Ratio (PDR). One fundamental challenge to supporting high-throughput, reliable multicast is the “crying baby” problem as first pointed out in [26] 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. *Pacifier* is the first protocol to efficiently address this challenge in WMNs in the context of file download applications, where the strict reliability requirement makes this problem particularly challenging.

- Second, we propose a novel solution to the main challenge in NC-based OR protocols, i.e., determining how many coded packets each forwarding node should transmit, through the design of the CCACK protocol. Existing protocols address this challenge by combining network coding with offline loss rate based heuristics to eliminate the overhead of feedback exchange, often at the cost of reduced performance in dynamic wireless environments. On the contrary, CCACK introduces an online **C**umulative **C**oded **ACK**nowledgment scheme that allows the protocol to sustain high performance in dynamic wireless environments, with practically zero overhead.

- The design of *Pacifier* and CCACK is guided by strong theoretical underpinning but at the same time emphasizes practical concerns, making the two protocols amenable to practical implementation on off-the-shelf hardware. An additional contribution of this thesis is the extensive evaluation of the proposed protocols under *realistic* settings and practical scenarios. First, both protocols were extensively evaluated using the Glomosim simulator, a widely-used wireless network simulator with a realistic physical model. In contrast to a recent trend in the wireless networking community that seems to reject simulation-based evaluation in favor of testbed evaluations, simulation remains valuable, as in many cases, it offers the chance of varying many parameters which are difficult to change in a testbed, and it complements the testbed evaluation. Real world systems, however, are complex, and even realistic simulations cannot capture all the details of a real world deployment. Hence, this thesis has a strong implementation component; both protocols have been prototyped and evaluated on MAP (Mesh@Purdue), a 22-node experimental 802.11a/b/g WMN testbed deployed on two academic buildings at Purdue University. In addition, to further facilitate research on "exotic" routing protocol design, we have made the source code for *Pacifier* publicly available. 17 research groups from 5 different countries have already downloaded it since September 2009.

- An additional contribution of this thesis lies in study of the evaluation methodology of this new class of "exotic" routing protocols. One implication of the advent of "exotic" techniques, such as OR and NC, is that many mechanisms such as reliability and rate control, that used to be below or above the routing layer in traditional routing protocols, have now moved to the routing layer. This consolidation of mechanisms from multiple layers into the routing layer poses new challenges to the methodology for evaluating and comparing this new generation of routing protocols. This thesis is the first to examine the diverse set of current practices in evaluating recently proposed protocols and identify their strengths and weaknesses. Our study suggests that there is an urgent need to carefully rethink the implications of the new merged-layer routing protocol design and develop effective methodologies for meaningful and fair comparison of these protocols. Finally, we make several concrete suggestions on the desired evaluation methodology.

**Bibliographic Note:** The material presented in this thesis is based on the following publications: [27], [28], and [29].

## 1.4 Organization of the Dissertation

This thesis is organized in 6 chapters. Chapter 2 gives a background on high throughput routing in WMNs, by briefly describing the three landmarks in the evolution of routing protocol design: link quality based routing metrics, opportunistic routing, and intra-flow network coding. Chapters 3 and 4 present the design, implementation and performance evaluation of *Pacifier* and CCACK, respectively. Chapter 5 presents an overview of the current practices in the evaluation methodology of "exotic" routing protocols, discusses their strengths and weaknesses and makes suggestions for a more meaningful evaluation. Finally, Chapter 6 concludes this thesis and discusses future work.

# 2. BACKGROUND

In this chapter, we give a brief background on high-throughput routing protocols in WMNs. We first describe traditional routing using link-quality based metrics in Section 2.1. We then turn to "exotic" techniques and describe opportunistic routing in Section 2.2 and intra-flow network coding in Section 2.3, using ExOR [20] and MORE [22], respectively, as examples.

## 2.1 Link quality based routing metrics

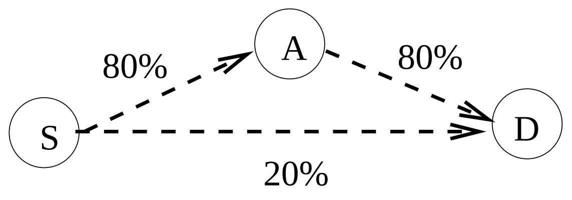
The first major effort towards high-throughput routing was on designing link-quality based routing metrics that replaced the commonly used hopcount metric. The main intuition behind link quality based routing metrics is the following: minimizing the hopcount maximizes the distance traveled by each hop, which is likely to minimize signal strength and maximize the loss rate. In contrast, it may be more advantageous to select a longer path, consisting of short links, each of high quality (e.g., low loss rate).

**ETX.** The idea of selecting a high-throughput path rather than the shortest path in a multihop wireless network was first introduced through the Expected Transmission Count (ETX) metric [18]. ETX is based on the expected number of transmissions required to send a unicast packet over a link, including retransmissions. To calculate ETX, each node measures the probability that a packet successfully reaches the receiver, denoted as $d_f$, and the probability that an ACK is successfully received by the sender, denoted as $d_r$. The ETX value of the link is given by

$$ETX = \frac{1}{d_f \times d_r} \tag{2.1}$$

The routing algorithm then selects the path with the least sum of ETX values of all its constituent links.

Figure 2.1 shows an example of how ETX avoids long lossy links. In this example, there are two paths from the source node $S$ to the destination node $D$: the direct path $S \to D$ consisting of only one link with 20% delivery ratio and the two-hop path $S \to A \to D$, consisting of two links, each with 80% delivery ratio. The hopcount metric would select the direct path $S \to D$ resulting in each packet being transmitted on average 5 times before being delivered to $D$. Instead, ETX selects the two-hop high quality path which requires on average only 2.5 transmissions per packet.



| **Path** | **Exp. transmissions** |
|---|---|
| S−>D | 1/0.2 = 5 |
| S−>A−>D | 1/0.8 + 1/0.8 = 2.5 |

Fig. 2.1. An example of the ETX metric.

To measure $d_f$ and $d_r$, each node broadcasts a probe packet every second. Each such probe contains the number of probes the node received from each of its neighbors in the previous 10 seconds. Since the 802.11 MAC layer protocol does not retransmit broadcast packets, nodes use this information to estimate the forward and reverse delivery probabilities.

**ETT.** ETX assumes that all links in the network have the same data rate. In practice, with the autorate adaptation mechanism of 802.11 enabled, links may have different data rates. In that case, the loss rate alone is not enough to describe the link quality. For example, it may be more advantageous to transmit a packet over an 11Mbps link with 30% loss rate rather than over a 1Mbps link with 0% loss rate. The Expected Transmission Time (ETT) metric [19] was proposed as an enhancement to ETX by considering both the loss rate and the bandwidth (data rate) of links. The ETT value of a link is computed as

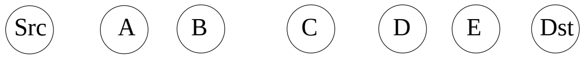
$$ETT = ETX \times \frac{S}{B} \tag{2.2}$$

where $S$ is the packet size and $B$ the link bandwidth. The routing algorithm again selects the path with the least values of all its constituent links. To calculate ETT for each link, a node measures ETX using the technique mentioned above, and the bandwidth using a technique similar to Packet Pair [30] (i.e., sending two probe packets back-to-back [31]).

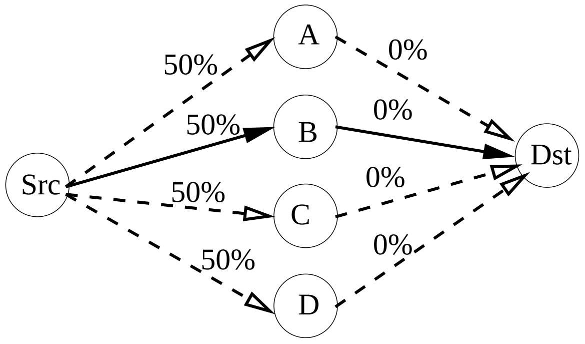
## 2.2 Opportunistic Routing

Opportunistic routing [20] belongs to a general class of wireless algorithms that exploit multi-user diversity. In contrast to traditional routing that uses a fixed path from a source to a destination, opportunistic routing provides more than one paths between the same source-destination pair. Traditional routing chooses the next hop before transmitting a packet. On the other hand, opportunistic routing sends the packet first and the decides the next hop among all neighbors that hear the packet successfully.

In a lossy environment, opportunistic routing can provide significant throughput improvements as a pre-selected next hop may not hear the transmitted packet but there may be many other neighbors that hear the packet and can serve as the next hop instead. With opportunistic routing, packets are *broadcast* at the MAC layer, so that all the nodes within the transmission range of a transmitter have a chance to receive the packet. Nodes then run a protocol to discover and agree on which nodes have received the packet. Among the nodes that have received the packet, only the node closest to the destination forwards (rebroadcasts) it. By delaying the decision of the next hop until after completion of the packet transmission, opportunistic routing



(a) Example 1: *Src*'s transmissions make different amounts of progress towards the destination.



(b) Example 2: Each of *Src*'s transmissions has many independent chances of being received by a node closer to the destination.

Fig. 2.2. Two examples explaining the throughput gain of opportunistic routing over traditional routing.

can try multiple long, lossy links in parallel, which results in high expected progress per transmission.

Figures 2.2(a), 2.2(b) show two typical examples in which opportunistic routing increases throughput compared to traditional routing.

In Figure 2.2(a), a chain of nodes A, B, C, D, and E lie between the source $Src$ and the destination $Dst$, serving as potential forwarders. The delivery probability from $Src$ to each of these nodes decreases with distance. A traditional routing protocol would select a subset of these nodes in a given order to from a path from $Src$ to $Dst$, for example $Src - B - D - Dst$. If a transmission from $Src$ is not received by B (because of fading, interference from other nodes, etc.), but it reaches A, that transmission is always wasted in traditional routing; A drops the packet and Src has to resend it. Similarly, if by chance a transmission from $Src$ reaches further from B, for example it is received directly by D, then again this transmission is wasted with traditional routing; D drops the packet and B has to forward it to D again. In contrast, opportunistic routing can exploit both these situations. In the first case A will transmit the packet, thus making some progress, while in the second case D will forward the packet instead of B, thus eliminating one transmission.

In Figure 2.2(b), assume that the loss rate for each of the links $Src-A$, $Src-B$, $Src-C$, $Src-D$ is 50% and the loss rate for each of the links $A-Dst$, $B-Dst$, $C-Dst$, $D-Dst$ is 0%. Traditional routing would send all the data packets through the same intermediate node, e.g., through the path $Src-B-Dst$. In this case, the loss rate of the path is 50%. In contrast, with opportunistic routing any of the four intermediate nodes can forward the packet, thus the loss rate drops to only $0.5^4 = 6.25\%$.

### 2.2.1 ExOR

ExOR [20] was the first opportunistic routing protocol for unicast transfers. We briefly review its major features: forwarding node (FN) selection, packet forwarding, and the global node FN scheduler.

**FN selection.** ExOR uses the ETX metric [18], based on loss rate measurements, to select the possible FNs. The source collects all the link loss rates periodically, calculates the ETX metric of each link, and includes in the FN list the nodes whose ETX distance to the destination is shorter than the source's distance. The FNs in the FN list are sorted in priority order using the ETX of each FN to the destination.

The algorithm for FN selection is run offline at the source. The algorithm starts by assuming that *every* node with an ETX to the destination lower than the source's ETX to the destination is a candidate FN 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; the remaining ones form a belt of FNs that connect the source to the destination.

**Packet forwarding and global scheduler.** To reduce the coordination overhead associated with any OR protocol, ExOR operates on batches of packets and uses a *global scheduler* trying to schedule the times at which FNs transmit their packets so that only one node sends at a time. Transmission of a batch starts with the source node transmitting all the packets of the current batch. Receiving nodes buffer successfully received packets and await the end of the batch. The highest priority FN then broadcasts the packets in its buffer. Each packet includes a copy of the sender's batch map, containing the sender's best guess of the highest priority node to have received each packet in the batch. The remaining FNs then transmit in order, sending only packets which were not acknowledged in the batch maps of higher priority nodes.

The source and the FNs continue to cycle through the priority list transmitting unacknowledged packets of the current batch until the destination has 90% of the packets of a batch. The last few packets in a batch would be the most expensive to send, since they would require all the overhead of running the transmission schedule, but the overhead would be divided among relatively few packets. For this reason, ExOR sends the last 10% of a batch over traditional routing, using 802.11 unicast.

## 2.3 Network Coding

With opportunistic routing, each packets is broadcast and all nodes in the transmission neighborhood have the chance to hear it. The goal is to ensure that among the nodes that hear a packet successfully, only the one closest to the destination will forward it. Without any coordination, all nodes that hear a packet will attempt to forward it, creating spurious retransmissions, which waste bandwidth. In order to ensure that *only one* node will forward the packet, a coordination protocol needs to run among the nodes. Coordination requires message exchange which can be costly in terms of wireless bandwidth. ExOR tries to avoid message exchange by using a global scheduler; however, this approach has its own drawbacks. First, it is difficult to implement a perfect global scheduler in practice. Second, the global scheduler allows only one node among the FNs to transmit at a given time, thus eliminating spatial reuse.

As was first shown in [22], intra-flow network coding offers a simple and efficient solution to the coordination problem in opportunistic routing protocols with practically zero overhead. With intra-flow network coding, the source sends random linear combinations of packets, and each intermediate node also randomly mixes packets it already has received before forwarding them. The destination can reconstruct a file consisting of $k$ packets if it receives $k$ linearly independent encoded packets. Random mixing at each intermediate node ensures that with high probability different nodes that may have heard duplicated versions of the same packet can still transmit linearly independent coded packets. As a result, redundant transmission is avoided not by exchanging feedback but by random network coding.

In addition to removing the need for feedback for coordination, intra-flow network coding has one more benefit. It reduces the feedback for reliability. [23,32] showed that the operation of mixing packets at the routers resembles the operation of rateless FEC codes. Actually, network coding 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 an actual hop-by-hop FEC implementation. With end-to-end FEC, the amount of redundancy the source is required to inject in the network is determined by the total loss over the path from the source to the destination, in the case of unicast, and by the total loss over the path from the source to the lossiest destination in the case of multicast. In contrast, with network coding, the maximum amount of redundancy injected by any node in the network is only determined by the lossiest link in the network.

### 2.3.1 MORE

MAC-independent Opportunistic Routing and Encoding (MORE) [22] was the first practical protocol that demonstrated the idea of combining network coding with opportunistic routing, with the primary goal of removing the need for coordination required in opportunistic routing. However, the design of MORE also guarantees

100% Packet Delivery Ratio (PDR), i.e., MORE is a protocol for reliable file transfers. Note that MORE is the only proposed "exotic" routing protocol that works for both unicast and multicast.

Similar to most "exotic" routing protocols, MORE is implemented as a shim between the IP and the 802.11 MAC layer. We briefly review its two major features: forwarding node (FN) selection and packet batching/forwarding.

**FN selection.** Similar to ExOR, MORE uses the ETX metric [18], to select the possible FNs. The source collects all the link loss rates periodically, calculates the ETX metric of each link and includes in the FN list the nodes whose ETX distance to that destination is shorter than the source's distance. In addition, for each FN the MORE 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 offline at the source. The unicast FN selection algorithm is borrowed from ExOR. After pruning nodes that are expected to perform less than 10% of the total transmissions, the source assigns TX_credits to the remaining ones, which form a belt of FNs that connect the source to the destination. In case of multicast, the algorithm is repeated for each destination, assuming a different hypothetical unicast flow; 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[1] and broadcasts it. Broadcasting a packet decrements the *credit_counter* by one unit.

The destination 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 (using 802.11 unicast) to notify the source to move to the next batch. In case of multicast, 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.

## 2.4 Summary

ExOR and MORE were two pioneering works in the new generation routing protocol design for WMNs, demonstrating through practical implementations and experimentation that "exotic" techniques, such as opportunistic routing and network

[1]Linear combinations of encoded packets are also linear combinations of the original packets.

coding, respectively, can indeed improve throughput of WMNs. Nonetheless, the design of these two protocols suffers from several drawbacks.

ExOR tries to solve the FN coordination problem, which is present in every opportunistic routing protocol, by implementing a global scheduler. The global scheduler allows only one FN to transmit packets at a time, thus significantly limiting spatial reuse, and preventing the protocol from realizing the maximum possible gain from opportunistic routing. In addition, implementing a global scheduler in practice is quite challenging, especially as the network size or the number of the flows in the network increases.

MORE addresses the coordination problem using network coding but introduces a new challenge; each FN needs to know *how many* coded packets it has to transmit rather than *which* packets. MORE addresses this new challenge using an *offline* heuristic at the source, based on link loss rates that are periodically collected from the whole network. As we show in Section 4.1.2, this approach may suffer significant performance degradation in dynamic wireless environments with continuously changing levels of channel quality, interference, and background traffic. In addition, MORE assigns no TX_credit to the source and applies no rate control, and thus it can easily congest the network.

The performance evaluation of both protocols focused on the simplest case of a single flow present in the network. Only [22] conducted one experiment with multiple flows present in the network, only to show that MORE's performance gain over ExOR and traditional routing *decreases* with the number of flows. In addition, MORE was evaluated on a relatively small, 20-node testbed. Thus, it is unclear if the protocol scales well as the network size increases. There are additional pitfalls in the evaluation methodology of either protocol, which we describe in Section 5.2.

Finally, the design of both protocols was focused on unicast and has largely ignored multicast. In particular, ExOR was only designed for unicast routing. MORE works for both unicast and multicast; however, its design is optimized for unicast. The multicast extension is a simple heuristic, which often selects too many FNs resulting in congestion, as we show in Chapter 3. In addition, MORE's policy of waiting for *all* multicast receivers to complete a batch before moving to the next one causes the "crying baby" problem, limiting the throughput of all the receivers to the throughput of the slowest receiver.

# 3. *Pacifier*: HIGH-THROUGHPUT, RELIABLE MULTICAST WITHOUT "CRYING BABIES" IN WIRELESS MESH NETWORKS

## 3.1 Introduction

In contrast to unicast routing, high-throughput, reliable multicast routing has received relatively little attention. Reliable multicast routing has many important applications in WMNs, such as software updates and video/audio file downloads. For example, a WMN node controlled by the network operator may host WMN-related software (e.g., a router software update or a security patch) or community-related audio/video files (e.g., a local football match that was held the day before) to WMN clients and serve as the multicast root. Or a group of WMN clients may want to download software hosted on an Internet server (i.e., a new version of Windows); in that case, a proxy at the WMN gateway serves as the multicast root.

A common characteristic of all these 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., [33–36]) 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 [37], 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., [38, 39]) which suffer the feedback implosion problem, or combine ARQ with congestion control (e.g., [35, 40]). Our recent work [41] studied the applicability of FEC and hybrid ARQ-FEC techniques, borrowed from the wired Internet, to WMNs, and showed that RMDP [42], a hybrid ARQ-FEC protocol, can achieve both reliability and high throughput.

More recently, as we mentioned in Chapter 2, researchers have applied network coding, a technique originally developed for the wireline Internet, to overcome the above challenge. [23] showed that the operation of mixing packets resembles the operation of rateless FEC codes. Actually, network coding 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 [24], the authors went one step further and showed that the reliability gain (expressed as the expected number of transmissions) of network coding 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 network coding for reliable multicast is still at a preliminary stage. As mentioned in Section 2.3.1, MORE [22] is the *only practical* network coding-based protocol that supports high-throughput, reliable multicast. It combines network coding with opportunistic routing, 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 [26] 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 [35], 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.

This chapter presents *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 opportunistic routing, intra-flow network coding, 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 network coding 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 network coding and opportunistic routing, 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 network coding 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.

The rest of the chapter is organized as follows. We begin with an overview of related work in the field of reliable multicast in Section 3.2. We then present the design of *Pacifier* and describe its various building blocks in Section 3.3. We discuss our simulation evaluation methodology in Section 3.4.1 and present extensive simulation results in Section 3.4.2. Section 3.5 describes the implementation and evaluation of *Pacifier* and MORE on a wireless testbed. Finally, Section 3.6 concludes this chapter.

## 3.2 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., [26,43]), to FEC schemes (e.g., [44]), to hybrid ARQ-FEC schemes (e.g., [42,45,46]), to rateless codes [47–50], 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 [51] classifies them into deterministic and probabilistic ones, depending on whether data delivery is fully reliable or not. Deterministic protocols (e.g., [37, 38, 40, 52–54]) 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., [33, 34]) 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, [41] studied the applicability of FEC and hybrid ARQ-FEC techniques, borrowed from the wired Internet, to WMNs, and showed that RMDP [42], 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. Network coding in theory is equivalent to hop-by-hop FEC [23, 24], 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/network coding 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 network coding). Due to the constraints on the buffer size and on packet delay, network coding 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 network coding-based protocol for high-throughput, reliable multicast routing (though it is also for unicast). To our knowledge, the only other practical network coding-based multicast protocol is CodeCast [36], which exploits network coding for *improving* but not *guaranteeing* reliability in multimedia multicast applications in mobile ad hoc networks.

## 3.3 *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 network coding-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.

### 3.3.1 Tree-based opportunistic routing

We argue that the use of opportunistic routing 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 [55]. 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 [22] for many unicast flows; the situation for many hypothetical unicast flows from a source to many multicast receivers is not very different. Second, the benefit of overhearing of broadcast transmissions, which is exploited by opportunistic routing 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 [16]), for improving the PDR.

The above observation motivates a simple multicast-tree based opportunistic routing 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.[1] The multicast tree is reconstructed at the source every time some receiver completes a batch (Section 2.3) and notifies the source.

**Batching and coded forwarding.** As in MORE, the source and the FNs in *Pacifier* use *intra-flow* random linear network coding. The hop-by-hop nature of network coding requires the source to break a file into small batches of packets so that the packet header overhead, encoding/decoding time, and memory requirements at the FNs remain low. We selected a batch size of $k = 32$ packets, same as in [22, 56]. For each batch, the source sends random linear combinations of the packets belonging to that batch. The random coefficients for each linear combination are selected from a Galois Field of size $2^8$, again same as in [22]. Intermediate FNs store all the innovative packets of the batch and also send random linear combinations of them. Every transmitted encoded packet is augmented with its coding vector, i.e., the random coefficients used to generate that packet. When a receiver receives any $k$ linearly independent coded packets of a batch, it decodes the batch to obtain the $k$ original packets. It then 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

[1]As [22] argues, periodic link loss rate measurements and their distribution to all nodes in the network are required in all state-of-the-art routing protocols, and the overhead this process incurs is not considered *Pacifier*-specific.

the "crying baby" problem as the completion time of each batch is limited by that of the worst receiver.

**How many packets should 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 $j$.[2] We also assume that wireless receptions at different nodes are independent, an assumption that is supported by prior measurements [57].

Let $N$ be the number of FNs in the multicast tree rooted at $s$. Let $\epsilon_{ij}$ 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:

$$L_{jk} = min(\sum_{i\in A(j)} z_i(1-\epsilon_{ij}), 1) - \sum_{i\in A(j)} z_i(1-\epsilon_{ik}) \qquad (3.1)$$

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:

$$z_{jk} = \frac{L_{jk}}{1-\epsilon_{jk}} \qquad (3.2)$$

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:

[2]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.

$$z_j = max_{k \in C(j)} z_{jk} \tag{3.3}$$

$z_j$ 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.

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:

$$\text{TX\_credit}_j = \frac{z_j}{\sum_{i \in A(j)} z_i(1 - \epsilon_{ij})} \tag{3.4}$$

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 tightly coupled; TX_credits are calculated for the whole network and then some FNs are pruned based on this calculation. As we will show in our evaluation in Section 4.4, this decoupling in *Pacifier* significantly improves the efficiency of both procedures. A second difference between the two approaches is that TX_credit calculation in *Pacifier* is optimized for multicast, while MORE optimizes it for unicast and resorts to a simple (but inefficient, as we show in Section 4.4) heuristic in the case of multicast.

### 3.3.2 Source rate limiting

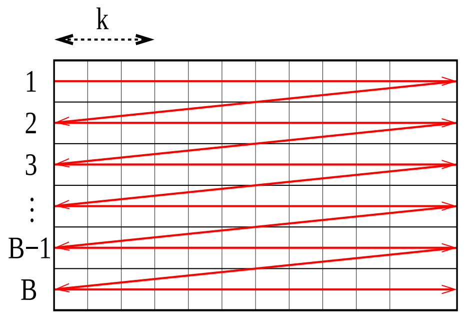
Recent studies have shown the importance of adding rate control to network coding-based unicast routing protocols, which exploit MAC layer broadcast [29, 55, 56, 58]. However, end-to-end rate control in multicast is much more complex than in unicast, and there is no widely accepted solution so far. In 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 [35]. 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.

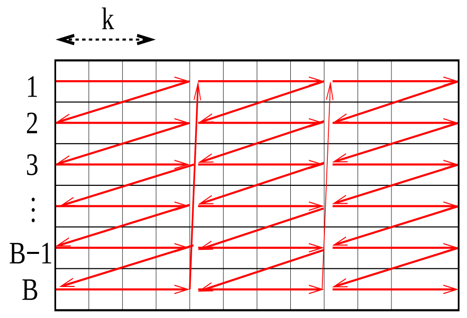
The work in [35] does not discuss how to set the timeout. In [59], the authors suggested a heuristic timeout of $3 \times T_p$ for the backpressure-based unicast version of BMCC, where $T_p$ is the transmission time of one data packet, which depends on the packet size and the MAC data rate. The factor 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.

### 3.3.3 Solving the “crying baby” problem

In MORE, the source keeps transmitting packets from the same batch until all the receivers acknowledge that batch, as shown in Figure 3.1(a). This policy makes



(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. 3.1. 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.

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. 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 3.1(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 (3.3) 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 above 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. This policy can impact even the throughput of a well-connected receiver, if, for example, congestion in the neighborhood around the source prevents the ACK to traverse the reverse path. Its impact, though, is more severe on the worst receivers. Assume all the well-connected receivers have acknowledged the batch in previous rounds. In the next round, for the remaining (worst) receivers, an ACK may take a long time to reach the source, since it has to traverse several hops, and, at each hop, it competes with data packets traversing down the tree. This could result in the source sending unnecessary packets from the current batch while waiting for an ACK to arrive. Instead, quickly moving to the next batch after finishing the scheduled packets, without waiting for any ACK, ensures that the "network pipe" is always filled with useful packets, and a delayed ACK will have little impact on the performance. Our evaluation in Sections 3.4.2, 3.5.4 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 [60] 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.

The above round-robin batching scheme is similar to the data carousel first introduced in Fcast [46], an FEC-based protocol. However, it is important to note that the use of network coding in *Pacifier* makes round-robin batching fundamentally more efficient than in the FEC-based protocol, by eliminating duplicate (i.e. non-innovative) packets. In Fcast, each batch of $k$ packets is encoded to produce $n > k$ packets (typical values for $k$, $n$, are 32 and 255, respectively), and a receiver can decode a batch if it receives any $k$ out of those $n$ packets. Once the source finishes transmission of all $n$ encoded packets from all $B$ batches, it starts a new round where it *retransmits again the same $n$ encoded packets* for each batch. This results in many receivers receiving duplicate packets and further delays decoding. For example, after all receivers have received the first $l < k$ packets of a batch in a round, they will receive those same $l$ packets in subsequent rounds. In contrast, in *Pacifier*, the source *sends different random combinations of the original $k$ packets in every round*, and in addition, every FN also mixes the packets it receives and sends out new combinations of them. This guarantees that, with very high probability, a receiver will never receive a duplicate packet.

**Adjusting TX_credit calculation.** In the basic version of *Pacifier* (Section 3.3.1), 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 (3.3). We note this new definition is also consistent with the policy of moving to the next batch whenever any receiver acknowledges the current batch.

**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 (3.3). The derivation of Equation (3.3) 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 (3.3), 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 (3.3). 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 3.4.2, we evaluate the impact of this knob by comparing the performance of *Pacifier* under different values of $knob$.

## 3.4 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 network coding and has the same memory requirements and the same fields in the packet header as MORE,[3] and hence it can be easily implemented in practice. We present an implementation study of *Pacifier* in Section 3.5.

### 3.4.1 Evaluation methodology

**Simulation setup.** We used the Glomosim simulator [61], 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 [22].

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

[3] *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.

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.

Following the methodology in [20, 22], 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:*[4] 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.

### 3.4.2 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 3.1 summarizes the different versions of MORE and *Pacifier* evaluated.

**Tuning MORE's pruning threshold.** Recall from Section 2.3.1 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.

[4]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.

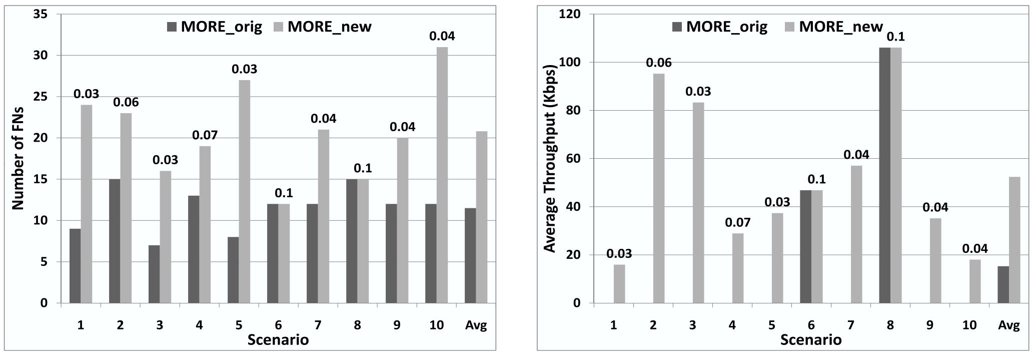
Table 3.1 Versions of MORE and *Pacifier* evaluated in our study. All versions include intra-flow network coding.

| **Name** | **Description** |
|---|---|
| MORE | MORE [22] 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 |

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 $\alpha$, 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 3.2 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 3.2(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 3.2(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.



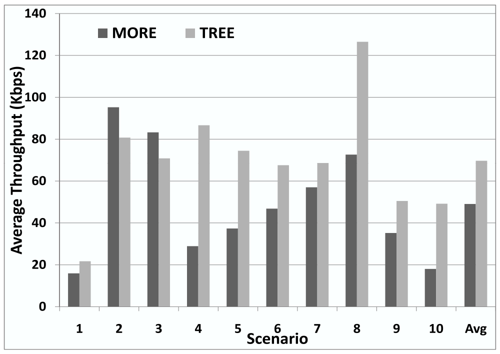
(a) Number of FNs (b) Throughput

Fig. 3.2. Number of FNs and throughput with the default threshold (MORE_orig) and the largest threshold not causing any disconnection (MORE_new), for 10 different scenarios. For MORE_new, the labels above the bars show the threshold used for each scenario.

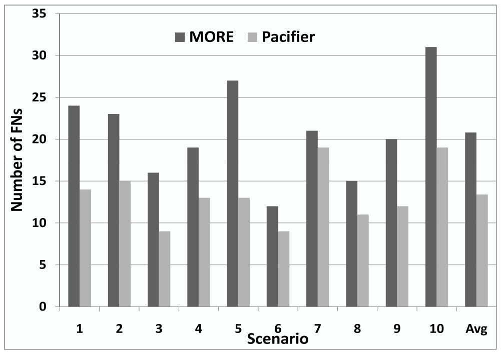
**Impact of tree-based opportunistic routing.** We start the evaluation of *Pacifier* by examining the impact of its tree-based opportunistic routing, by comparing the basic version of *Pacifier* (TREE), with MORE. The only difference between the protocols is the algorithm used for selecting FNs and assigning TX_credits to them. The results for 10 different scenarios are shown in Figure 3.3.

Figure 3.3(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%.

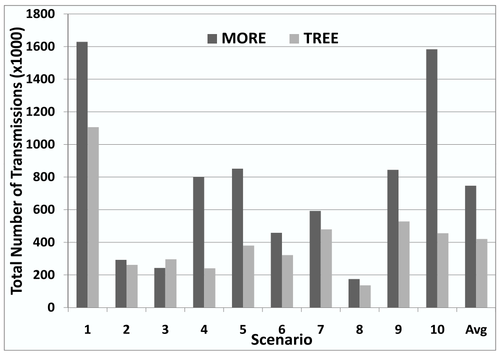
The higher throughput achieved by TREE compared to MORE can be explained by the fewer FNs and lower total number of transmissions in the former compared to the latter. In particular, Figure 3.3(b) shows that the use of a tree instead of a union of belts results in on average 36% fewer FNs in TREE than in MORE. Note that in some cases, TREE uses equal or 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,



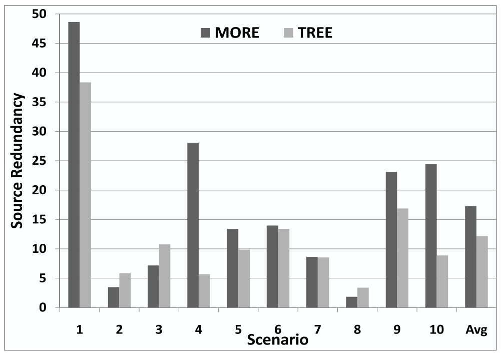
(a) Throughput



(b) Number of FNs



(c) Total # of Transmissions

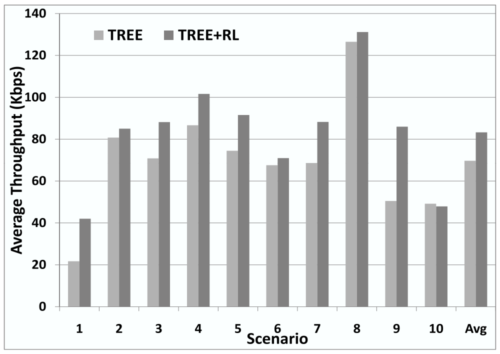


(d) Source Redundancy

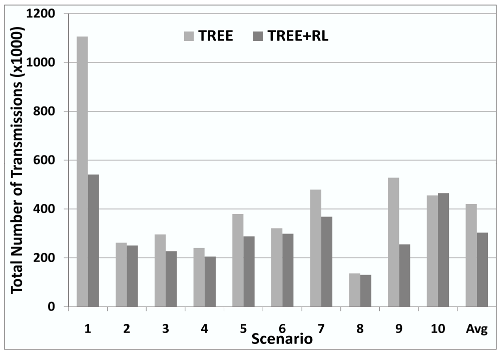
Fig. 3.3. Throughput, number of FNs, total number of transmissions, and source redundancy with MORE and TREE for 10 different scenarios.

since there is no random, threshold-based pruning. Figure 3.3(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 3.3(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.

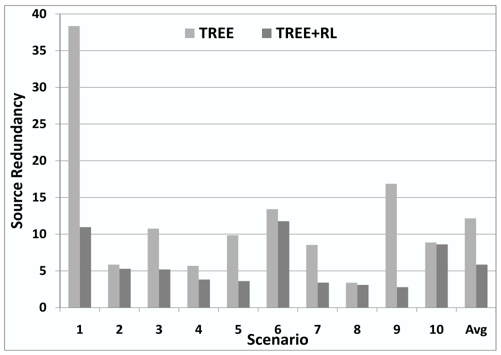
**Impact of source rate limiting.** We next evaluate the impact of backpressure-based rate limiting at the source, as implemented in the TREE+RL version of *Pacifier*. Figure 3.4(a) shows that the use of rate limiting at the source improves the



(a) Throughput



(b) Total # of Transmissions



(c) Source Redundancy

Fig. 3.4. Throughput, total number of transmissions, and source redundancy with MORE and TREE+RL for 10 different scenarios.

throughput by 5% (Scenario 6) to 94% (Scenario 1), with an average of 20%, compared to TREE. Figure 3.4(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 3.4(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.

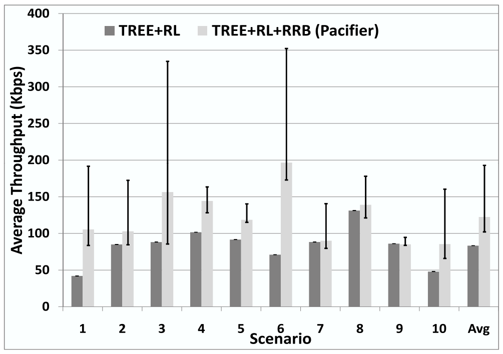


Fig. 3.5. Throughput with TREE+RL and TREE+RL+RRB (*Pacifier*) for 10 different scenarios. The error bars show throughput of the best and the worst receiver.

**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 3.5 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

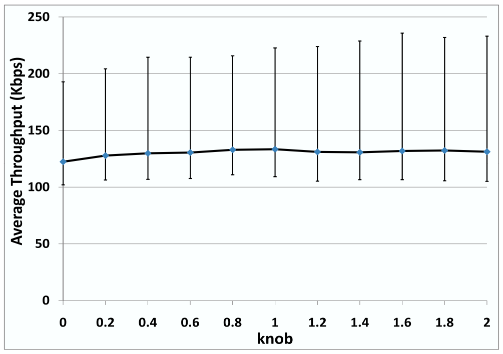


Fig. 3.6. Average throughput with *Pacifier* as a function of *knob*. The average is taken over 10 scenarios. The error bars show average max and min values over the 10 scenarios.

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.

**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 3.3.3. We vary the value of *knob* from 0 (the version evaluated before) to 2. 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 3.6 shows the average, max, and min throughput with *Pacifier*, as *knob* varies from 0 to 2. Every point is the average over 10 scenarios. Somewhat surprisingly, higher *knob* values improve the max throughput and $knob = 1$ appears to maximize the average and the min throughput. $knob = 0$, which is expected to optimize the performance of the best receiver, achieves on average the lowest max, average, and min throughput, compared to all the other *knob* values. This confirms our speculation in Section 3.3.3 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.

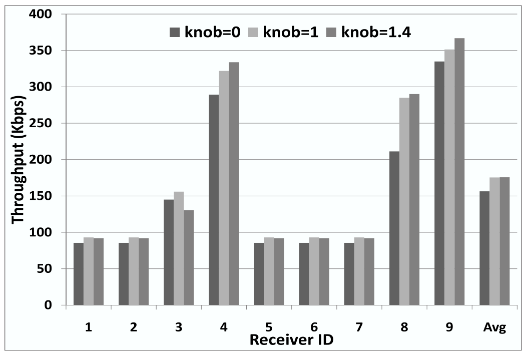
An additional counter-intuitive observation from Figure 3.6 is that the throughput does not change monotonically as the *knob* increases. The reason for this behavior is that the the TX_credits assigned to FNs actually interfere in a very complex way. In a nutshell, increasing the TX_credit of an FN $j$ can potentially decrease the TX_credit of its child nodes, as the grandchild nodes of $j$ now have more chance of overhearing $j$'s transmissions. Consequently, the chance of packet reception at $j$'s grand-grandchild nodes from their upstream nodes is affected in complicated ways.[5]

This complex interdependence of throughput, *knob* values, and TX_credits is shown in Figure 3.7. In this figure, we plot the individual throughput for each of the 9 receivers, with 3 different *knob* values (0. 1, and 1.4) in 3 different scenarios (Figures 3.7(a), 3.7(c), 3.7(e)), and initial TX_credits of the FNs in the same scenarios, for the same *knob* values (Figures 3.7(b), 3.7(d), 3.7(f)). We picked up Scenario 3, which had the highest max throughput for most of the *knob* values, Scenario 10, which had the lowest min throughput for all *knob* values, and Scenario 8.

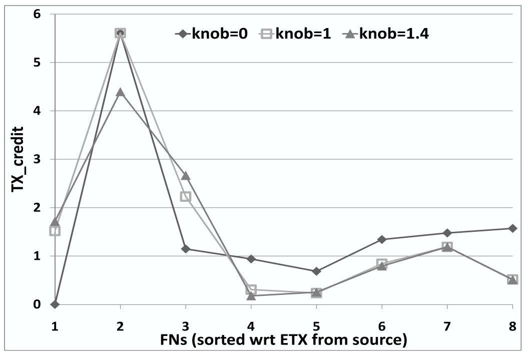
In Figure 3.7(a), we observe that increasing the *knob* value improves throughput for the three best receivers (9, 4, and 8). However, for the remaining 6 receivers, throughput increases when *knob* changes from 0 to 1 but it reduces again when it changes form 1 to 1.4. Finally, the average throughput is same with *knob* 1 and 1.4, slightly larger than with *knob* 0.

In Figure 3.7(c), we observe that the best receiver changes when *knob* changes from 0 to 1. With *knob* 0, the best receiver is receiver 9, closely followed by receiver 8 (their throughput differ only by 3%). But changing the *knob* to 1 results in a large throughput improvement for receiver 8 (by 46%), which now becomes the best receiver, while throughput of receiver 9 only increases by 8%. Also, when moving

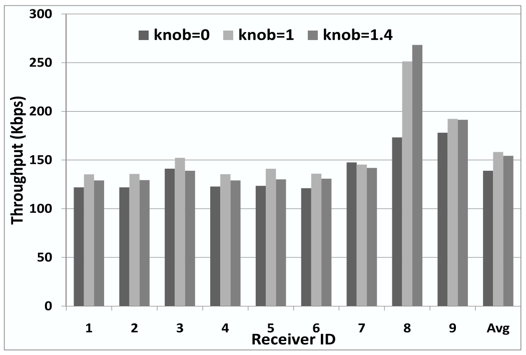
[5]Recall our TX_credit calculation is a polynomial heuristic; optimal TX_credit assignment to all FNs is an NP-hard problem.



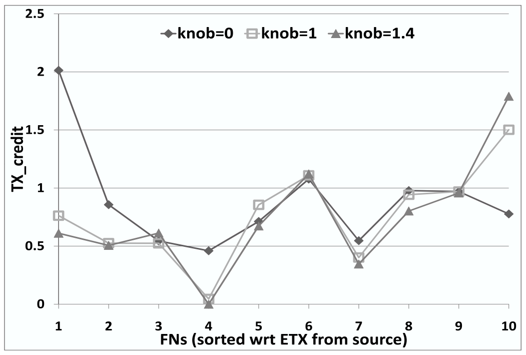
(a) Receiver Throughput - Scenario 3.



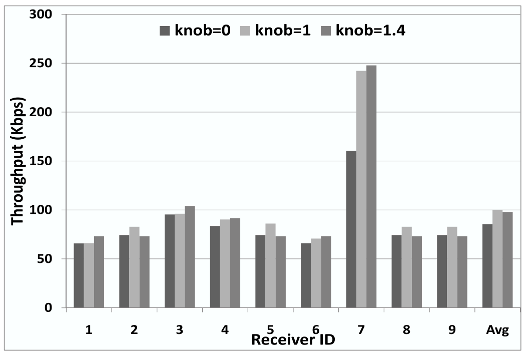
(b) FN TX_credits - Scenario 3.



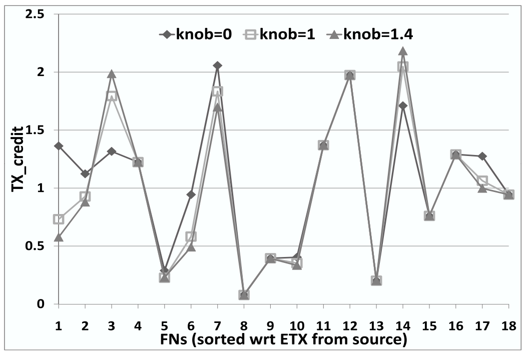
(c) Receiver Throughput - Scenario 8.



(d) FN TX_credits - Scenario 8.



(e) Receiver Throughput - Scenario 10.



(f) FN TX_credits - Scenario 10.

Fig. 3.7. Individual receiver throughput and FN TX_credits with 3 different *knob* values in 3 scenarios. FNs are sorted in increasing ETX distance from the source.

to *knob* 1.4, throughput of the best receiver increases even more, but the average throughout decreases compared to with *knob* 1 (although it remains higher than with *knob* 0).

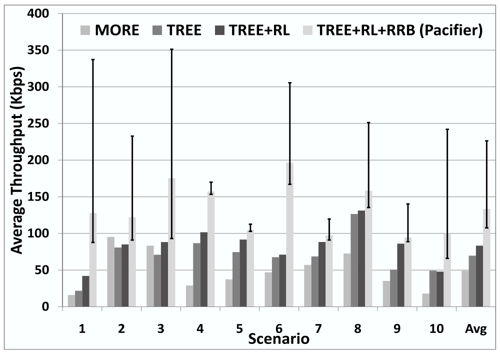
Finally, in Figure 3.7(e) throughput of the three best receivers (7, 3, and 4) and of the worst receiver (6) increases as the *knob* increases. However, for the remaining 5 receivers throughput only increases when *knob* changes from 0 to 1 but it decrease when it changes from 1 to 1.4, and the average throughput also exhibits the same behavior.

For each of the 3 scenarios, we observe that the TX_credits of the FNs follow very different patterns as shown in Figures 3.7(b), 3.7(d), 3.7(f). In Scenario 3, TX_credits are in general higher with *knob* 0. In Scenario 8, there is no specific pattern – we can observe the TX_credit of the 4th FN going to 0, for large *knob* values. Finally, in Scenario 10, a large fraction of the FNs maintains the same TX_credits for different *knob* values.

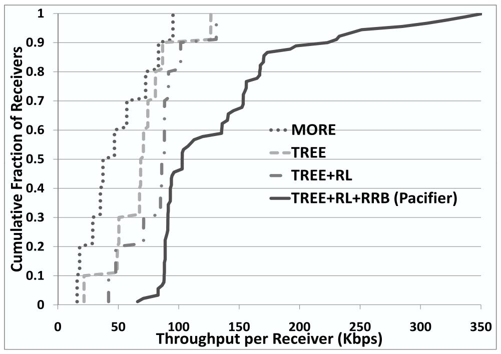
In summary, the discussion above shows that there is no optimal value for *knob*. We find setting *knob* = 1 in *Pacifier* appears to improve the max throughput while maximizing the average and the min throughput.

**Overall comparison** Figure 3.8(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 3.8(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



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



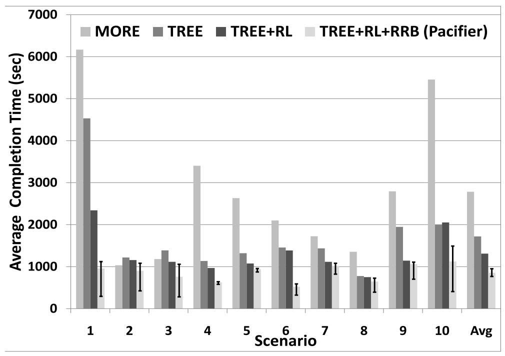
(b) CDF of the 90 throughput measurements obtained with each protocol for 10 scenarios with 9 receivers each.

Fig. 3.8. Overall throughput comparison of MORE, TREE, TREE+RL, and *Pacifier*.

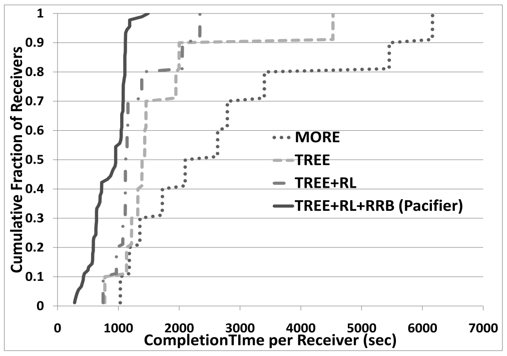
(equal to that of the worst receiver) due to the “crying baby” problem. In contrast, with *Pacifier*, receivers finish independently of each other and the CDF has a continuous form. In the median case, *Pacifier* outperforms TREE+RL, TREE, and MORE by 20%, 49%, and 178%, respectively.

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.

Figures 3.9(a), 3.9(b) compare the four protocols in terms of download completion time. In Figure 3.9(a), we observe that *Pacifier* reduces the download completion time of the best receiver in all 10 scenarios compared to MORE by a factor of 2.1x-21x



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



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

Fig. 3.9. Download completion time comparison of MORE, TREE, TREE+RL, and TREE+RL+RRB (*Pacifier*).

(scenarios 7 and 1, respectively). It also reduces the download completion time of the worst receiver (i.e., the "crying baby") in 9 out of 10 scenarios compared to MORE by a factor of 1.1x-5.5x (scenarios 3 and 1, respectively), and only increases it in one scenario (scenario 2) by 5%. In Figure 3.9(b), we observe that *Pacifier* improves the median download completion time compared to MORE by a factor of 2.2x, the 90th percentile (best receivers) by a factor of 4.9x and the 10th percentile (worst receivers) by a factor of 2.4x.

## 3.5 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.

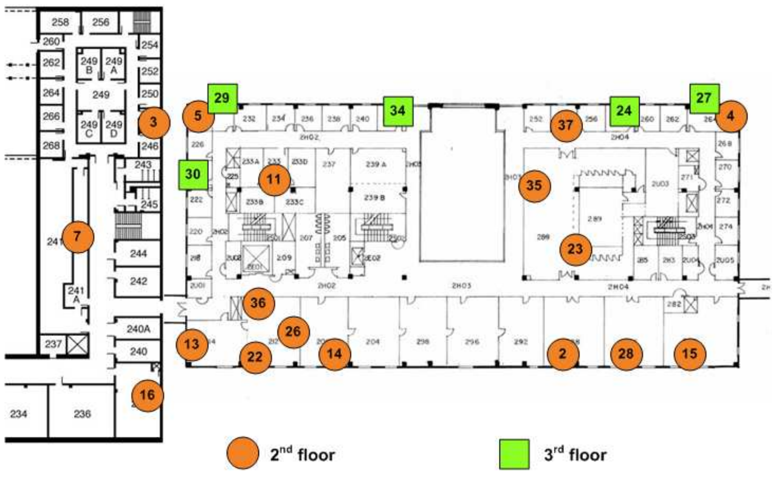


Fig. 3.10. A schematic of the MAP testbed.

### 3.5.1 Testbed description

Our testbed, Mesh@Purdue (MAP) [62], currently consists of 22 mesh routers (small form factor desktops) deployed on two floors of two academic buildings on the Purdue University campus. A schematic of the testbed is shown in Figure 3.10. Each router has two radios. For this study, we used one of them: the Atheros 5212 based 802.11a/b/g wireless radio operating in b ad hoc mode. Each radio is attached to a 2dBi rubber duck omnidirectional antenna with a low loss pigtail to provide flexibility in antenna placement. Each mesh router runs Mandrake Linux 10.1 (kernel 2.6.8-12) and the open-source *madwifi* driver [63] 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.

### 3.5.2 Implementation details

Network coding-based wireless protocols (e.g., [22, 64]) 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 [22].[6] 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 [65], 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. 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[7] 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).

#### 3.5.2.1 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*

[6]The publicly available implementation of MORE [65] using the Click modular router from the authors of [22] currently supports only unicast.

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

*positive.* A layer-2.5 implementation [22] 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.

#### 3.5.2.2 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 3.3.3). 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.

**ACK reliability.** To provide reliability, the ACKs in MORE are *unicast* at the MAC layer. In contrast to the 802.11 broadcast mode, the 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 [22], ACKs are sent to the source over the shortest ETX path to ensure quick propagation. In addition, in [22], 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 filed") of the IP header, which can be set using *setsockopt* at the application layer, and the priority properties in Linux routing [66]. The basic queuing discipline in Linux, *pfifo_fast*, is a three-band first-in, first-out queue. Each band is *txqueuelen* packets long, as configured with *ifconfig*. In our implementation, we set *txqueuelen = 5*, as mentioned in 3.5.2. Packets are enqueued in the three bands based on their TOS bits. The three bands, 0, 1, 2, have different priorities, with band 0 having the highest priority and band 2 having the lowest priority. Packets from a given band are dequeued only when all higher priority bands are empty. By default, the TOS bits are set to 0000 and packets are enqueued in band 1. For ACKs, we set them to 1010. This combination corresponds to "minimum delay + maximum reliability" (or "mr+md") and enqueues the ACKs in the highest priority band 0.

Another factor that caused significant delay to the ACK packets and resulted in very low throughput was the ARP messages. Since ACKs are unicast at the MAC layer, the sender of an ACK first sends an ARP request before the actual transmission of the ACK packet, in order to learn the MAC address that corresponds to the IP address of the next hop. If no reply is received, the ARP request is retransmitted after a timeout (the default is 1 sec). Unfortunately, both the ARP requests and the ARP replies are *broadcast* at the MAC layer. Since 802.11 broadcast implements no reliability mechanism for broadcast frames, ARP messages are susceptible to loss due to poor channel conditions or collisions (under heavy traffic). Indeed, we observed in our experiments that sometimes ARP requests were retransmitted up to 90 times, which resulted in a 1.5 min delay, before the actual ACK was sent. To deal with this problem, before each experiment, we cached permanently at each node on the shortest ETX path from a receiver to the source, the IP-MAC mapping of the next hop, using the *ip* command, thus completely eliminating the exchange of ARP messages during the experiment.

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 3.4.1). The source code for the two protocols and the ETX module together is over 9000 lines of C code.

### 3.5.3 Experimental setup

In the implementation of the two protocols we used the same parameters as in our simulation study in Section 4.4, i.e., the batch size was $k = 32$ packets, the random coding coefficients were chosen from a Galois Field of size $2^8$, and the *knob* value for *Pacifier* was again set to 1. 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%.

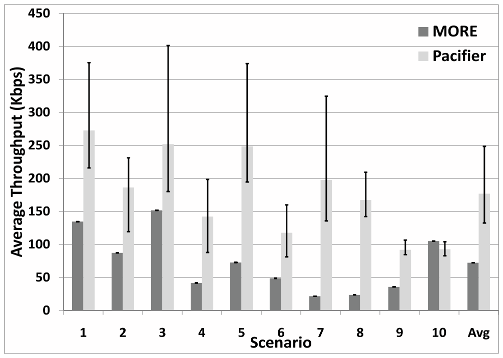
We ran each protocol on 10 different scenarios (i.e., selection of source and multicast group members). In each scenario, 1 source and 4 receivers were randomly selected among the 22 nodes of our testbed. In each scenario, we first ran the ETX module for 10 minutes to collect the pairwise loss rates 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), and we present separately the results for each day.

### 3.5.4 Experimental results

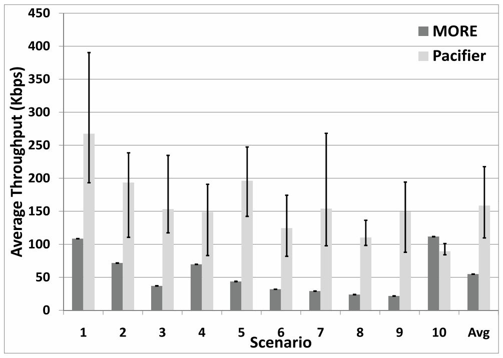
Figures 3.11(a)-3.11(d) show the average throughput achieved with MORE and *Pacifier* in each of the 10 scenarios, as well as the throughput of the best and the worst receiver (top and bottom of error bars) in each scenario, on 4 different days.

Similar to the simulation results, we observe that *Pacifier* outperforms MORE in 9 out of 10 scenarios on all 4 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 3.8(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.

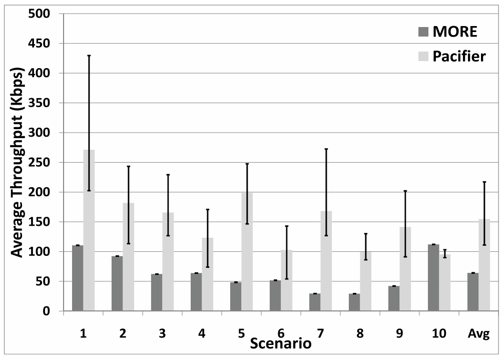
We observe again that *Pacifier* solves the "crying baby" problem, allowing well-connected receivers in each case to achieve throughputs much higher than the average value, while also improving throughput of the worst receivers in almost all scenarios. Averaging over 10 scenarios for each of the 4 days, the throughput of the best receiver with *Pacifier* is 244%, 302%, 239%, and 259% higher than with MORE, but, in some cases, it can be much higher, e.g., more than 8x in Scenario 7, Days 2 and 3, and Scenario 8, Day4, more than 10.5x in Scenario 6, Day 4, 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%, 101%, 74%, and 53% higher than with MORE on



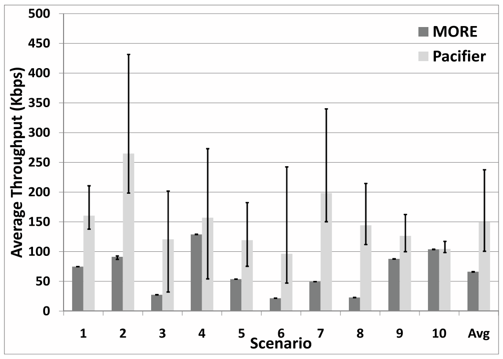
(a) Day 1



(b) Day 2



(c) Day 3

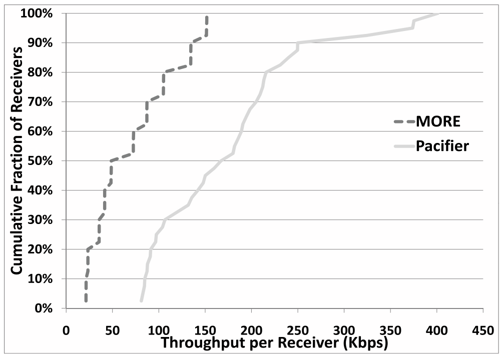


(d) Day 4

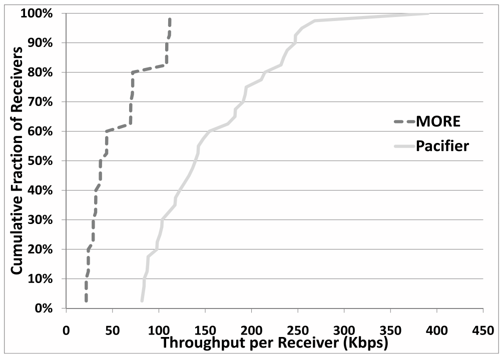
Fig. 3.11. Testbed throughput comparison of MORE and *Pacifier* in 10 different scenarios and 4 different days.

each of the 4 days, and the maximum improvement can be as high as 5.4x (higher than in the simulation results) in Scenario 7, Day 1.

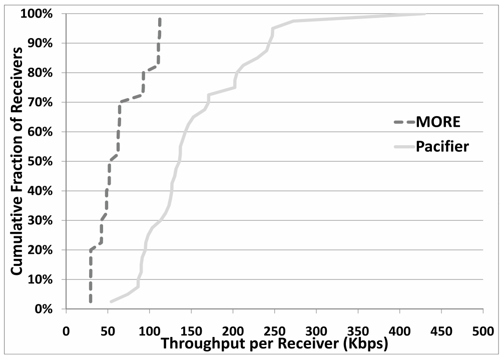
Figures 3.12(a)-3.12(d) plot the CDF of the 40 throughput values obtained from the 10 scenarios with 4 receivers each, for the two protocols, on each of the 4 days. Similar to Figure 3.8(b) for the simulation results, we observe that the CDFs for MORE exhibit a staircase form, since for each scenario, all 4 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 always has a continuous form. *Pacifier* outperforms MORE on all 4 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*



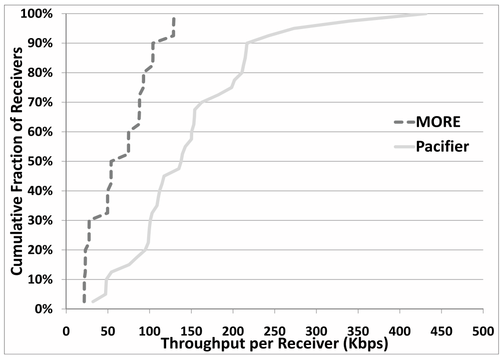
(a) Day 1



(b) Day 2



(c) Day 3



(d) Day 4

Fig. 3.12. CDFs of 40 testbed throughput measurements obtained with MORE and *Pacifier* for 10 scenarios with 4 receivers each on 4 different days.

than with MORE. This again shows that *Pacifier* not only solves the “crying baby” problem, it also simultaneously offers a significant improvement to the performance of the “crying baby” itself.

## 3.6 Summary

Designing high-throughput, reliable multicast protocols faces two challenges: the inherent lossiness of wireless links and the “crying baby” problem. In this chapter, we presented *Pacifier*, the first practical network coding-based high-throughput, reliable multicast protocol for WMNs. *Pacifier* seamlessly integrates tree-based OR, intra-flow network coding, 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 protocol 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 the throughput of the worst receivers.

# 4. EFFICIENT NETWORK CODING BASED OPPORTUNISTIC ROUTING THROUGH CUMULATIVE CODED ACKNOWLEDGMENTS

## 4.1 Introduction

In this chapter, we propose a novel solution to a fundamental challenge in network coding based opportunistic routing: *How many coded packets should each forwarder transmit?* This challenge, if not efficiently addressed, may prevent network coding based opportunistic routing protocols from realizing the maximum possible gains. We begin by explaining this challenge in more detail in 4.1.1, and then review existing approaches and their drawbacks in 4.1.2. The drawbacks of the existing approaches motivate our approach which we present in 4.1.3.

### 4.1.1 The challenge in network coding based opportunistic routing protocols

We illustrate the main challenge in network coding based opportunistic routing protocols, e.g., MORE, with the example shown in Figure 4.1. This figure shows a

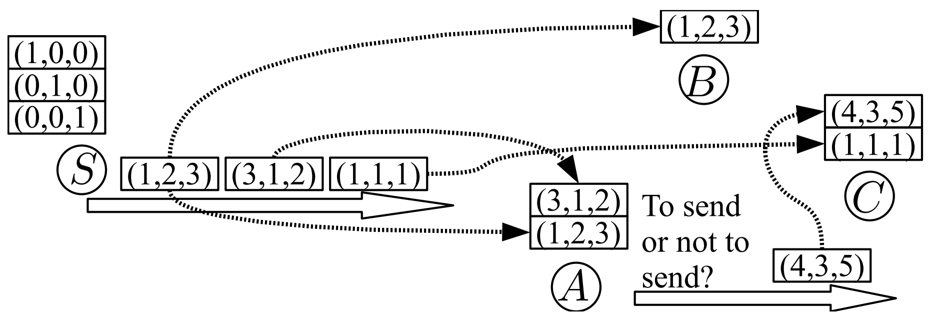


Fig. 4.1. The importance of knowing how many coded packets to transmit.

typical scenario of a network coding based opportunistic routing protocol. The source $S$ has three downstream FNs $A$, $B$, and $C$. Assume for simplicity that $S$ has three innovative packets $X_1$, $X_2$, and $X_3$ to send. Instead of transmitting the native packets, $S$ transmits three coded packets $X_1 + X_2 + X_3$, $3X_1 + X_2 + 2X_3$, and $X_1 + 2X_2 + 3X_3$ in sequence, which are denoted by the corresponding *coding vectors* $(1, 1, 1)$, $(3, 1, 2)$, and $(1, 2, 3)$. Assume that $(1, 1, 1)$ coded packet is received by $C$, and the $(3, 1, 2)$ and $(1, 2, 3)$ packets are received by $A$ and by $\{A, B\}$, respectively. The downstream FNs $A$, $B$, and $C$ have received a sufficient amount of innovative packets. Collectively, the three FNs can now act as the new source and the original source $S$ should stop transmission. However, it is a non-trivial task for $S$ to know whether its downstream FNs have accumulated a sufficient amount of innovative packets.

The same challenge exists for the intermediate FN $A$. After transmitting a useful coded packet $(4, 3, 5)$, which is received by FN $C$, $A$ has to decide whether it should continue or stop sending coded packets. Furthermore, $A$ has limited knowledge about the reception status of the three packets transmitted by $S$ (e.g., $A$ may not know that $C$ has received $(1, 1, 1)$ from $S$), which makes the decision of whether to stop transmission even harder for $A$ than for the source $S$.

Note that overhearing, a common way of acknowledging non-coded wireless traffic, does not suit network coded traffic. Consider the same example in Figure 4.1. $C$ generates a coded packet $c_1(1, 1, 1) + c_2(4, 3, 5)$. If the randomly chosen coefficients happen to be $c_1 = c_2 = 1$, then a $(5, 4, 6)$ packet is sent. Suppose $A$ overhears this new packet. If $A$ were aware of the reception of the $(1, 1, 1)$ packet by $C$ and also knew the coefficients $c_1 = c_2 = 1$, then $A$ could deduce that the previously transmitted $(4, 3, 5)$ packet was received successfully since $((5, 4, 6) - 1 \cdot (1, 1, 1))/1 = (4, 3, 5)$. Nonetheless, in practice, neither piece of the information is available to $A$, it is thus impossible for $A$ to know whether the $(4, 3, 5)$ packet is received or not by only overhearing the $(5, 4, 6)$ packet sent by $C$.

One way to address the challenge is to combine individual packet overhearing, as in non-coding based protocols, with a *credit system*, based on coded transmissions, and have the forwarders perform detailed bookkeeping to guarantee credit conservation in the system. This approach is taken in MC$^2$ [56]. Although theoretically optimal [67], this approach is quite complex in practice. In addition, like every approach that relies on individual packet overhearing, it requires a reliable control plane. In typical WMN environments with high packet loss rates or contention [13], this approach can cause excessive signaling overhead and retransmissions, which can significantly limit the performance.

### 4.1.2 Offline loss rate based approaches

Since theoretically optimal solutions are hard to implement in practice, existing network coding based opportunistic routing protocols use heuristics based on link loss rates, to address the challenge in a simple manner, and to minimize the control overhead.

As we have described in Section 2.3.1, MORE [22], the first network coding based opportunistic routing protocol, employs an offline approach which requires no coordination among FNs. In MORE, the source calculates and assigns a *transmission credit* to each FN, using the ETX metric [18], computed from loss rate measurements. Receptions from upstream nodes are then used to trigger new transmissions at the FNs, with precomputed relative frequencies using the transmission credits. Since the ETX metric expresses the *expected* behavior, the approach used in MORE cannot guarantee that the destination will always receive enough packets, due to the randomness of the wireless channel. Hence, the source in MORE keeps transmitting packets from the same batch until it receives an ACK from the destination, unnecessarily increasing interference.

Many other works that improve MORE also use offline measured loss rates as a basic component in their proposed solutions (e.g., [60, 68, 69] and our proposed *Pacifier* in Chapter 3).

The drawback of all these approaches is that performance heavily depends on the accuracy and freshness of the loss rate measurements. Loss rate estimates are obtained through periodic probing and are propagated from all nodes to the source. Apparently, the higher the probing frequency, the higher the accuracy, but also the higher the overhead. As a recent study [70] showed, even low-rate control overhead in non-forwarding links can have a multiplicative throughput degradation on data-carrying links.

To reduce this overhead, the authors of MORE collect the loss rates and calculate the credits only in the beginning of each experiment. In practice, this suggests that loss rate measurements should be performed rather infrequently. Unfortunately, recent WMN studies [58, 71] have shown that, *although link metrics remain relatively stable for long intervals in a quiet network, they are highly sensitive to background traffic*. For example, in [71], the authors observe that 100 ping packets (one per second) between two nodes in a 14-node testbed caused an increase of 200% or more to the ETT [72] metric of around 10% of the links.[1] Even worse, a 1-min TCP transfer between two nodes in the same network caused an increase of more than 300% to the ETT metric of 55% of the links.

In summary, since *acknowledging* network coded traffic online is expensive, solutions based on loss rate measurements try to avoid that by *predicting* offline the amount of coded traffic each FN should send. Although attractive because of their simplicity, and their minimal coordination overhead, these approaches may suffer significant performance degradation in dynamic wireless environments with continuously changing levels of channel quality, interference, and background traffic. Overestimated loss rates cause redundant transmissions, which waste wireless bandwidth. On the other hand, underestimated loss rates may have an even worse impact, since nodes may not transmit enough packets to allow the destination to decode a batch. Increasing the frequency of the loss rate measurements is not always a solution, since the

[1]The ETT metric estimates the quality of a link taking into account both the loss rate (through the ETX metric) and the link bandwidth.

incurred overhead may counterbalance the low coordination overhead. This motivates the need for a new approach, *oblivious* to loss rates.

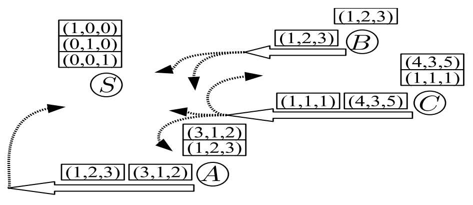
### 4.1.3 Our approach – Cumulative Coded Acknowledgments

In this chapter, we present a novel approach to addressing the main challenge in network coding based opportunistic routing protocols, i.e., determining how many coded packets each FN should transmit, through the design of CCACK, a new efficient network coding based opportunistic routing protocol. Unlike existing protocols, FNs in CCACK decide how many packets to transmit in an online fashion, and this decision is completely oblivious to link loss rates.[2] This is achieved through a novel **C**umulative **C**oded **ACK**nowledgment scheme that allows nodes to acknowledge network coded traffic to their upstream nodes in a simple and efficient way, with practically *zero overhead.* In other words, unlike existing network coding based opportunistic routing protocols which use network coding to avoid sending feedback, CCACK *encodes* feedback to exploit its benefits while hiding its overhead. Feedback in CCACK is not required strictly on a per-packet basis; this makes the protocol resilient to individual packet loss and significantly reduces its complexity, compared to [56].

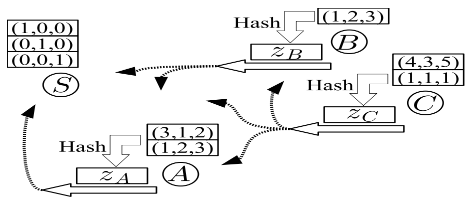
Take the scenario in Figure 4.1 as a continuing example. One naive approach to ensure that $S$ (resp. $A$) knows when to stop transmission is through the use of *reception reports*, for which each node broadcasts *all the basis vectors* of the received linear space to its upstream nodes, as illustrated in Figure 4.2(a).[3] In this figure, node $C$ broadcasts its two corresponding basis vectors, $(1, 1, 1)$ and $(4, 3, 5)$. Similarly, $A$ and $B$ transmit the basis vectors back to their upstream nodes. $S$ (resp. $A$) can use the overheard basis vectors to reconstruct the *collective knowledge space* (CKS) of

[2]By "oblivious to link loss rates" we mean here that loss rates are not taken into account in determining how many packets each FN should transmit. We still use MORE's loss rate based offline algorithm in CCACK to build the FN belt, for a fair comparison between the two protocols. We note though that the coded feedback mechanism in CCACK is orthogonal to the belt construction.

[3]We sometimes refer to the linear space spanned by the received vectors as the *knowledge space.*



(a) Uncoded Feedback



(b) Coded Feedback

Fig. 4.2. Different types of feedback for network-coded traffic.

the downstream nodes. $S$ (resp. $A$) then decides whether to continue transmission depending on whether the CKS of the downstream nodes is strictly larger than the local knowledge space at $S$ (resp. $A$).

An obvious drawback of this approach is the size of the feedback messages. For practical network coding with symbol size GF($2^8$) and batch size 32, each coding vector contains 32 bytes. To convey a space of dimension $\kappa \gg 1$ thus requires $\kappa$ 32-byte vectors, which is too large to piggyback to normal forward traffic. The unreliability of the wireless channel further exacerbates the problem as the $\kappa \times 32$-byte feedback messages need to be retransmitted several times until they are overheard by all the upstream nodes.

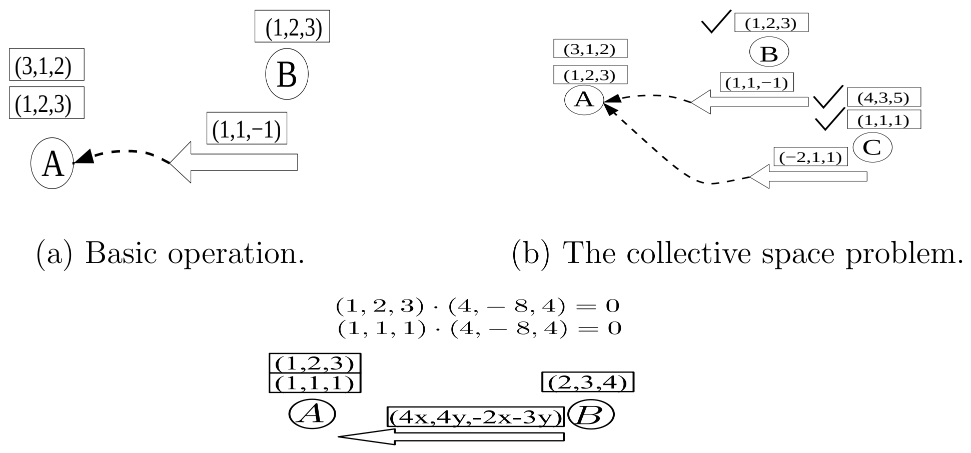
In contrast, in CCACK each node uses *a single coded feedback vector* to represent the entire space, which may consist of $\kappa \gg 1$ basis vectors. In the broadest sense, the three coded acknowledgment vectors $z_A$ to $z_C$ in Figure 4.2(b) serve as a hash for their corresponding spaces. As will be explained in Section 4.3, we have devised a simple mechanism that successfully *compresses* (most of) the space information into a single vector, say $z_A$ for node $A$, while allowing upstream nodes to *extract* the original space from $z_A$ without exchanging any additional control information. Each single vector $z_A$ can be easily piggybacked to the forward data traffic. This compressed/coded acknowledgment is critical to the efficiency since in CCACK overhearing any of the data packets of $A$ with piggybacked coded ACK will convey to the upstream nodes the entire space (or most of the space) of $A$. This thus drastically reduces the need of retransmitting feedback information over the unreliable wireless channel.

In addition to efficiently solving the challenge of how many packets each FN should transmit, the cumulative coded acknowledgment scheme in CCACK enables us to develop an efficient rate control algorithm. In contrast, MORE has no explicit rate control mechanism and its performance degrades as the number of flows in the network increases [22, 67–69].

To evaluate CCACK, 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 network coding and opportunistic routing, and are conducted over a variety of network topologies. Our results show that CCACK improves both throughput and fairness over MORE, by 27-45% and 5.8-8.8%, respectively, on average, with different number of flows. For some challenged flows which completely starve under MORE, CCACK increases throughput by up to 21x and fairness by up to 124%.

In addition, the coding and memory overheads of CCACK are comparable to those of MORE, making it easily deployable on commodity hardware. To demonstrate this, we present an application layer implementation of CCACK and MORE on Linux and their performance evaluation on a 22-node 802.11 WMN testbed deployed in two academic buildings at Purdue University. Although the small size of our testbed along with the limitations of our implementation limit the potential gains, our testbed results show that CCACK improves both throughput and fairness, by up to 3.2x and 83%, respectively, with average improvements of 11-36% and 5.7-8.3%, respectively, for different numbers of flows, validating the benefits of our approach.

The remaining of this chapter is organized as follows. In Section 4.2, we introduce the basic principles of coded feedback through a simple existing coded feedback scheme. We identify two problems with this scheme which motivate the design of CCACK, presented in Section 4.3. Section 4.4 evaluates the performance of CCACK



(a) Basic operation.

(b) The collective space problem.

(c) The false positive problem.

Fig. 4.3. Null-Space-Based (NSB) coded feedback.

and MORE through extensive simulations and Section 4.5 describes the implementation and evaluation of CCACK and MORE on a wireless testbed. Finally, Section 4.6 concludes the paper.

## 4.2 A simple Coded Feedback Scheme

One candidate solution (which has been used in the past in a different context [36]), attractive due to its simplicity, is null-space-based (NSB) coded feedback, i.e., each node sends to each upstream node one vector randomly chosen among all vectors in the null space of the innovative vectors the node has received in the past. Take for example Figure 4.3(a). Let $B_v$ denote the buffer containing the innovative coding vectors received by an FN ($B_v$ contains two vectors $(1, 2, 3)$ and $(3, 1, 2)$ at node $A$ and one vector $(1, 2, 3)$ at node $B$ in Figure 4.3(a)). Every time $B$ broadcasts a coded data packet to its own downstream nodes, it also appends to the packet header an ACK vector $z_B$ satisfying:

$$z_B \cdot v = 0, \quad \forall v \in B_v \tag{4.1}$$

Namely, the inner product between $z_B$ and $v \in B_v$ is zero. There may be multiple choices of $z_B$ that satisfy (4.1) (e.g., in Figure 4.3(a), $z_B$ can be any vector of the form $(-2x - 3y, x, y)$). $z_B$ is then chosen *uniformly randomly* among all valid vectors satisfying (4.1). Let $S_B = \langle v : v \in B_v \rangle$ denote the linear space spanned by vectors in $B_v$. One can easily show that:

**Lemma 1** *With the above random construction of $z_B$, any vector $v' \in S_B$ must satisfy $z_B \cdot v' = 0$. Moreover, for any vector $v'' \notin S_B$ we have $\mathsf{prob}(z_B \cdot v'' = 0) = \frac{1}{2^8}$ assuming the* GF($2^8$) *finite field is used.*

From the above lemma, when the upstream node $A$ hears such a packet from $B$, it simply needs to compute the inner product of its own innovative vectors with $z_B$. In Fig. 4.3(a), suppose that $z_B$ is chosen as $(0, 1, -1)$. Since $(1, 2, 3) \cdot (1, 1, -1) = 0$, $A$ concludes that $B$ has received packet $(1, 2, 3)$. On the other hand, since $(3, 1, 2) \cdot (1, 1, -1) = 2 \neq 0$, $A$ concludes that packet $(3, 1, 2)$ is an innovative packet for $B$, and hence it should send more coded packets to $B$.

Although attractive due to its simplicity, the NSB feedback scheme suffers from two significant limitations when used in network coding-based opportunistic routing.

**Problem 1: The collective space problem.** Take Figure 4.1 for example. Nodes $B$ and $C$ would like to convey their space information to $A$ so that $A$ can stop packet transmission. Based on the NSB concept, $B$ and $C$ send $z_B = (1, 1, -1)$ and $z_C = (-2, 1, 1)$, respectively, which are orthogonal to their local innovative vectors (see Figure 4.3(b)). The idea is to hope that, upon the reception of $z_B$ and $z_C$, $A$ will know that the knowledge spaces of $B$ and $C$ have *collectively* covered the local knowledge space of $A$ and thus will stop transmission.

Nonetheless, when $A$ checks the inner product of the coded feedback and its own innovative packets, we have

$$z_B \cdot (3, 1, 2) = 2 \neq 0 \text{ and } z_C \cdot (3, 1, 2) = -3 \neq 0.$$

Therefore $A$ *thinks that the coding vector* $(3, 1, 2)$ *is innovative to both its downstream nodes and thus continues transmission even when collectively $B$ and $C$ already have* *enough information.* This misjudgment is caused by that the NSB coded feedback does not convey the collective space of all downstream nodes but only the space relationship between the individual pairs (e.g., $A$ vs. $B$ and $A$ vs. $C$). Therefore, if we apply the NSB coded feedback as in [36] to unicast opportunistic routing, $A$ will not stop transmission until one of its downstream nodes has a local knowledge space that completely covers the local knowledge space of $A$. This defeats the purpose of opportunistic routing.

**Problem 2: Non-negligible false-positive probability.** Take Figure 4.3(c) for example. $A$ wants to send two packets to $B$ and a network coded packet has been received by $B$ already. To convey its local knowledge space back to $A$, $B$ sends an orthogonal vector $z_B$ satisfying (4.1), which is randomly chosen to be any vector of the form $z_B = (4x, 4y, -2x - 3y)$. Suppose that $B$ chooses $z_B = (4, -8, 4)$ and $A$ receives such $z_B$. *Since $z_B$ is orthogonal to all the innovative vectors of $A$, $A$ will wrongfully conclude that the knowledge space of $B$ covers the local knowledge space of $A$. $A$ thus attempts no further transmission.* Although Lemma 1 guarantees that this false positive event happens only with probability $\frac{1}{2^8}$, its impact to the system performance is significant. The reason is that in a multi-hop transmission, any single hop that experiences this false positive event will cause an upstream node to stop transmission prematurely. The communication chain is thus broken and the destination may not be able to receive enough independent packets for decoding. Although one can fix this false-positive issue by retransmitting another $z_B$ vector, the necessary timer management for the unreliable feedback channel and the additional interference caused by retransmission easily negate the benefits of sending coded feedback.

## 4.3 CCACK design

In this section, we present the design of CCACK. We begin with an overview of the protocol and then we describe its two main components: construction of a novel

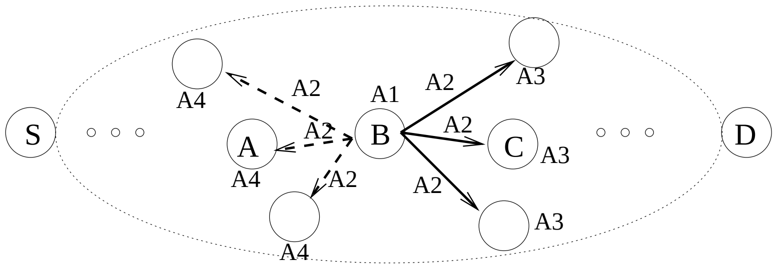


Fig. 4.4. Overview of CCACK operations. **A1**: Node B creates a coded packet and a coded feedback ACK vector, stores coding vector in $B_w$. **A2**: Node B broadcasts the packet with the ACK vector appended. **A3**: Node C (and other downstream nodes) store the packet in $B_u$ and in $B_v$ if it is innovative. **A4**: Node A (and other upstream nodes) process the ACK vector (H_test with all vectors stored in $B_u$, $B_w$).

cumulative coded feedback scheme which addresses the two problems we discussed in Section 4.2, and a rate control algorithm, built upon this coded feedback scheme.

### 4.3.1 CCACK overview

An overview of CCACK 's operation is shown in Figure 4.4.

The source and the intermediate FNs in CCACK use intra-flow random linear network coding. We selected a batch size of $N = 32$ packets and the random coefficients for each linear combination are selected from a Galois Field (GF) of size $2^8$, same as in [22, 56, 60].

Nodes in CCACK maintain per flow state, which includes the *current batch* of the flow, a *credit counter* (Section 4.3.4), and three buffers: a packet buffer $B_v$, and two coding vector buffers $B_u$ and $B_w$. With the exception of $B_u$ and $B_w$, all the other information is also maintained in MORE. The size of $B_v$ is equal to the batch size $N$, since the number of innovative packets is bounded by the batch size. The size $B_u$ and $B_w$ can be larger, since these buffers only store 32-byte coding vectors and not whole packets. In our implementation we used a size equal to $5 \times N$. Similar to MORE, this information is soft-state and it is flushed if no packet for a flow is received for 5 minutes.

The source and the FNs broadcast randomly mixed packets and store the coding vectors of these packets in $B_w$. Whenever a node overhears a packet, the node first checks whether the packet is innovative by comparing the coding vector to those of the existing packets in $B_v$. If innovative, the newly received packet is stored in $B_v$, similarly to MORE and other existing network coding based opportunistic routing protocols. Regardless being innovative or not, the node also checks whether the newly received packet is from an upstream node. If yes, then it stores the forward coding vector in $B_u$.

Each forward coding vector in $B_u$ and $B_w$ can be marked as $\mathsf{H}$ (heard by a downstream node) or $\neg\mathsf{H}$ (not heard). A coding vector is marked as $\neg\mathsf{H}$ when initially is inserted in either of the two buffers, since the node has no information at that time whether any downstream node has heard the packet or not.

Similar to [36], nodes in CCACK embed an additional *ACK vector* in the header of each coded data packet of the forward traffic to report a subset of the packets (or coding vectors) they have received (heard) in the past from their upstream nodes. For the following, we use the terms *forward coding vectors* and *ACK coding vectors* to denote the coding coefficients used to encode the payload of the packets and the feedback vectors used to acknowledge the space, respectively. The construction of the ACK coding vector using the vectors stored in $B_u$ is described in Section 4.3.3. Nodes mark forward coding vectors as $\mathsf{H}$, using the inner product of these vectors and the ACK coding vectors they receive from downstream nodes, as we explain in Section 4.3.3.

The destination periodically broadcasts coded feedback to its upstream nodes in the form of ACK vectors (without any payload). This is necessary to inform its upstream nodes whether they should temporarily stop transmitting, since the destination sends no data packets. Once it receives $N$ innovative packets for a batch, it decodes the batch to obtain the $N$ original packets. It then sends an end-to-end ACK back to the source along the shortest ETX path in a reliable manner.[4]

[4]In our current implementation, similar to MORE, the source moves to batch $i + 1$ only when it receives the end-to-end ACK from the destination for batch $i$. As [60] showed, a better approach is

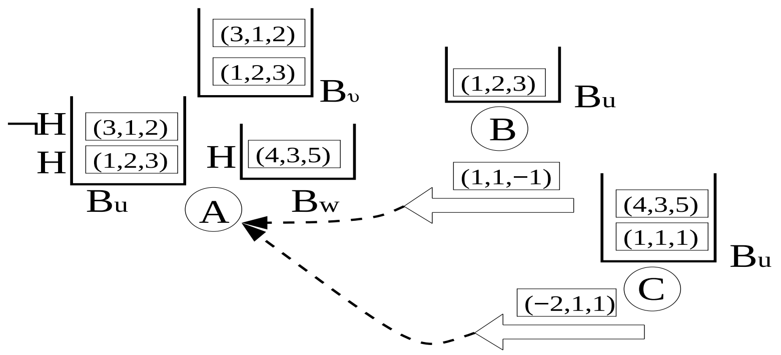


Fig. 4.5. Solving the collective space problem in CCACK .

### 4.3.2 Solving the collective-space problem

In contrast to the simple NSB coded feedback scheme (Section 4.2), nodes in CCACK construct the ACK coding vectors using *all the received forward coding vectors stored in* $B_u$, and not only the innovative vectors stored in $B_v$. Also, when an upstream node $A$ overhears a packet from a downstream node, it uses the ACK coding vector of that packet to decide whether any of the coding vectors in $B_u \cup B_w$, instead of $B_v$, have been heard by the downstream node.

Nodes keep checking the rank of the $B_u$ and $B_w$ vectors marked as $\mathsf{H}$. When this rank becomes equal to the rank of innovative packets in $B_v$ for a node $A$, $A$ stops transmitting either temporarily, until it receives another innovative packet, or permanently if the rank of the $B_v$ vectors is already equal to $N$. In both cases, the downstream nodes have received a sufficient number of packets that cover the innovative packets of $A$ from the knowledge space perspective.

Focusing on $B_u$ and $B_w$ vectors instead of $B_v$, this new structure solves the collective-space problem of the NSB coded feedback. Continue our example from Figure 4.3(b) in Figure 4.5. For node $A$, $B_u$ contains the received coding vectors $(1, 2, 3)$ and $(3, 1, 2)$ while $B_w$ contains the transmitted vector $(4, 3, 5)$. Suppose we reuse the NSB coded feedback for nodes $B$ and $C$. Then by checking inner products

for the source to move to batch $i + 1$ immediately after it stops transmitting packets for batch $i$. In the future we plan to incorporate this feature in CCACK.

with $z_B$ and $z_C$, $A$ knows that the $(1, 2, 3) \in B_u$ and $(4, 3, 5) \in B_w$ have been received. Since the rank of $(1, 2, 3)$ and $(4, 3, 5)$ is the same as the rank of $B_v$ vectors, $A$ stops transmission.

### 4.3.3 Solving the false positive problem

Instead of sending to their upstream nodes a vector randomly chosen from the null space of the vectors in $B_u$, nodes in CCACK use a new algorithm to construct the ACK vectors. With the new algorithm, nodes still append *one* ACK vector to each data packet but construct it in such a way that it is equivalent to appending $M$ vectors from the null space of the vectors in $B_u$. We now describe this new ACK design that drastically reduces the false-positive probability from $\frac{1}{2^8}$ to $\left(\frac{1}{2^8}\right)^M$ for any integer $M \geq 1$.

Each node maintains $M$ different $N \times N$ *hash matrices* $H_1$ to $H_M$ where $N = 32$ is the batch size and each entry of the matrix is randomly chosen from $GF(2^8)$. All nodes in the network are aware of the $H_1$ to $H_M$ matrices of the other nodes. This is achievable by using the ID of a node as a seed to generate the $H_1$ to $H_M$ matrices. We assume that all vectors are row vectors and we use the transpose $u^{\mathrm{T}}$ to represent a column vector (constructed from the row vector $u$).

To improve the efficiency of our feedback mechanism, we associate a *usage_count* with every vector in $B_u$. When a vector is placed in $B_u$, its *usage_count* is set to 0. Every time this vector is selected in the feedback construction algorithm, its *usage_count* is incremented by 1. The ACK vector is always constructed using those vectors in $B_u$ with the lowest counts. This will reduce the probability that the same vectors are repeatedly acknowledged many times.

Nodes construct the ACK vectors using the following algorithm:

```
§ CONSTRUCT THE ACK VECTOR
1: Start from a $0 \times N$ matrix $\Delta$.
2: **while** The number of rows of $\Delta < N - M$ **do**
```

```
3:      Choose the $u$ with the smallest *usage_count* from $B_u$. If more than one such $u$
        exist, choose one randomly.
4:      **for** $j = 1$ to $M$ **do**
5:          **if** the $1 \times N$ row vector $uH_j$ is linearly independent to the row space of $\Delta$
            and the number of rows of $\Delta < N - M$ **then**
6:              Add $uH_j$ to $\Delta$.
7:              Perform row-based Gaussian elimination to keep $\Delta$ in a row-echelon form.[5]
8:          **end if**
9:      **end for**
10:     Increment the *usage_count* of $u$ by 1.
11: **end while**
12: Choose randomly the coding coefficients $c_1$ to $c_N$ such that the following matrix
    equation is satisfied:
        $\Delta(c_1, \cdots, c_N)^{\mathrm{T}} = (0, \cdots, 0)^{\mathrm{T}}.$
    Remark 1: We also require that the randomly chosen coefficients $c_1$ to $c_N$ are not
    all zero.
    Remark 2: By Line 3 there will be at least 1 degree of freedom when solving the
    above equations. Since $\Delta$ is in the row-echelon form, it is easy to choose $c_1$ to $c_N$.
13: Use the vector $(c_1, \cdots, c_N)$ as the ACK vector.
```

When a node A overhears a packet with an ACK vector $z$ from a downstream node, it uses again the inner product to check all its vectors in $B_u$ and $B_w$ and determine whether any of them has been heard by the downstream node. More explicitly, a vector $u \in B_u$ (or $B_w$) is marked $\mathsf{H}$ if and only if $u$ passes *all the following $M$ different "H_tests"* (one for each $H_j$):

$$\forall j = 1, \cdots, M, \quad uH_j z^{\mathrm{T}} = 0, \tag{4.2}$$

where $H_1$ to $H_M$ are the hash matrices of the downstream node of interest.

*Remark:* In our practical implementation, instead of choosing completely random hash matrices $H_1$ to $H_M$ (each with $N^2$ random elements), we simply choose $H_1$ to $H_M$ as "random diagonal matrices", with the $N$ diagonal elements for each $H_j$ randomly chosen from 1 to 255 (excluding zero) and all other elements being zero.

[5]Since $\Delta$ is always of row-echelon form, it is easy to check whether the new vector is linearly independent to the row space of $\Delta$.

This simplification improves the efficiency as the matrix multiplication $uH_j$ can be performed in linear instead of $N^2$ time.

We now quantify the false positive probability (passing all $M$ tests simultaneously) with this new coded feedback scheme.

**Proposition 4.3.1** *Consider an upstream/downstream node pair $A_U$ and $A_D$, and $A_U$ receives an ACK vector $z_0$ from $A_D$. The hash matrices $H_1$ to $H_M$ of node $A_D$ are chosen uniformly randomly. For any $w$ vector in $B_u \cup B_w$ of the upstream node $A_U$, if such $w$ is in the space of the $u$ vectors selected by the downstream node $A_D$, then it is guaranteed that such $w$ vector will pass all $M$ tests in* (4.2)*. If such $w$ is not in the space of the selected $u$ vectors, then the false-positive probability (passing all $M$ tests) is $\left(\frac{1}{2^8}\right)^M$.*

**Proof** Let $S_B$ denote the linear space spanned from the $u$ vectors selected by $A_D$. If $w$ in $B_u \cup B_w$ of $A_U$ is in $S_B$, then $w = \sum_i \alpha_i u_i$ is the linear combination of the selected $u$ vectors (indexed as $u_i$). Since by construction $u_i H_j z_0^{\mathrm{T}} = 0$ for all selected $u_i$, we have $wH_j z_0^{\mathrm{T}} = \sum_i \alpha_i u_i H_j z_0^{\mathrm{T}} = 0$.

Suppose that $w$ in $B_u \cup B_w$ of $A_U$ is not in $S_B$. Conditioning on the non-zero $z_0$ vector, for any $j$ the $H_j z_0^{\mathrm{T}}$ vector must be randomly distributed over the null space of $S_B$, since $u_i H_j z_0^{\mathrm{T}} = 0$ for all selected $u_i$ vectors and since $H_j$ is chosen randomly. Moreover, conditioning on the non-zero $z_0$, $H_1 z_0^{\mathrm{T}}$ to $H_M z_0^{\mathrm{T}}$ are independently distributed over the null space of $S_B$. As a result, even though only a single vector $z_0$ is transmitted, the CCACK scheme has the same effect of using the NSB coded feedback $M$ times, sending out $M$ independently randomly selected vectors ($H_1 z_0^{\mathrm{T}}$ to $H_M z_0^{\mathrm{T}}$) from the null space of $S_B$. By Lemma 1, the overall false-positive event is when all $M$ tests return false positive. The overall false positive probability becomes $\left(\frac{1}{2^8}\right)^M$. The proof is complete. ■

The $M$ value represents a tradeoff between how many vectors one can acknowledge $\frac{32}{M}$ and the false-positive probability $(2^{-8})^M$. Since *any false alarm event for any packet over any link will trigger the land-sliding cost of breaking the communication* *chain*, we observe in our experiments that any $M \leq 3$ will severely jeopardize the reliability of the CCACK. In our implementation we thus choose $M = 4$, which gives a false positive probability of $2.23 \times 10^{-10}$ that is necessary for the effectiveness of CCACK.

It is worth noting that a naive way of avoiding false-positive events is to increase the underlying finite field size $\mathrm{GF}(2^b)$. This is not viable for WMNs. One reason is that to achieve the level of false-positive probability needed in our CCACK scheme ($M = 4$), we need $b = 32$, which uses 4 bytes to represent a single coding symbol. The size of each forward coding vector and each coded feedback vector thus grows from $32 \times 1$ bytes to $32 \times 4$ bytes, which substantially increases the overhead. An even bigger challenge is that each addition and multiplication coding operation now operate on $\mathrm{GF}(2^{32})$. A table look-up method has to have $2^{32} \times 2^{32}$ 4-byte entries, which takes prohibitively 4 million terabytes to store. Since table look-up is impossible, one thus has to use online polynomial-based computation each time a coding operation needs to be performed, which is far beyond today's microprocessor capability.

### 4.3.4 Rate control

The cumulative coded feedback scheme in CCACK helps nodes to determine when they should stop transmitting packets for a given batch, but it does not tell anything about *how fast* nodes should transmit before they stop. Unlike in MORE, in CCACK we cannot use receptions from upstream to trigger new transmissions, since the goal is exactly to stop the upstream nodes, when the downstream nodes have sufficiently enough packets. In addition, we want to apply rate control to the source as well, and not only to the FNs.

The rate control algorithm in CCACK uses a simple credit scheme, which is oblivious to loss rates but aware of the existence of other flows in the neighborhood, and leverages CCACK's cumulative coded acknowledgments.

For each flow $f$ at a node, we define the "differential backlog"[6] as:

$$\Delta Q^f = \dim(B_v^f) - \dim(B_H^f) \tag{4.3}$$

where $B_H^f$ is the set of vectors in $B_u^f \cup B_w^f$ marked as $\mathsf{H}$, and $\dim(S)$ denotes the number of linearly independent packets in the set $S$. Note that $\dim(B_H^f) \leq \dim(B_v^f)$. $\Delta Q^f$ is the difference between the number of innovative packets at a given node and the cumulative number of innovative packets at its downstream FNs for flow $f$. As we saw in Section 4.3.2, when $\Delta Q^f = 0$, i.e., $\dim(B_v^f) = \dim(B_H^f)$, the node stops transmitting packets for flow $f$. Note that for the destination of flow $f$, $\Delta Q^f = 0$.

We also define the relative differential backlog $\Delta Q_{rel}^f$ for each flow $f$ as:

$$\Delta Q_{rel}^f = \frac{\Delta Q^f}{\Delta Q^f + \Delta Q_N} \tag{4.4}$$

where, $\Delta Q_N$ is the total differential backlog of all the neighbor nodes for all flows, calculated as follows. Every time a node $n_j$ broadcasts a coded data packet, it includes in the packet header its current total differential backlog $\Delta Q_{n_j}^{tot}$ of all flows crossing that node. All nodes that hear this packet update their $\Delta Q_N$ as an exponential moving average:

$$\Delta Q_N = 0.5 \times \Delta Q_N + 0.5 \times \Delta Q_{n_j}^{tot} \tag{4.5}$$

Every node in CCACK (including the source and the destination) maintains a credit counter for each flow. Every time there is a transmission opportunity for a node A, one flow $f$ is selected in a round robin fashion, among those flows with $\Delta Q^f > 0$, and the credit counter of that flow is incremented by $\alpha \times \Delta Q_{rel}^f + \beta$. If the counter is positive, the node transmits one coded packet for flow $f$ and decrements the counter by one, otherwise it selects the next flow. The credit increment $\alpha \times \Delta Q_{rel}^f + \beta$

[6]Our solution is inspired by the theoretical backpressure based rate control algorithms [73]. The difference is that, instead of queue lengths, we use innovative coded packets to define a *cumulative differential backlog* for flow $f$ at every node with respect to all its downstream nodes for that flow.

is larger for flows with large “backpressure”, and thus packets of such flows will be transmitted more frequently. For our implementation we selected $\alpha = 5/6$ and $\beta = 1/6$. If $\Delta Q_{rel}^{f} = 1$, then $\alpha \times \Delta Q_{rel}^{f} + \beta = 1$ and the credit counter will always remain equal to 1, effectively allowing the node to always transmit.

## 4.4 Evaluation

### 4.4.1 Methodology

We evaluated the performance of CCACK and compared it against MORE using extensive simulations. We used the Glomosim simulator [61], 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 CCACK. For the implementation of MORE, we followed the details in [22].

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 and combined with the Rayleigh fading model to make the simulations realistic. Because of fading, transmission and sensing range are not fixed but vary significantly around their average values.

We simulated each protocol in 9 different randomly generated topologies, i.e., placement of the 50 nodes. We varied the number of concurrent flows from 1 up to 4. For a given number of flows, we repeated the simulation 10 times for each topology, selecting randomly each time a different set of source-destination pairs, i.e., we had a total of 90 different scenarios for a given number of flows. In each scenario, every source sent a 12MB file, consisting of 1500-byte packets.

Following the methodology in [20, 22], 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. There was no overhead due to loss rate measurements during the file transfer.

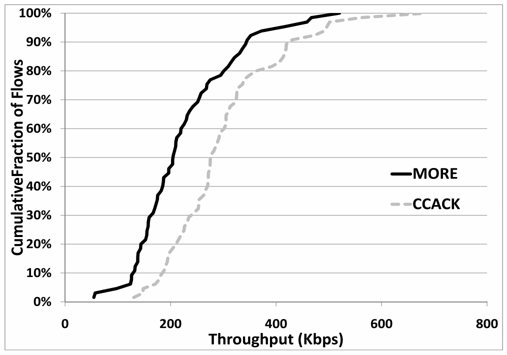
It is generally known that the full benefit of opportunistic routing over traditional routing is exposed when the destination is several hops away from the source [22]; in those cases, opportunistic routing reduces the overhead of retransmissions incurred by high loss rates and increased self-interference. Hence, for the single-flow experiment, among the 90 flows we simulated, we show the results of the 65 flows for which the destination was not within the transmission range of the source (with ETX shortest paths of 3-9 hops). For the evaluation with multiple flows, we kept scenarios with flows of shorter paths, when those flows interfered with other flows. On the other hand, we do not show the results for scenarios where the multiple flows were out of interference range of each other, since those scenarios are equivalent to the single-flow case. We were left with 68 scenarios with 2 flows, and 69 scenarios with 3 and 4 flows.

### 4.4.2 Single flow

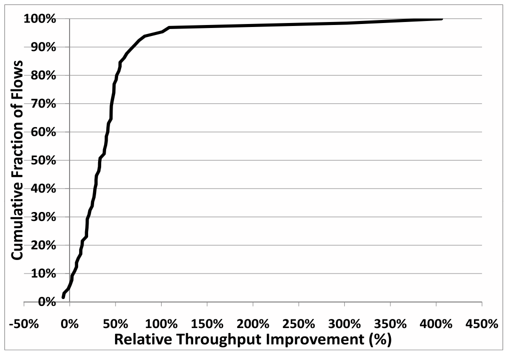
We begin our evaluation with a single flow. Figure 4.6(a) plots the Cumulative Distribution Function (CDF) of the throughputs of the 65 flows with MORE and CCACK. We observe that CCACK outperforms MORE; the median throughput with CCACK and MORE is 276Kbps and 205Kbps, respectively.

Figure 4.6(b) plots the CDF of the relative throughput improvement of CCACK over MORE for all 65 flows, defined as $\frac{T^f_{CCACK}-T^f_{MORE}}{T^f_{MORE}} \times 100\%$, where $T^f_{CCACK}$, $T^f_{MORE}$ are the throughput of flow $f$ with CCACK and MORE, respectively. We observe that CCACK achieves a higher throughput than MORE for 95% of the flows. The median gain of CCACK over MORE is 34%. However, for some challenged flows with the destination 7-9 hops away from the source, the throughput with CCACK is 2-5x higher than with MORE.

**Where does the gain for CCACK come from?** Figure 4.7(a) plots the total number of data transmissions with CCACK and MORE in each of the 65 scenarios, as

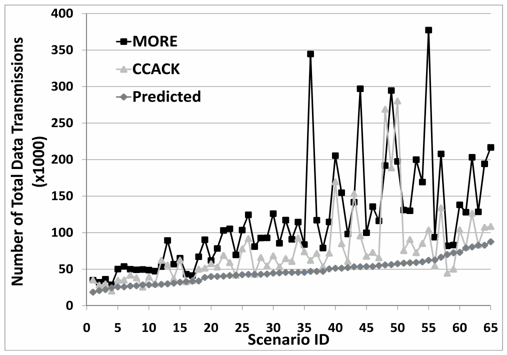


(a) CDF of throughputs achieved with MORE and CCACK.

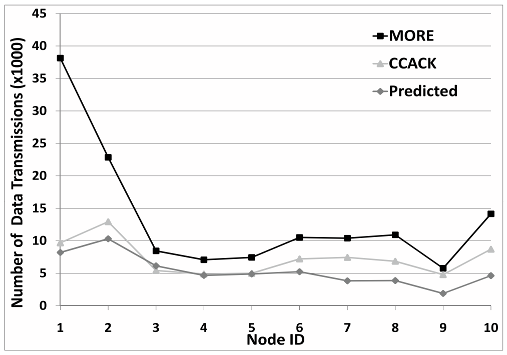


(b) CDF of relative throughput improvement of CCACK over MORE.

Fig. 4.6. Throughput comparison between CCACK and MORE – single flow.



(a) Total number of data transmissions per scenario.



(b) Total number of data transmissions per node for one scenario.

Fig. 4.7. Total number of data transmissions with MORE and CCACK, and predicted number of transmissions, based on MORE's credit calculation algorithm, with a single flow.

well as the predicted number of transmissions in each scenario using MORE's offline ETX-based credit calculation algorithm. The 65 scenarios are sorted with respect to the predicted number of transmissions.

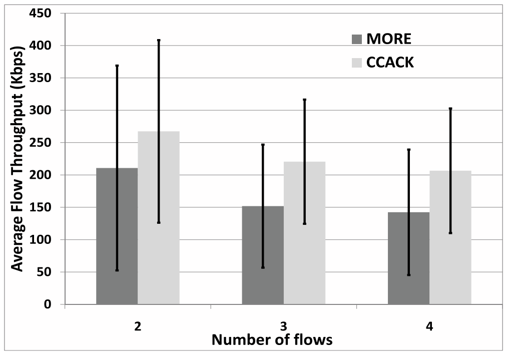
We observe that nodes with MORE perform a higher number of transmissions than the predicted number in all 65 scenarios. The actual number is often more than twice the predicted number, and in some scenarios up to 6-7x the predicted number. This shows that the credit calculation algorithm based on offline ETX measurements mispredicts the required number of transmissions even in the absence of background traffic. The cause is self-interference which changes the loss rates, which in most cases become higher than in a quiet network, where only probing traffic exists [58]. Moreover, the source in MORE keeps transmitting packets until it receives an ACK from the destination. With long paths, this may result in a large number of unnecessary transmissions, as the ACK travels towards the source.

In contrast, the number of data transmissions with CCACK is much lower than with MORE in all but 2 scenarios. In most scenarios it is close to the predicted number, and in some cases, it is even lower. This shows the effectiveness of the coded feedback mechanism in CCACK, combined with the online rate control mechanism of Section 4.3.4.

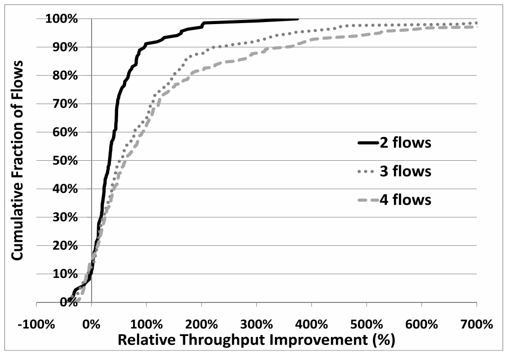
Figure 4.7(b) shows an example (one scenario) of how data transmissions are distributed over the FNs. Nodes are sorted with respect to their ETX distance to the destination, i.e., node 1 is the source and node 10 is the FN closest to the destination. With MORE, the source and the FN closest to the source, perform many more transmissions than the remaining FNs, (2.5-7.6x and 1.4-4.6x, respectively). In contrast, CCACK ensures that these nodes stop transmitting when the remaining downstream FNs have received enough innovative packets. Overall, with CCACK, all 10 nodes perform fewer transmissions than with MORE. The savings range from 17% (for node 9) up to 74% (for the source).

### 4.4.3 Multiple flows

We now evaluate CCACK and MORE with multiple concurrent flows. Here, in addition to throughput, we compare the two protocols in terms of fairness, using



(a) Average per-flow throughputs (bars) and standard deviations (lines).



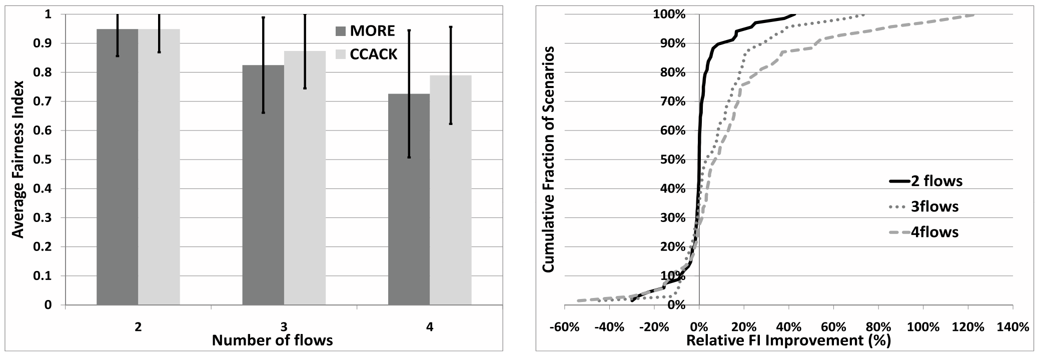
(b) CDF of relative throughput improvement of CCACK over MORE.

Fig. 4.8. Throughput comparison between CCACK and MORE – multiple flows.

*Jain's fairness index* (FI) [74]. Jain's FI is defined as $(\sum x_i)^2/(n \times \sum x_i^2)$, where $x_i$ is the throughput of flow $i$ and $n$ is the total number of flows. The value of Jain's FI is between 0 and 1, with values closer to 1 indicating better fairness.

**Throughput comparison.** Figures 4.8(a), 4.8(b) compare the throughput with CCACK and MORE with 2, 3, and 4 flows. Figure 4.8(a) plots the average per-flow throughput with the two protocols as a function of the number of flows. We observe that CCACK outperforms MORE by 27% on average in the 2-flow case, and by 45% on average in the 3-flow and 4-flow cases. Note that the gain of CCACK is higher with a larger number of flows, when the congestion level becomes higher causing substantial changes to the ETX values. given flow at nodes whose downstream nodes have collectively received a sufficient number of packets, a large amount of bandwidth is saved which can be used by the nodes or their neighbors for transmitting packets for other flows. In contrast, the gain of MORE over traditional routing in [22] drops as the number of concurrent flows increases.

Figure 4.8(b) plots the CDF of per-flow relative throughput improvement with CCACK over MORE, as defined in Section 4.4.2, with 2, 3, and 4 flows. CCACK



(a) Average per scenario FIs (bars) and standard deviations (lines).

(b) CDF of relative FI improvement of CCACK over MORE.

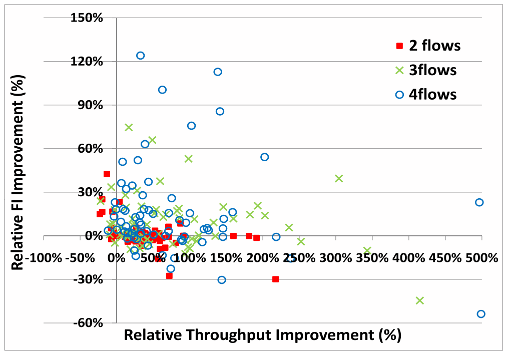
Fig. 4.9. Fairness comparison between CCACK and MORE – multiple flows.

improves per-flow throughputs for more than 85% of the flows in all 3 cases (with 2, 3, and 4 flows). The median improvement is 33%, 55%, and 62%, respectively, with 2, 3, and 4 concurrent flows. Similar to the single flow experiments, some starving flows with MORE show a several-fold improvement with CCACK, up to 4.7x, 9.1x, and 21.4x, in the 2-, 3-, and 4-flow cases, respectively.[7]

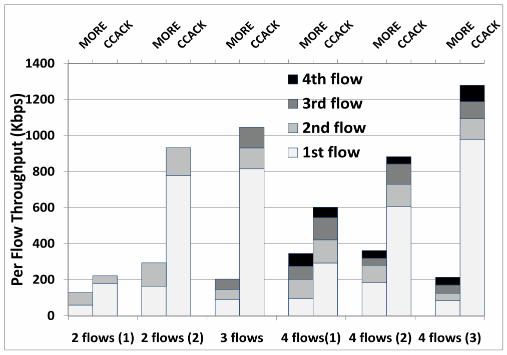
**Fairness comparison.** Figures 4.9(a), 4.9(b) compare the fairness with CCACK and MORE in case of 2, 3, and 4 concurrent flows. Figure 4.9(a) plots the average FI with the two protocols. We observe that the average FI is the same with the two protocols in the 2-flow case, but is higher with CCACK in the 3-flow, and 4-flow case by 5.8% and 8.8%, respectively.

Figure 4.9(b) plots the CDF of per-scenario relative FI improvement with CCACK over MORE, defined similarly to the relative throughput improvement in Section 4.4.2, with 2, 3, and 4 flows. We observe that CCACK improves fairness in more scenarios as the number of flows in the network increases – in 40% of the 2-flow scenarios, 65% of the 3-flow scenarios, and 72% of the 4-flow scenarios. Similar to the throughput

[7]Note that the heavy tails of the 3-flow and 4-flow curves are not shown in Figure 4.8(b) for better clarity.



(a) Scatterplot of relative throughput improvement vs. relative FI improvement with 2, 3, and 4 flows.



(b) Per-flow throughputs with MORE and CCACK for the 6 scenarios with the largest FI decrease under CCACK. The throughputs of different flows in each bar are not additive.

Fig. 4.10. Investigating the relationship between throughput and fairness.

results, the improvement is very large for some scenarios: up to 74% with 3 flows, and up to 124% with 4 flows. This shows again that CCACK improves throughput for some challenged flows, which completely starve with MORE.

**Throughput vs. fairness.** We now investigate more closely the relationship between throughput and fairness. Figure 4.10(a) shows the scatterplots of the relative total throughput improvement per-scenario vs. the relative FI improvement per-scenario, in the 2-, 3-, and 4-flow experiments.

We observe that CCACK improves at least one of the two metrics in all but two scenarios (two points in the 3rd quadrant of Figure 4.10(a)). There are a few points in the 2nd quadrant for all three cases; these are scenarios, where CCACK improves fairness, at the cost of a small total throughput decrease. The majority of the points for the 2-flow case are gathered in the 1st and 4th quadrants, i.e., CCACK either improves throughput at the cost of a (typically) small decrease in fairness, or it improves both metrics. The majority of the points are gathered in the 1st quadrant for the 3-flow and 4-flow cases. This shows that as the number of flows increases, CCACK improves both throughput and fairness in most scenarios.

We now take focus on a few points in the fourth quadrant in Figure 4.10(a), corresponding to scenarios where FI is reduced by more than 20% with CCACK. There are two 2-flow, one 3-flow, and three 4-flow scenarios (points). Note that all 6 points correspond to large throughput improvements, from 72% up to 499%. One may wonder if these improvements are achieved at the cost of compromising the fairness, i.e., the throughput of only one flow increases significantly, causing starvation to the remaining flows.

Figure 4.10(b) shows that this is not the case. This figure plots the individual per-flow throughputs with MORE and CCACK for these 6 scenarios. We observe that CCACK improves throughput of *all* flows involved in all but 2 cases (2nd flow in the *2 flows(1)* scenario and 4th flow in the *4 flows (1)* scenario). The reduction in the FI actually comes from the fact that throughput improvement is much higher for some flows than for some others, and not as a result of starvation of some flows. Take the last scenario (*4 flows (3)*) as an example. CCACK improves throughput of the first flow by 11x (from 85Kbps to 978Kbps), but also improves throughputs of the other 3 flows by 183%, 108%, and 113%.

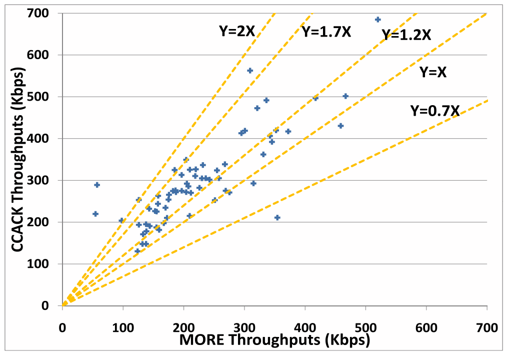
A closer look at the topology of that scenario reveals an interesting situation. The first flow is a 1-hop flow, whose FN belt overlapped with the FN belt of the 4-th, 9-hop flow, near the source of the 4th flow. Apparently, the time required for the destination of the 9-hop flow to decode a batch is much longer than for the destination of the 1-hop flow. During this time, the source of the 9-hop flow keeps transmitting coded packets – remember that MORE applies no rate control and assigns an infinite TX_credit to the source. These transmissions keep triggering new transmissions at the FNs of the 9-hop flow. Now remember that each router in MORE serves flows crossing it in a round robin fashion. Hence, those FNs of the 9-hop flow that also forward packets for the 1-hop flow keep switching between the two flows. In other words, a long flow keeps some routers busy preventing them from serving exclusively the 1-hop flow, although there is no need for these routers to forward packets of the long flow all the time. As a result, the short flow achieves a low throughput of only 85Kbps. In contrast, with CCACK, the coded acknowledgment scheme quickly causes the source and the first FNs of the 9-hop flow to stop transmitting packets, once the remaining FNs collect enough packets. Hence, the FNs near the source are able to forward packets only for the 1-hop flow, increasing its throughput to 978Kbps.

### 4.4.4 Which flows benefit the most from CCACK?

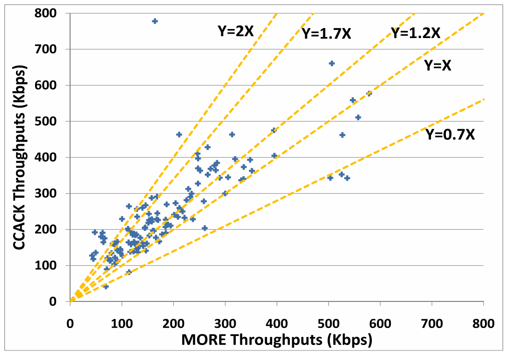
Finally, we examine which flows benefit the most from CCACK in single-flow and multi-flow scenarios. Figures 4.11(a)-4.11(d) show the scatterplots for the individual flow throughputs achieved with MORE and CCACK with 1, 2, 3, and 4 concurrent flows in the network.

**Single flow.** In Figure 4.11(a), we observe that the majority of the points in the single-flow case lie between the lines $Y = 1.2X$ and $Y = 1.7X$, i.e., the throughput gain of CCACK over MORE for a large range of absolute MORE throughput values (100-350Kbps), is typically 20-70%, independent of the absolute throughput value of MORE. In other words, when a single flow is present in the network, CCACK benefits equally both low-throughput and medium-throughput flows. For flows of higher absolute MORE throughput (>350Kbps), the gain of CCACK is smaller; for these flows, the destination is 3-4 hops away from the source and MORE itself can realize most of the gains over traditional routing. On the other hand, for a few flows with very long path lengths (>7hops), the gains are higher than 100% – these are the points on the left of the $Y = 2X$ line in Figure 4.11(a). For these flows, throughput with MORE can be as low as 55Kbps; in contrast, with CCACK there is no flow with throughput lower than 130Kbps.

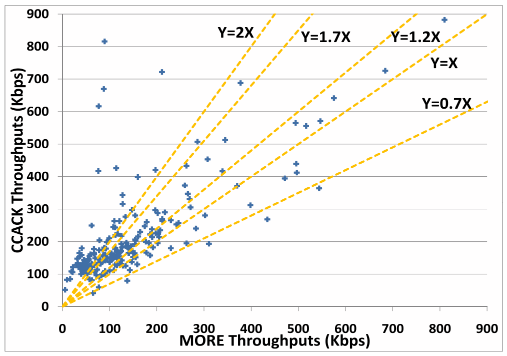
These high-gain points reveal an additional benefit of CCACK. Recall that with both protocols, the destination sends an end-to-end ACK to the source after decoding a batch to trigger the beginning of the next batch. In MORE, this ACK has to



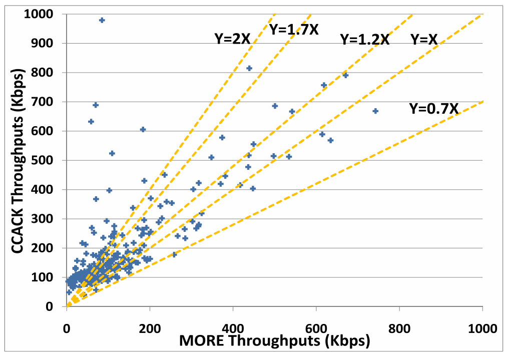
(a) Single flow.



(b) Two flows.



(c) Three flows.



(d) Four flows.

Fig. 4.11. Scatterplots of per-flow throughputs achieved with MORE and CCACK with 1, 2, 3, and 4 concurrent flows.

compete with coded traffic as it travels towards the source, since nodes never stop transmitting. With long paths, it may take a long time for the ACK to reach the source, and this can lead to significant throughput degradation for these flows, as has also been shown in [60]. In contrast, with CCACK, the ACK can quickly travel towards the source without any contention if there is no other flow in the network, since all nodes have already stopped transmitting, thanks to the coded feedback.

**Multiple flows.** In Figures 4.11(b), 4.11(c), and 4.11(d), we observe that, as the number of flows increases, more points are gathered on the left of the $Y = 1.7X$ line;

in the 4-flow case, in Figure 4.11(d), a large fraction of points are gathered on the left of the $y = 2X$ line. Note that the absolute MORE throughputs for many of these points are very low; in particular in the 3-flow and 4-flow cases, throughputs with MORE are as low as only 5Kbps; in contrast, with CCACK, there is only one flow with 41Kbps, and all the remaining flows achieve throughputs higher than 50Kbps. In other words, many flows starve with MORE, as the number of flows in the network increases, and CCACK significantly benefits those flows, with the gains being as high as 21x. In contrast to the single-flow case, these are not necessarily flows with long routing paths, as we saw in the example of Figure 4.10(b).

In contrast to CCACK flows, there is no clear trend for MORE flows of medium to high throughput. The gain of many of those flows remains between 20% and 70%, as in the single-flow case, since there is no room for further room for improvement in a congested network. However, in the 3- and 4-flow scenarios, we also observe gains higher than 70% for some flows of medium MORE throughput (200-400Kbps). On the other hand, for many of those flows, and also for flows of higher MORE throughput, the throughput is slightly reduced with CCACK – the number of points between the lines $Y = X$ and $Y = 0.7X$ increases with the number of flows. This is because these flows typically maintain high throughput with MORE by causing starvation to some other flows. CCACK's rate control algorithm reduces the throughputs of those flows in favor of the most challenged flows, improving overall fairness in the corresponding scenarios.

### 4.4.5 CCACK overhead

Finally, we estimate CCACK overhead compared to MORE. Similar to [22], we discuss three types of overhead: coding, memory, and packet header overhead.

**Coding overhead.** Unavoidably, CCACK's coding overhead is higher than MORE's, since routers have to perform additional operations both when transmitting and when receiving a packet. However, all the additional CCACK operations are performed on

Table 4.1 Coding overhead in CCACK in terms of GF($2^8$) multiplications. Operations marked with (*) are common in MORE and CCACK.

| **Operation** | **Avg.** | **Std. Dev.** |
|---|---|---|
| **Packet Transmission** | | |
| Coded pkt construction (src/FNs)* | 48000/27240 | 0/13128 |
| ACK vector construction | 11584 | 5369 |
| Total (src/FNs) | 59584/38824 | 5369/10021 |
| **Packet Reception** | | |
| Independence check* | 326 | 156 |
| H_tests | 428 | 316 |
| Rank of $\mathsf{H}$ pkts in $B_u \cup B_w$ | 292 | 169 |
| Total | 1046 | 416 |

$N$-byte *vectors* instead of the whole $K$-byte *payload*. Therefore, in practical settings (e.g., with $N = 32$ and $K = 1500$), the coding overhead of CCACK is expected to be comparable to that of MORE.

To verify this, we measured the per-packet cost of the various operations performed upon a packet transmission/reception averaged over all packets transmitted/received at all nodes in the 90 simulation scenarios of Section 4.4.2. Table 4.1 provides the average values and the standard deviations. The costs are given in terms of GF($2^8$) multiplications, which are the most expensive operations involved in coding/decoding [22].

The construction of an ACK vector in CCACK requires on average 11584 multiplications. The total coding cost in transmitting a packet (i.e., constructing a coded packet and an ACK vector) in CCACK is only 24% higher than MORE's, assuming the worst case cost for packet encoding (48000 multiplications). If we use instead the average packet encoding cost at FNs (27240 multiplications), the total cost of transmitting a packet in CCACK is only 38824 multiplications, i.e., lower than MORE's encoding cost at the source.[8]

When receiving a packet, the cost of checking for independence (also in MORE) requires on average only 326 multiplications. The additional operations of performing the H_tests (if the received packet comes from downstream) and maintaining the rank of the H packets in $B_u \cup B_w$ (if a received packet from downstream passes all $M$ H_tests) require on average only 428 and 292 multiplications, respectively, i.e., their costs are comparable to the independence check cost. The total cost of packet reception operations in CCACK is only 1.7% of the total packet transmission cost. Hence, the bottleneck operation in CCACK is preparing a packet for transmission at an FN with 32 innovative packets in $B_v$.

In [22], the authors found that the bottleneck operation in MORE (packet encoding at the source) takes on average $270\mu s$ on a low-end Celeron 800MHz, limiting the maximum achievable throughput with MORE to 44Mbps with a 1500 byte packet. In CCACK, the cost of the bottleneck operation is 24% higher, so we can expect a maximum achievable throughput of 35Mbps. Note that this value is still higher than the effective bitrate of current 802.11a/g WMNs [75].

**Memory overhead.** Same as in MORE, each router in CCACK maintains an innovative packet buffer $B_v$ for each flow, and also a 64KB look up table for reducing the cost of the GF($2^8$) multiplications [22]. With a packet size of 1500 bytes, the size of $B_v$ is 48KB. The extra overhead in CCACK comes from the two additional buffers $B_u$ and $B_w$, which store, however, only 32-byte vectors, and not whole packets. In our implementation, the total size of $B_u$ and $B_v$ is $2 \times 5 \times 32 \times 32 = 10KB$, which is relatively small compared to the size of MORE's structures.

**Header overhead.** The N-byte ACK vector and the total differential backlog $\Delta Q_{n_j}^{tot}$ are the two fields we add to the MORE header. The differential backlog per flow is bounded by the batch size $N$. With $N = 32$, two bytes are enough to support up

[8]Note that the source in CCACK does not have to construct an ACK vector, and hence the cost at the source is the same as in MORE.

to 2048 flows, and the total size of the two fields is equal to 34 bytes. However, in CCACK, we do not include in the packet header the transmission credits for the FNs, which are required in MORE. This can potentially make CCACK's header smaller than MORE's depending on the number of FNs.

## 4.5 Protocol implementation and testbed evaluation

In this section, we describe an implementation of CCACK on the MAP testbed and present experimental results comparing CCACK and MORE.

### 4.5.1 Implementation details

For the implementation of CCACK, we followed the same approach as in Section 3.5.2. For ease of debugging, deployment, and evaluation, we implemented CCACK at the application layer, using broadcast sockets, instead of at layer 2.5. For a fair comparison, we also implemented MORE at the application layer, following all the details in [22]. We note again that such an implementation unavoidably results in some performance degradation for both protocols, compared to an implementation closer to the MAC layer, from crossing the kernel-user boundary. Actually, the degradation is larger for CCACK because its credit mechanism is closely coupled with the MAC layer, as we explain later in this section. Most of the implementation details are the same as in Section 3.5.2. We focus here on one additional challenge we faced in the implementation of CCACK.

**Removing the dependence on the MAC layer.** In an ideal implementation at layer 2.5, a node running either MORE or CCACK transmits a packet when (1) *the 802.11 MAC allows* and (2) *the credit counter is positive.* In our application layer implementation, we cannot get any feedback from the MAC, and hence, we had to modify the transmission policy for the two protocols.

In our implementation of MORE, the application instead 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.[9] Similar to a layer-2.5 implementation, the credit counter is incremented every time a packet is received from an upstream node, and decremented after every transmission.

Unlike in MORE, the credit counter in CCACK is incremented every time the MAC layer signals a transmission opportunity. Since the application cannot know when there is a transmission opportunity without access to the MAC layer, we approximate the number of transmission opportunities via the following heuristic. A node increments its credit counter every time it hears a data packet transmission from another node by a fraction of $1/N$ of the actual increment determined by the rate control algorithm, where $N$ is the number of nodes in the node's neighborhood. The intuition behind this is that with a fair MAC layer every node in a neighborhood would roughly get an equal number of transmission opportunities. To avoid possible deadlock situations, where every node in a neighborhood is waiting for another node to transmit, we also use a timeout equal to one data packet transmission time, after which a node always increments its credit counter.

### 4.5.2 Experimental setup

In the implementation of the two protocols, we used the same parameters as in our simulation study in Section 4.4. 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 1-5 hops in length, and the loss rates of the links vary from 0% to 91%, with an average value of 36%.

We experimented with 20 single-flow scenarios (i.e., randomly selected source-destination pairs), 10 2-flow scenarios, and 6 3-flow scenarios. For each scenario, we first ran the ETX module to collect the pairwise loss rates and ETX metric for

[9]We try to keep this time as short as possible by limiting the socket buffer size and the card's queue length to a couple of packets, as we have described in Section 3.5.2.

each link of our testbed, and then we ran the two protocols, MORE and CCACK, in sequence. With both protocols, the source sent a 2.3MB file consisting of 1460-byte packets.

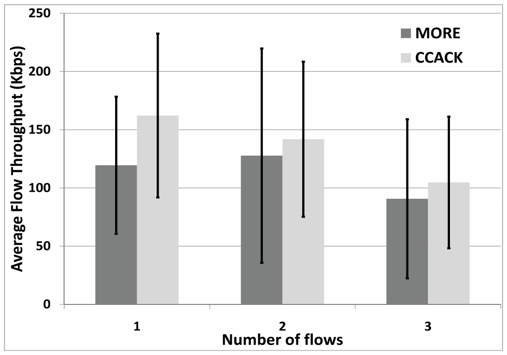
As we have explained in Section 4.4.1, the gain of CCACK over MORE is more pronounced with flows over long paths, where the destination is several hops away from the source. Unfortunately, the size of our testbed limited our choices in flow selection. Hence, in the single-flow experiments described below, we also included flows where the destination was 2 hops away from the source (unlike in Section 4.4.2, where the minimum source-destination distance was 3 hops). Similarly, the small size of the testbed resulted in a large fraction of the nodes being within sensing range of each other; this prevented us from increasing the total number of flows beyond three, since the medium became congested, resulting in very poor performance for both protocols.[10] These two limitations, along with the implementation limitations we discussed in Section 4.5.1, are expected to limit the gains of CCACK over MORE, compared to the simulations results in Section 4.4.

### 4.5.3 Experimental results

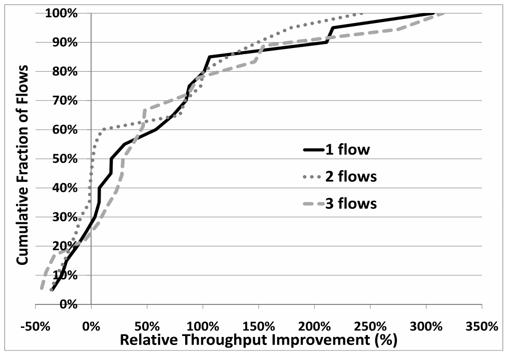
The testbed evaluation results are shown in Figures 4.12(a)-4.12(d).

Figures 4.12(a), 4.12(b) compare throughput with CCACK and MORE with 1, 2, and 3 flows. In Figure 4.12(a), we observe that CCACK outperforms MORE by 36% on average in the 1-flow scenarios, by 11% in the 2-flow scenarios, and by 15% on average in the 3-flow scenarios. Figure 4.12(b) plots the CDF of per-flow relative throughput improvement with CCACK over MORE, as defined in Section 4.4.2, with 1, 2, and 3 flows. CCACK improves per-flow throughputs for 72% of the flows in the 1-flow scenarios, 55% of the flows in the 2-flow scenarios, and 75% of the flows in the 3-flow scenarios. The median improvements are 18%, 3%, and 28%, re-

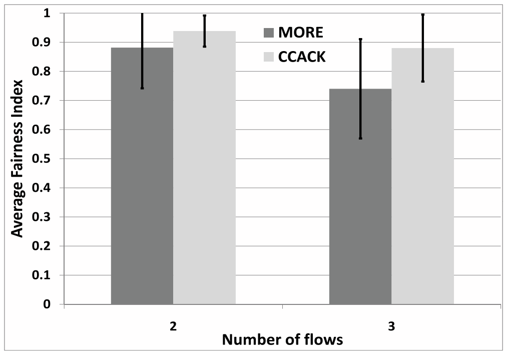
[10]As explained in [22], intra-flow network coding based protocols cannot increase the capacity of the network and they can only improve throughput as long as the total load remains below the network capacity.



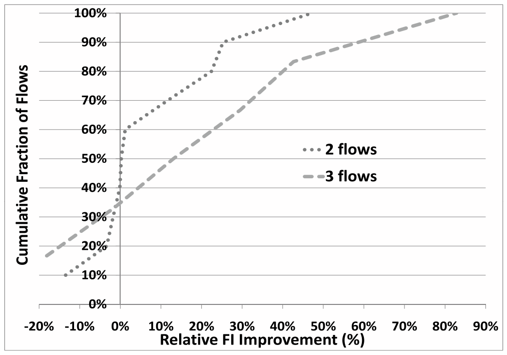
(a) Average per-flow throughputs (bars) and standard deviations (lines).



(b) CDF of relative throughput improvement of CCACK over MORE.



(c) Average per scenario FIs (bars) and standard deviations (lines).



(d) CDF of relative FI improvement of CCACK over MORE

Fig. 4.12. Testbed evaluation results.

spectively, in the 1-, 2-, and 3-flow scenarios. These gains are lower than the ones observed in the simulation results in Section 4.4, due to the limitations we discussed in Section 4.5.2. In spite of these limitations though, our results still demonstrate the benefit of CCACK over MORE in the case of challenged flows. We observe that about 20% of the flows in 1-flow and 2-flow scenarios, and 17% of the flows in the 3-flow scenarios show a several-fold throughput improvement with CCACK, up to 3x, 2.4x, and 3.2x, respectively.

Figures 4.12(c), 4.12(d) compare fairness with CCACK and MORE in case of 2, and 3 concurrent flows. Figure 4.12(c) plots the average FI with the two protocols. We observe that the average FI is higher with CCACK in both the 2-flow, and 3-flow case by 5.7% and 18.9%, respectively. These values are actually higher than the simulation results. Due to the small size of the testbed, the network gets more easily congested, even with 2 flows and CCACK's backpressure-inspired credit mechanism is very effective in allocating the medium's bandwidth fairly among contending flows. Figure 4.12(d) plots the CDF of per-scenario relative FI improvement with CCACK over MORE. CCACK improves fairness in 60% of the 2-flow scenarios, and 65% of the 3-flow scenarios and the gains can be as high as 83% in some challenged scenarios.

## 4.6 Summary

The use of random linear network coding has significantly simplified the design of opportunistic routing (OR) protocols by removing the need of coordination among forwarding nodes for avoiding duplicate transmissions. However, network coding based opportunistic routing protocols face a new challenge: *How many coded packets should each forwarder transmit?* To avoid the overhead of feedback exchange, most practical existing network coding based opportunistic routing protocols compute offline the expected number of transmissions for each forwarder using heuristics based on periodic measurements of the average link loss rates and the ETX metric. Although attractive due to their minimal coordination overhead, these approaches often suffer significant performance degradation in dynamic wireless environments with continuously changing levels of channel gains, interference, and background traffic.

In this chapter, we presented a novel approach to network coding based opportunistic routing through the design of CCACK, a new efficient network coding based opportunistic routing protocol. Instead of avoiding feedback exchange, CCACK encodes feedback messages in addition to encoding data packets. A novel Cumulative Coded Acknowledgment scheme allows nodes in CCACK to acknowledge network coded traffic to their upstream nodes in a simple and efficient way, oblivious to loss rates, and with practically zero overhead. The cumulative coded acknowledgment scheme in CCACK also enables an efficient credit-based, rate control algorithm. Our experiments on a 22-node 802.11 WMN testbed show that compared to MORE, a state-of-the-art network coding based opportunistic routing protocol, CCACK improves both throughput and fairness, by up to 3.2x and 83%, respectively, with average improvements of 11-36% and 5.7-8.3%, respectively, for different numbers of concurrent flows. Our extensive simulations show that the gains are much higher in large networks, with longer routing paths between sources and destinations.

# 5. HOW TO EVALUATE “EXOTIC” WIRELESS ROUTING PROTOCOLS?

One common characteristic of the “exotic” optimization techniques such as opportunistic routing and network coding, is that they have moved many mechanisms such as reliability and rate control, that used to be below or above the routing layer in traditional protocols, to the routing layer. In this chapter, we show that the consolidation of mechanisms from multiple layers into the routing layer poses new challenges to the methodology for evaluating and comparing this new generation of routing protocols. We then discuss the diverse set of current practices in evaluating recently proposed protocols and their strengths and weaknesses. Our discussion suggests that there is an urgent need to carefully rethink the implications of the new merged-layer routing protocol design and develop effective methodologies for meaningful and fair comparison of these protocols. Finally, we make several concrete suggestions on the desired evaluation methodology. In particular, we show that the traffic sending rate plays a fundamental role and should be carefully controlled.

## 5.1 Renaissance of wireless routing protocol design

In this section, we give a brief overview of the evolution of routing protocol design for multihop wireless networks. Different from Section 1.1, where our goal was to give a brief background on the two techniques upon which we built our two proposed protocols, *Pacifier* and CCACK, here we give a more comprehensive picture by covering all the “exotic” techniques (opportunistic routing, intra-flow network coding, and inter-flow network coding), the two traditional techniques (reliability and rate control) that were moved to the routing layer, and more protocols, aiming to emphasize two main points: First, the shift in the main goals of routing protocol design as we passed from the mobile ad hoc network (MANET) era to the wireless mesh network (WMN) era, and second, how this shift affected the evaluation methodology of routing protocols.

The recent evolution of wireless networking from the ad hoc networking era to the mesh networking era has ignited a new Renaissance of routing protocol design for multihop wireless networks.

In the ad hoc networking era, the primary challenge faced by routing protocols (e.g., DSR [14], AODV [15]) was to deal with frequent route breaks due to host mobility in a dynamic mobile environment. Accordingly, most research efforts were focused on designing efficient route discovery/repair schemes to discover or repair routes with minimum overhead. These protocols relied on 802.11 unicast (with its built-in ACK-based local recovery scheme and exponential backoff) to deal with packet loss due to channel errors or collisions.

The design goals of the ad hoc routing protocols also drove their evaluation methodology. The comparison between different protocols was usually in terms of Packet Delivery Ratio (PDR) and control overhead (e.g. [76, 77]). The offered load, typically of some constant rate, was low so that the resulting data traffic and control overhead do not exceed the network capacity. The main parameter varied in the evaluations was the *pause time* of the random waypoint mobility model, which characterized how dynamic the environment was. The focus of such a methodology was to offer a direct comparison of various protocols' ability to transfer data to the destination under host mobility, while incurring low control overhead. Interestingly, often times the protocol comparisons boiled down to tradeoffs between PDR and control overhead [76, 77].

Transition to WMNs changed these rules. In a WMN, routers are static and hence route changes due to mobility are not a concern anymore. The main performance metric is now *throughput*, often times even at the cost of increased control overhead.

The first major effort towards the new design goal was on designing link-quality path metrics (e.g., ETX [18], ETT [19]) that replaced the commonly used shortest-path metric. The protocols using these link-quality metrics still followed the layering principle: the routing layer finds a good route, and 802.11 unicast is used to deliver packets hop by hop.

**Opportunistic routing.** Seeking further throughput improvement, researchers looked into new, "exotic" techniques, which largely abandoned the layering principle. The first such technique was opportunistic routing. Instead of having a decoupled MAC and routing layer, ExOR explored an inherent property of the wireless medium, its broadcast nature. Instead of first determining the next hop and then sending the packet to it, it broadcasts the packet so that all neighbors have the chance to hear it; among those that received the packet, the node closest to the destination forwards the packet. This also implies that some coordination is required, so that the neighboring nodes can agree on who should rebroadcast the packet next. To reduce the coordination overhead, ExOR proposed sending packets in batches.

**Intra-flow network coding.** The second "exotic" technique applied network coding to multihop wireless networks. With network coding, each mesh router randomly mixes packets it has received before forwarding them. The random mixing ensures with high probability that nodes will not forward the same packet, and hence coordination overhead is minimized. Network coding has one more positive effect. It resembles traditional Forward Error Correction (FEC) techniques, which offer reliability through redundancy, with the extra advantage that it is applied at every hop, and not end-to-end [23, 24]. Together, network coding eliminates the need for reliability on a per-hop or per-packet basis. Since each coded packet contains information about many packets, the destination can reconstruct the original data if it receives sufficiently many packets.

Both techniques use unreliable 802.11 broadcast as the hop-by-hop forwarding technique, which is a significant departure from traditional routing protocols. The use of broadcast is a necessity for opportunistic routing as well as effective network coding. With 802.11 broadcast, there is no reliability mechanism and no exponential backoff to protect from packet loss due to either channel errors or collisions. However, it is not required either. The beauty of opportunistic routing is that we do not tie each transmission to any particular next hop; if one node does not receive a packet some other node will probably receive it. In addition, with network coding, we can send infinite amount of redundancy until the destination can reconstruct the original packets.

Since the MAC now does not have to deal with retransmissions and exponential backoffs, it can send at much higher packet rates than in the unicast mode; it is essentially limited only by carrier sensing. Sending at higher rates potentially implies higher goodput. Since the design goal is focused on high throughput, this observation has an immediate implication for the evaluation methodology of these new protocols: instead of using a constant rate (CBR) of $X$ packets per second, *the source node should send as fast as the MAC allows.*

However, making the sources send as fast as the MAC allows has a serious side effect. It can cause congestion in the network if the aggregate transmission rate of the nodes exceeds the network capacity. As [58] showed, in contrast to the wired Internet, where congestion is the result of a complex interaction among many flows, in a wireless network, congestion can happen even with a single flow, in a simple topology and even with 802.11 unicast. The use of broadcast in this new generation of routing schemes simply worsens the situation, since the lack of exponential backoff in the 802.11 broadcast mode means nodes never really slow down.

**Rate control.** With congestion, the queues of the nodes become full, causing significant packet loss. We thus need to reintroduce the mechanism for preventing the network from reaching this state: rate control. SOAR [55] is a new opportunistic routing protocol that has a built-in rate control mechanism, both at the source (using a sliding window) and at intermediate routers (using small queues to avoid unpredictable queuing delays). Other protocols (e.g., [56, 78]) propose hop-by-hop, backpressure-based mechanisms to limit the amount of traffic injected in the network. Hence, *rate control, which used to be the responsibility of higher layer protocols (transport or application), is now brought down to the routing layer.*

**Inter-flow network coding.** The final frontier is that of *increasing the network capacity* itself! The basic idea is again simple: a router can XOR packets from different flows (hence inter-flow network coding as opposed to intra-flow network coding discussed previously) and broadcast them. If the next hop of each flow has already overheard all the mixed packets except for the one destined for it, it can XOR them again with the XORed packet to obtain its own packet. COPE [79] was the first protocol that brought this idea from theory into practice. By mixing packets belonging to different flows and transmitting them as one, one reduces the total number of transmissions required, and hence increases the "effective" capacity of the network.

Since the technique stretches the capacity of the network, the most natural way to show its improvement, i.e., the implied evaluation methodology, is to subject the network to a traffic load (not too much) above the physical capacity, i.e., the network should already be congested before network coding is turned on, which will then increase the effective capacity just enough to eliminate the congestion. Viewed differently, only in a congested network there will be many opportunities for packet mixing at the intermediate routers, and coding will give a significant throughput improvement compared to no coding.

**Reliability.** Since 802.11 broadcast is unreliable, with the exception of intra-flow network coding, which embraces FEC, all other techniques, which rely on MAC-layer broadcast, require some ARQ-based recovery mechanism. ExOR uses end-to-end retransmissions by going through the same batch of packets until 90% of them are received by the destination; SOAR and COPE use asynchronous cumulative hop-by-hop acknowledgments; COPE also relies partly on 802.11 unicast (known as pseudobroadcast [79]). Hence, in addition to rate control, one more mechanism, *reliability, which used to be the responsibility of either upper (end-to-end) or lower (hop-by-hop) layers, is now brought to the routing layer.*

In summary, the "exotic" techniques used in new routing protocols for WMNs have largely abandoned the layering principle and adopted a merged-layer approach,

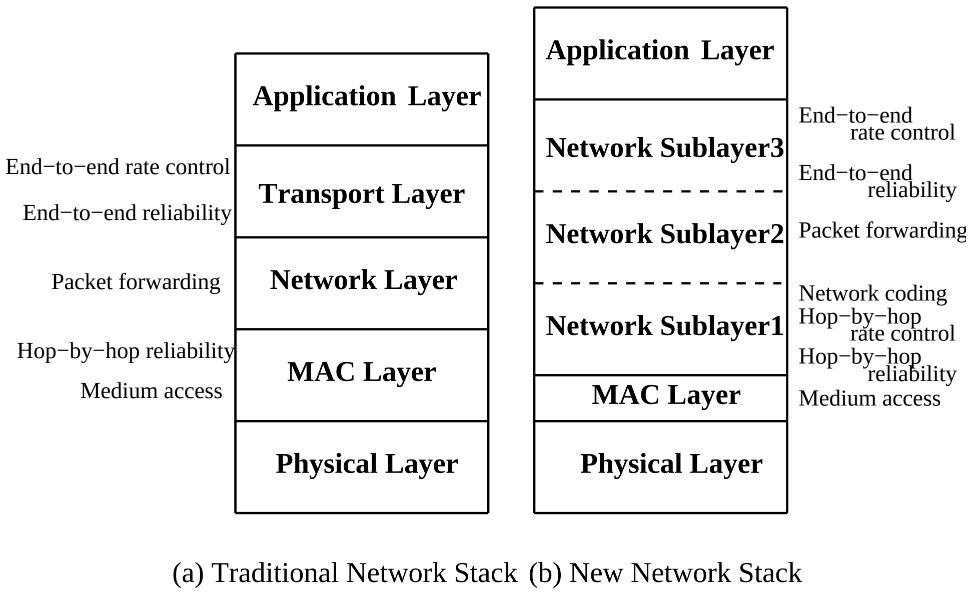


Fig. 5.1. The evolution of the protocol stack. Part of the MAC layer functionality is moved to the routing layer (network sublayer 1). Transport layer is also blurred into the routing layer (network sublayer 3)

as shown in Figure 5.1. Mechanisms that used to be at lower or higher layers are now blended into the routing layer. *This consolidation of mechanisms and techniques into the routing layer has made the evaluation of routing protocol performance a much subtler task than before.* For example, some mechanisms and techniques may be conflicting: inter-flow network coding desires traffic load to be above the network capacity while rate control targets the exact opposite.

In the next section, we discuss the diverse set of current practices in evaluating this new generation of routing protocols. We show that, in contrast to traditional routing protocols, there have been no clear guidelines that drive the evaluation of these protocols; often times each new protocol is evaluated with a different methodology.

## 5.2 State of affairs

There have been many high-throughput routing protocols for WMNs proposed over the last few years. We review here the evaluation methodologies used in a rep-

Table 5.1 Methodologies used in evaluating recent high-throughput WMN routing protocols.

| **Protocols** | **Evaluation Methodology** | **Example** |
|---|---|---|
| **Unreliable** | Make both protocols reliable but in different ways | ExOR [20] |
| | Evaluate for a wide range of sending rates, with deteriorating PDR | COPE [79] |
| | Compare a protocol with rate control against a protocol without rate control | SOAR [55] |
| | Old ad hoc methodology: keep the sending rate fixed below capacity, measure PDR | ROMER [80] |
| **Reliable** | Compare a reliable protocol against an unreliable protocol | MORE [22] |
| | Compare a reliable protocol against an unreliable protocol under TCP | noCoCo [78] |
| | Modify an unreliable protocol to incorporate the same reliability mechanism of a new protocol | noCoCo [78] |

resentative subset of them, as summarized in Table 5.1. We separate two classes of protocols: unreliable protocols, in Section 5.2.1 and reliable protocols, in Section 5.2.2. A protocol is defined as reliable if it guarantees 100% PDR to the destination through local or end-to-end recovery mechanisms (retransmissions or redundancy); otherwise, it is defined as unreliable. Finally, we briefly discuss an orthogonal issue in the evaluation of both classes of protocols, the use of MAC autorate adaptation, in Section 5.2.3.

### 5.2.1 Evaluation of unreliable protocols

In the case of unreliable protocols, the main objective is high throughput perceived by the destinations, i.e., high goodput. The new trend in the evaluation methodology is to saturate the network, letting the sources send as fast as possible so that the traffic load in the network exceeds the available capacity; then measure the maximum amount of traffic the protocol can deliver to the destination.

However, such a methodology is flawed in that it completely deemphasizes the PDR metric. The fact that certain applications do not require 100% PDR does not mean that reliability is a factor that can be completely neglected. Many applications have certain lower bounds for reliability; for example the quality of a video deteriorates with packet loss, and hence if the PDR drops below a threshold, the video quality becomes unacceptable.

**Practice 1: Making both protocols reliable.** ExOR guarantees reliable end-to-end delivery of 90% of each batch; every node keeps retransmitting packets belonging to a given batch until they are acknowledged by a node closer to the destination. The last 10% of the packets could incur a lot of overhead if they were sent through ExOR, and hence they are sent through traditional routing, which does not offer any guarantee for end-to-end reliability.

The authors argued that a direct comparison of ExOR with traditional routing would be unfair and they conducted the experiments in a way that guaranteed 100% PDR with both of them. In each case, the size of the file to be downloaded was 1MB. Instead of using traditional routing to carry the last 10% of the file, the evaluation of ExOR was based on the transmission of a 1.1 MB file, so as to compensate for loss. In contrast, the traditional routing protocol was only used to determine the route offline. The 1MB file was then transferred sequentially hop-by-hop, thus eliminating collisions, and also packet drops due to queue overflows.[1]

While this methodology was largely fair, it eliminated one important feature of traditional routing that does not exist in ExOR: spatial reuse. To avoid duplicate transmissions, nodes in ExOR are assigned priorities, and only one node transmits at a time – hence, coordination is achieved at the cost of reduced spatial reuse. In contrast, with traditional routing simultaneous transmissions can take place across

[1]The packet losses due to channel errors were masked in the testbed through 802.11 retransmissions.

the network as long as they do not interfere with each other. This advantage can turn into a drawback in the presence of a large number of hidden terminals. In other words, by trying to make the comparison fair by adding reliability to traditional routing, the authors also removed one feature of traditional routing. Whether this feature harmed traditional routing depends on the particular environment used for the evaluation.

**Practice 2: No rate control - varying the sending rate.** COPE in [79] was compared against a traditional routing protocol (Srcr), under UDP traffic.[2] In an 802.11a network with a nominal bitrate of 6Mbps, the experiment was repeated for gradually increased total offered load. The aggregate throughput over the total offered load for the two protocols was then presented, as shown in Figure 5.2 (Figure 12 in [79]).

We make several observations on Figure 5.2. First, the advantage of COPE is best shown when the traffic load in the network is pushed beyond the capacity. Since it is not clear what the traffic load is, the best thing is to measure throughput for varying offered load, as done by the authors. As expected, at low loads, COPE performs similarly to traditional routing. As the load increases, COPE offers on *average* 3-4$x$ throughput improvement over traditional routing. Second, like traditional routing, the goodput of COPE also peaks when the offered load is around the effective capacity of the network (now higher because of inter-flow network coding), and decreases quickly as the load further increases, and the PDR value, which can be easily calculated by dividing the y value by the x value, deteriorates sharply, possibly below the acceptable level of many applications. Third, if the protocols have rate control mechanisms, ideally the goodput should remain constant when the offered load is increased to beyond the network capacity. Since neither protocol has rate control, we witness the decline of the goodput.

**Practice 3: Comparing a protocol with rate control against a protocol without.** SOAR applies sliding window-based rate control at the sources, trying

[2] [79] also evaluated COPE and Srcr under TCP. In that case, although the two protocols are unreliable, reliability is provided by the transport layer.

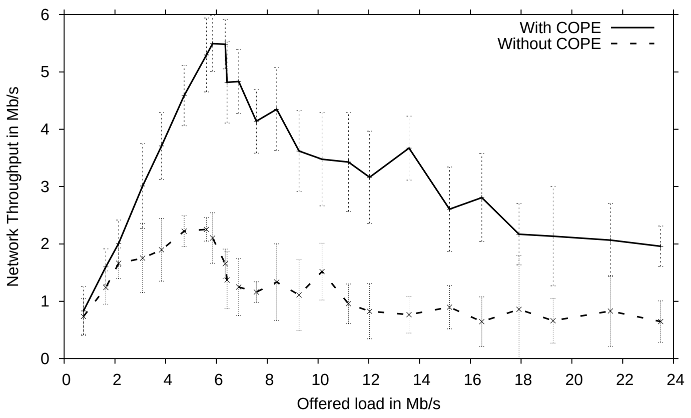


Fig. 5.2. Evaluation of COPE and traditional routing in an ad hoc network for UDP flows. Reproduced Figure 12 from [79].

to discover the optimal sending rate online. In contrast, traditional routing has no rate control. This immediately creates a challenge for a fair comparison of the two protocols. Faced with this challenge, the authors decided to perform the evaluation in a saturated network, where each source transmits at 6Mbps, same as the nominal bitrate of the network.

Saturating the network creates an adverse situation for traditional routing, which is expected to perform poorly under these conditions and suffer significant packet loss due to queue overflows. In contrast, SOAR adapts the sending rate online, based on the network conditions. SOAR was shown to offer a large throughput improvement over traditional routing (one example is shown in Figure 5.3, Figure 14(b) in [55]). However, it is not clear what part of the improvement is because of the opportunistic forwarding and what part is because of the rate control.

**Practice 4: Old methodology (for evaluating ad hoc protocols).** ROMER [80] is another opportunistic routing protocol which exploits link bitrate diversity in order to maximize throughput. It uses the 802.11 unicast autorate adaptation mechanism, and tries to send traffic over high rate links, in contrast to ExOR and SOAR, which always use a fixed link bitrate. ROMER was evaluated under yet another methodology different from ExOR and SOAR. The authors compared the PDR (and throughput

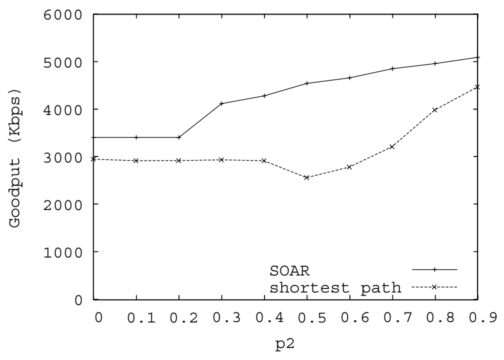


Fig. 5.3. Evaluation of SOAR and traditional routing: 4 parallel flows in a grid topology. Reproduced Figure 14(b) from [55].

gain) achieved by ROMER over traditional routing, following the old methodology for ad hoc protocol evaluation. The parameter varied is the link failure probability, while the source sending rate is kept constant (and the value is unclear).

Due to the autorate adaptation, it is difficult to estimate the capacity of the network used for the evaluation. The high delivery rates achieved (at least) by ROMER (in Figure 5.4, Figure 5 in [80]) make us conjecture that the sending rate was not high enough to congest the network, in contrast to in [55] and [79]. However, a single sending rate does not reveal the maximum gain achieved by ROMER, in particular if this rate is far below the capacity of the network.

### 5.2.2 Evaluation of reliable protocols

Traditional routing protocols left to the transport layer the responsibility for end-to-end reliability. However, TCP, the de facto reliable transport layer protocol for the wired Internet, has been reported to perform poorly in multihop wireless networks [81–90], especially in environments with many hidden terminals and highly lossy links. The reason is that TCP performs congestion control in addition to reliability and correlates these two mechanisms. High packet loss causes TCP flows to suffer timeouts and excessive backoff, and it prevents them from increasing their

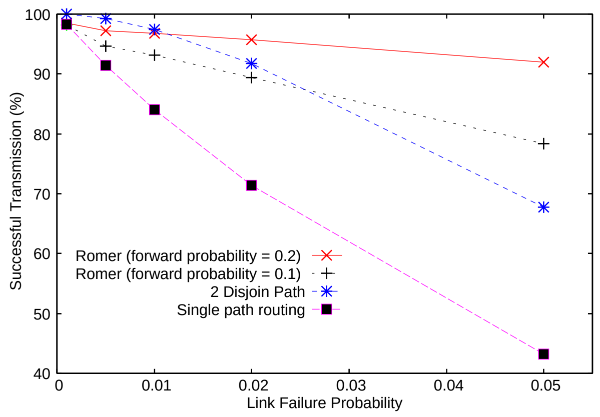


Fig. 5.4. Evaluation of ROMER and traditional routing: PDR over link failure probability. Reproduced Figure 5 from [80].

window size and utilizing the wireless medium efficiently. This is the reason many new protocols ignore TCP, and incorporate mechanisms for end-to-end reliability at the network layer instead.

**Practice 5: Comparing a reliable with an unreliable protocol.** In [22], MORE is compared against traditional routing showing a median throughput gain of 95%. The authors used UDP traffic for both protocols sent at the maximum possible data rate, i.e., the source transmitted as fast as the MAC allowed. As we have already explained, in a highly congested environment, 802.11 unicast cannot help traditional routing to recover from packet drops due to queue overflows. In contrast, with MORE there is no queuing. With a batch size of $k$ packets, every MORE router only needs to keep $k$ linearly independent packets in a buffer; linearly dependent packets do not include any new information and can be safely dropped. Hence, a MORE router does not experience losses due to queue overflows, no matter how fast it receives packets from its upstream nodes. In addition, the FEC element contained in network coding masks packet losses due to collisions and channel errors through redundancy. Thus, a reliable protocol was compared against an unreliable one.

This does not necessarily mean that the comparison favored MORE over traditional routing. In the evaluation of the two protocols, a fixed size file was sent from the source to the destination with each protocol, however with traditional routing only a fraction of this file is finally delivered to the destination. Depending on the fraction of the file that is lost and the time taken for the transfer, this evaluation could favor any of the two protocols. In other words, adding an end-to-end reliability mechanism to traditional routing would increase the numerator of the throughput formula (the amount of data delivered) but it would also increase the denominator (the time taken for the total transfer); this could lead to either an increase or a decrease to the throughput achieved with traditional routing.

**Practice 6: Running an unreliable protocol under TCP.** An easy way to provide end-to-end reliability with an unreliable routing protocol is to run it under TCP; no change is required to the protocol itself. This is one of the approaches followed by [78] in the evaluation of noCoCo. noCoCo improves COPE by scheduling the transmissions at the nodes in order to maximize the gain from inter-flow network coding. Coupled with scheduling in noCoCo is a backpressure, hop-by-hop congestion control mechanism. The main idea is that a node does not transmit a new packet before the next hop implicitly or explicitly acknowledges the reception of the previous transmission. This mechanism eliminates queue overflows and packet dropping and guarantees end-to-end reliable packet delivery. Hence, in noCoCo, sources do not transmit as fast as the MAC allows; their sending rates are limited by the congestion control mechanism.

In the evaluation, noCoCo was compared against COPE [79] and traditional routing. The main goal was to quantify the gains of coordinated network coding used in noCoCo against opportunistic network coding, used in COPE. TCP was used with COPE and traditional routing to provide reliability (and congestion control) at the transport layer. However, TCP is known to perform poorly in multihop wireless networks; in addition, it was shown to interact poorly with COPE and limit the coding opportunities and consequently the throughput gain [79]. Hence, this methodology again blurred the true gain from coordinated coding, since different congestion control and reliability mechanisms are used. The authors acknowledged this point and noted that it should be taken into account when trying to interpret the results.

**Practice 7: Modifying an unreliable protocol.** To finally isolate the gain from coordinated coding, the authors of noCoCo also modified traditional routing and COPE to use the same backpressure-based algorithm for congestion control and reliability, thus removing the negative side-effects of TCP.

### 5.2.3 Use (or no use) of autorate adaptation

802.11 unicast allows a sender to change the bit rate automatically, based on the quality of the link to the receiver. On the other hand, the majority of the "exotic" optimization techniques are based on 802.11 broadcast, and hence most of the new routing protocols based on these techniques (with the exception of ROMER) do not use autorate adaptation. For "fair" comparison, the evaluation of these protocols often disables autorate adaptation for the traditional, unicast routing, e.g., in [20, 55, 78, 79] (one notable exception is [22]). We argue the contrary; the methodology is unfair to traditional routing if it can benefit from autorate adaptation.

## 5.3 Recommendations

We have witnessed the inconsistencies in the current evaluation methodologies of the new generation of routing protocols. In the following, we make recommendations for more consistent and meaningful evaluation methodologies.

**The importance of rate control.** Rate control is fundamental for the optimal operation of any (unreliable or reliable) protocol, as it ensures that the traffic load does not exceed the network capacity limit.

Figure 5.5 shows our envisioned throughput performance for well designed unreliable protocols. Traditional routing under UDP has no rate control mechanism incorporated. When the offered load exceeds the network capacity, packets start get-

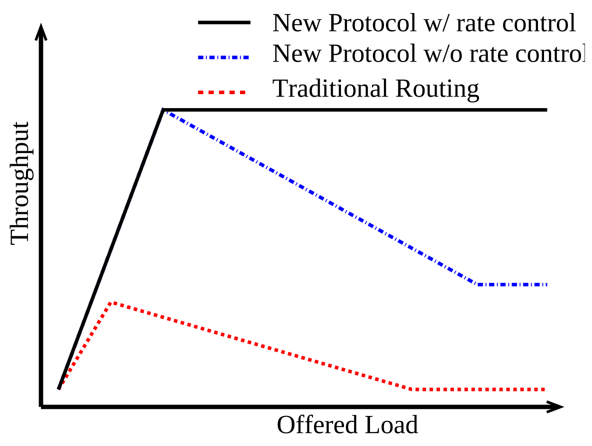


Fig. 5.5. Envisioned throughput performance for well designed unreliable protocols (with built-in rate control), in contrast to traditional routing and high-throughput protocols without rate control.

ting dropped due to congestion, possibly reducing the throughput much below its possible maximum value. New protocols with “exotic” techniques are expected to offer a dramatic increase to the throughput; they can even increase the capacity bound (e.g., from inter-flow network coding). However, without rate control, congestion can build up and throughput will also start decreasing when the (new) capacity point is exceeded. By adding appropriate rate control, the goodput is expected to remain constant when the offered load is beyond the capacity. One implication of this design guideline is that there may be no need to vary the offered load beyond the capacity point any more.

For reliable protocols, PDR remains 100% but the argument for rate control is still valid. When reliability is provided through the traditional way (ARQ), some rate control is implicitly imposed, since retransmissions are given priority over new packets. However, when reliability is part of the “exotic” technique (e.g., intra-flow network coding embraces FEC), the source may never slow down, unless explicitly forced by rate control. In any case, exceeding the capacity of the network will lead to unpredictable behavior which will appear either in the form of increased delays, severe unfairness among flows, or reduced throughput. As an example, the gain of

MORE over traditional routing in [22] is reduced in the presence of multiple flows. A related recommendation is that a protocol should also be evaluated with multiple flows, e.g., as in [22, 55], as the rate control for each flow becomes more challenging.

Note that the best method for applying rate control in wireless networks is still an open problem and is out of the scope of this paper. In general, online mechanisms (both end-to-end, e.g., sliding-window based [55], and hop-by-hop, e.g., backpressure based [56, 78]) or even offline computations [58] can be applied. Interestingly, the importance of rate control has attracted significant interest in recent years in the theory community in the form of cross-layer optimizations (e.g. [91]). However, these theoretical works often make simplistic assumptions about the wireless channel and the interference model in trying to formulate and to solve the formulated optimization problems.

**Isolating the benefit from new optimization techniques.** The evaluation of a new protocol that exploits a new optimization technique should try to isolate the gain from this "exotic" technique, alone. The tricky part here is that in adding a new optimization technique, a new protocol often incorporates other old techniques brought down to the routing layer from the upper layers, such as end-to-end reliability and rate control. To isolate the benefit of the new optimization, such techniques should be also incorporated in the traditional routing protocols. Similarly, comparing a reliable protocol against an unreliable one should be avoided; if the new protocol includes a mechanism for end-to-end reliability, a similar mechanism should be added to the old protocol.

**Separating rate control from end-to-end reliability.** When comparing a new reliable protocol to an unreliable one, the simplest method to add end-to-end reliability to the unreliable (traditional or not) routing protocol is to run it under TCP [78]. While this approach is simple, as no modification to the protocol itself is required, it may obscure the performance gain.

If the new protocol includes only reliability but no online congestion control (e.g., as is the case with FEC-style reliability), it is overkill to run the old protocol under

TCP which includes both mechanisms which interact with each other. In this case, the throughput gap between the new and the old protocols may appear larger as a result of poor performance of TCP congestion control.

If the new protocol includes both reliability and online rate control (e.g., as is the case with ARQ-style reliability), it can be compared against the old protocol under TCP as a base-case comparison. Even so, since it is known that TCP performs poorly in wireless environments, it may still be unclear what the real gain from the new "exotic" technique is.

We advocate that in both cases, one should attempt to incorporate the reliability/rate control features of the new protocol to the old protocol, following the methodology of [78]. In this case, the comparison will be able to isolate the gain from the "exotic" technique exploited in the new protocol. We acknowledge this is not always easy to do. In some cases the reliability and congestion control mechanisms are disjoint components of the new protocol, not related to the new "exotic" technique used (e.g., in noCoCo). In this case reliability is typically provided in the traditional way (through retransmissions). This disjoint mechanism should be also incorporated to the old protocol used for comparison. In other cases, the reliability component of the new protocol may be part of the "exotic" technique itself (e.g., in MORE), and not a disjoint ARQ component. In such cases, the reliability component should be carefully added to the old protocol, for example, by adding FEC, and not by running it under TCP, so that the comparison is not affected by the negative effects of TCP's rate control mechanism.

**How to incorporate rate control to traditional routing?** Similar arguments against TCP apply here. If two unreliable protocols are compared, one with a rate control component and one without, running the second protocol under TCP is not a good solution, because the reliability mechanism is not required. What should be done is again incorporating the rate control mechanism of the new protocol to the old protocol. For example, in the evaluation of SOAR, the window-based rate control mechanism used in SOAR could be easily incorporated to traditional routing; in that case the comparison would isolate the gain of opportunistic forwarding.

**MAC autorate adaptation.** We argue that a good practice is for new “exotic” protocols to make an attempt to incorporate autorate adaptation. We acknowledge this is not an easy task and perhaps it is not always feasible. Even in those cases, we argue autorate adaptation should always be enabled for the case of traditional routing; an “exotic” protocol should be shown to outperform traditional routing both with and without autorate adaptation.

## 5.4 Summary

In this chapter, we discussed the resulting diverse set of current practices in the evaluation of “exotic” WMN routing protocols. We showed that, in contrast to traditional routing protocols, there have been no clear guidelines that drive the evaluation of these protocols; often times each new protocol is evaluated with a different methodology, with its own strengths and weaknesses. We then made several concrete suggestions for a more consistent and meaningful evaluation methodology.

Finally, we postulate that a fundamental reason for the complexity of evaluating high-throughput WMN routing protocols is that the research community still does not have a unified framework for understanding the interactions of MAC layer, congestion, interference, network coding, and reliability. WMN routing schemes are still being proposed as point solutions in a space of options; the real problem goes beyond how to evaluate them, but rather lies in how to understand the fundamental roles of their constituent parts.

# 6. CONCLUSIONS

The recent evolution of wireless networking from the ad hoc networking era to the mesh networking era has ignited a new Renaissance of routing protocol design for multihop wireless networks. The change in the design goals and the continuous demand for "high throughput" has led to a set of "exotic" optimization techniques, such as opportunistic routing and network coding, which took two important departures from traditional routing. First, they abandoned the notion of the wireless link, by exploiting *wireless broadcast* at the MAC layer. Second, they largely abandoned the layering principle, adopting a *cross-layer* approach.

The new class of routing protocols exploiting these "exotic" techniques promises a dramatic increase to overall network throughput compared to traditional routing protocols. Nonetheless, a careful consideration of the design and evaluation methodology of these new protocols reveals a different picture: the first "exotic" routing protocols are a *proof of concept*, often working particularly well only for a specific scenario, but suffering from several limitations that may severely limit their gains in different scenarios. In addition, the first "exotic" routing protocols have almost solely focused on unicast routing and have ignored multicast. Overall, there is a long way ahead in turning proof of concept to robust, practical, complete protocols.

This thesis takes a significant step towards this direction. We have designed, implemented and evaluated two robust, practical, "exotic" routing protocols that advance the state-of-the-art for multicast and unicast routing. In addition, we revealed the challenges that this new class of protocols pose to the evaluation methodology, showed the limitations of the current practices, and made suggestions for a more meaningful evaluation methodology.

In contrast to the significant innovations in high-throughput unicast routing, high-throughput multicast routing has received little attention. This thesis fills this gap by proposing *Pacifier*, the first high throughput reliable multicast routing protocol for file download applications, that require 100% Packet Delivery Ratio (PDR). One fundamental challenge to supporting high-throughput, reliable multicast is the "crying baby" problem, when one receiver with a particularly poor connection slows down the rest of the receivers. *Pacifier* is the first protocol to efficiently address this challenge for file download applications, where the strict reliability requirement makes this problem particularly challenging. Extensive performance evaluation shows that *Pacifier* significantly outperforms MORE, the only "exotic" routing protocol for multicast, which ignores this problem.

A fundamental challenge faced by all network coding based opportunistic routing protocols is determining how many coded packets each forwarding node should transmit while minimizing the coordination overhead. Existing protocols, including *Pacifier*, address this challenge by combining network coding with offline loss rate based heuristics to eliminate the overhead of feedback exchange, often at the cost of reduced performance in dynamic wireless environments. This thesis proposes a novel solution to this challenge through the design of the CCACK protocol. CCACK introduces an online **C**umulative **C**oded **ACK**nowledgment scheme that allows the protocol to sustain high performance in dynamic wireless environments, with practically zero overhead.

An additional contribution of this thesis is that both protocols have been prototyped and evaluated on MAP (Mesh@Purdue), a 22-node experimental 802.11a/b/g WMN testbed at Purdue University. Thus, the contributions of this thesis are shown to have direct practical application and significance. To facilitate further research on this newly emerged area of "exotic" routing protocol design, we have made the source code of our *Pacifier* implementation publicly available.[1]

An additional contribution of this thesis lies in the evaluation methodology of this new class of "exotic" routing protocols. This thesis is the first to examine the

[1]At the time of this thesis, the code has been downloaded by 17 research groups from 5 different countries.

diverse set of current practices in evaluating recently proposed protocols, identify their strengths and weaknesses, and show the urgent need to carefully rethink the implications of the new merged-layer routing protocol design and develop effective methodologies for meaningful and fair comparison of these protocols. Finally, we make several concrete suggestions on the desired evaluation methodology.

## 6.1 Open issues and future work

The work presented in this dissertation raises a number of interesting questions specific to the techniques presented and inspires future research directions.

### 6.1.1 Open challenges in the design of “exotic” routing protocols for WMNs

**Compatibility with traffic other than file transfers.** The majority of the initial “exotic” routing protocols (e.g., ExOR, MORE) were explicitly designed for file transfer applications. Similarly, the work in this thesis focused on file transfer applications for multicast (*Pacifier*) or unicast (CCACK). One open question is whether this type of batch based protocols are appropriate for other types of traffic, e.g., video streaming. Streaming applications have very different requirements from file transfers. On one hand, they do not require 100% PDR (though a high PDR is always desirable since it improves the user-perceived quality of the stream); on the other hand, they pose strict requirements in terms of packet delay and jitter. Due to these differences, it is not clear whether “exotic” routing protocols that guarantee 100% PDR, e.g., MORE, CCACK, or *Pacifier* can work with streaming applications.

Focusing on multicast, *Pacifier* cannot be used to address the “crying baby” problem in streaming applications, since the round robin batching scheme would violate the deadlines associated with video frames. Recently, theoretical work [92] has shown that network coding combined with multi-resolution streaming can address this problem more efficiently than multi-resolution streaming alone. Thus, one interesting future research direction is to look into practical aspects of this integration (a traditional technique coupled with an "exotic" one) and design a practical protocol that will efficiently address the "crying baby" problem for streaming applications in WMNs.

**"Exotic" routing protocols in multi-rate wireless mesh networks.** The majority of the "exotic" wireless protocols proposed so far suffer from one drawback: they work only with a single MAC bitrate (one notable exception is ROMER [80], however, its performance has only been evaluated using simulations). The reason, as we have explained in Section 5.2.3, is that this class of protocols use 802.11 broadcast at the MAC layer, which, in contrast to unicast, does not support rate adaptation. Enabling rate adaptation for broadcast based "exotic" unicast/multicast protocols is one of the next significant challenges in the area of mesh networking. Preliminary work has unsuccessfully tried to address this challenge through offline selection of the optimal rate at each node after periodic loss measurements [93]. In contrast, the lessons learned from unicast rate adaptation algorithms (which adapt the rate in very short time scales) suggest that the right approach is through online feedback. Our CCACK protocol was a first demonstration of how network coding can help to effectively compress useful feedback for broadcast based "exotic" routing protocols instead of completely eliminating it. We believe that this type of coded feedback is the key to a successful integration of rate adaptation algorithms with "exotic" broadcast based protocols.

**"Exotic" routing protocols over "exotic" radios.** "Exotic" routing protocols have been designed to work on top of traditional 802.11 radios. As hardware technology advances and the demand for spectrum continues to grow, novel radios are enhanced with many additional "exotic" features. Such "exotic" features include MIMO technology (e.g., in the new 802.11n standard), smart steerable and adaptive beamforming directional antennas for spatial separation, cognitive radios for dynamically changing the frequency space a signal occupies to utilize white spaces, and even radios and protocols that exploit interference rather than avoiding it [94]. How much gain we can expect from such radio innovations in a multihop network remains an open question. In the context of routing protocol design, additional important questions are raised: can "exotic" routing protocols be efficiently combined with exotic radios, given that several "exotic" radios (e.g., directional antennas or cognitive radios) preclude the use of wireless broadcast? What changes do we have to make to the current "exotic" routing protocols? How should we design the next generation routing protocols in order to extract the maximum possible gains from such "exotic" radios?

**Integration of exotic routing protocols in the networking stack.** Existing network coding based routing protocols have been implemented as user-level programs, either communicating with the wireless interface using raw sockets (e.g., MORE) or directly at the application layer (e.g., *Pacifier*). Although such implementations allow for faster modification and easier experimentation, they inevitably suffer some performance degradation from frequently crossing the kernel-user boundary. As a consequence, all testbed evaluations so far have used only the lowest MAC bitrates. In order to evaluate the impact of rate adaptation, or to explore the synergy with "exotic" radios that offer significantly higher bitrates, "exotic" routing protocols have to be placed in the correct place in the networking stack. Identifying the right place is itself a challenging question, as these protocols require collaboration with the lower layers, while at the same time they often implement functions such as end-to-end rate control or reliability which used to be the responsibility of upper layers.

### 6.1.2 "Exotic" techniques in other types of networks

The practical advantages of network coding (intra- or inter-flow) have been mainly demonstrated in multihop wireless networks. However, other types of wireless networks can also benefit from these "exotic" techniques. The key idea is again to exploit the broadcast property of the wireless medium along with opportunistic overhearing.

As an example, network coding can be used to design efficient retransmission schemes in single-hop networks, e.g., 802.11 WLANs or cellular networks. Under high loss rates, e.g., in the presence of a large number clients, the simple retransmission mechanism of 802.11 unicast can cause significant overhead and severely limit the throughput of the WLAN, as the AP may spend a significant portion of the air-time retransmitting lost packets. This motivates the need for novel, more efficient retransmission schemes. We illustrate how coding can help in improving the efficiency of the retransmission mechanism in a WLAN with the following example. Consider an AP and two clients $C_1$, $C_2$. The AP has two packets, $p_1$, $p_2$, one for each client, respectively. In a lossy wireless network, it may happen that both packets are lost and the AP would have to retransmit both of them. However, due to the broadcast nature of the wireless medium, it can also happen that $C_1$ received the packet $p_2$ destined to $C_2$ and $C_2$ received the packet $p_1$ destined to $C_1$. In that case, the AP can XOR the two packets and broadcast one the combined packet $p_1 \oplus p_2$. Then $C_1$ can extract its own packet $p_1$ by XORing $p_1 \oplus p_2$ with $p_2$ and similarly, $C_2$ can extract $p_2$ by XORing $p_1 \oplus p_2$ with $p_1$, thus saving one transmission. This simple idea can be extended beyond the two-client example, further improving the retransmission efficiency.

The potential benefits of network coding based retransmission schemes have been demonstrated theoretically and experimentally through the design of MU-ARQ [95] and ER [96], respectively. However, similar to the case of multihop WMNs, these first coding based retransmission protocols for WLANs are simply proof of concept, often resorting to simple heuristics, and cannot realize the full potential of the inter-flow network coding. We are currently working towards developing a novel network coding based retransmission protocol that will successfully address the limitations of these first protocols. Our approach combines the two types of network coding (intra- and interflow) with the goal of designing a systematic coding strategy rather than simple heuristics to achieve optimal theoretical performance and at the same time allow for a practical implementation.

# LIST OF REFERENCES

# LIST OF REFERENCES

[1] I. F. Akyildiz, X. Wang, and W. Wang, “Wireless mesh networks: a survey,” *Comput. Netw.*, vol. 47, no. 4, pp. 445–487, 2005.

[2] “MIT Roofnet.” http://www.pdos.lcs.mit.edu/roofnet.

[3] “Bay area wireless users group.” http://www.bawug.org.

[4] “Seattle wireless.” http://www.seattlewireless.net.

[5] “Champaign-Urbana community wireless network.” http://www.cuwireless.net.

[6] “Southampton wireless network.” http://www.sown.org.uk.

[7] “Wireless leiden.” http://www.wirelessleiden.nl.

[8] “Radiant networks..” http://www.radiantnetworks.com.

[9] “Technology For All (TFA).” http://tfa.rice.edu.

[10] “Google WiFi.” http://wifi.google.com/.

[11] V. Brika, S. Rayanchua, S. Sahaa, S. Sena, V. Shrivastavaa, and S. Banerjee, “A Measurement Study of a Commercial-grade Urban WiFi Mesh,” in *Proc. of ACM SIGCOMM/USENIX IMC*, 2008.

[12] J. Eriksson, S. Agarwal, V. Bahl, and J. Padhye, “A feasibility study of mesh networks for an all-wireless office,” in *Proc. of ACM MobiSys. Also MSR Tech. Rep. MSR-TR-2005-170,*, June 2006.

[13] 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.

[14] D. B. Johnson and D. A. Maltz, *Dynamic Source Routing in Ad Hoc Wireless Networks.* Kluwer Academic, 1996.

[15] C. E. Perkins and E. M. Royer, “Ad hoc on-demand distance vector routing,” in *Proc. of IEEE WMCSA*, February 1999.

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

[17] J. G. Jetcheva and D. B. Johnson, “Adaptive Demand-Driven Multicast Routing in Multi-Hop Wireless Ad Hoc Networks,” in *Proc. of ACM MobiHoc*, October 2001.

[18] 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.

[19] R. Draves, J. Padhye, and B. Zill, "Routing in multi-radio, multi-hop wireless mesh networks," in *Proc. of ACM MobiCom*, 2004.

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

[21] R. Ahlswede, N. Cai, S.-Y. R. Li, and R. W. Yeung, "Network information flow," *IEEE Transactions on Information Theory*, vol. 46, July 2000.

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

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

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

[25] C. E. Perkins and P. Bhagwat, "Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers," in *Proc. of ACM SIGCOMM*, August 1994.

[26] 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.

[27] D. Koutsonikolas, Y. C. Hu, and C.-C. Wang, "Pacifier: High-throughput, reliable multicast without crying babies in wireless mesh networks," in *Proc. of IEEE INFOCOM*, 2009.

[28] D. Koutsonikolas, C.-C. Wang, and Y. C. Hu, "CCACK: Efficient Network Coding Based Opportunistic Routing Through Cumulative Coded Acknowledgments," in *Proc. of IEEE INFOCOM*, 2010.

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

[30] S. Keshav, "A control-theoretic approach to flow control," *Proc. of the conference on Communications architecture and protocols*, pp. 3–15, 1993.

[31] R. Draves, J. Padhye, and B. Zill, "Comparison of routing metrics for static multi-hop wireless networks.," in *Proc. of ACM SIGCOMM*, 2004.

[32] D. Lun, M. Medard, and M. Effros, "On coding for reliable communication over packet networks," in *Proc. of 42nd Annual Allerton Conference on Communication, Control, and Computing*, 2004.

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

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

[35] 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.

[36] 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.

[37] 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.

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

[39] 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.

[40] 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.

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

[42] 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.

[43] 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.

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

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

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

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

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

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

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

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

[52] 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.

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

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

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

[56] 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.

[57] 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.

[58] 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.

[59] B. Scheuermann, C. Lochert, and M. Mauve, "Implicit hop-by-hop congestion control in wireless multihop networks," *Elsevier Ad Hoc Networks*, vol. 6, pp. 260–286, Apr. 2008.

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

[61] 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.

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

[63] madwifi, "`http://madwifi.org`."

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

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

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

[67] B. Radunovic, C. Gkantsidis, P. Key, and P. Rodriguez, "An optimization framework for opportunistic multipath routing in wireless mesh networks," in *Proc. of IEEE INFOCOM Minisymposium*, 2008.

[68] X. Zhang and B. Li, “Optimized multipath network coding in lossy wireless networks,” in *Proc. of IEEE ICDCS*, 2008.

[69] X. Zhang and B. Li, “Dice: a game theoretic framework for wireless multipath network coding,” in *Proc. of ACM MobiHoc*, 2008.

[70] J. Camp, V. Mancuso, O. Gurewitz, and E. Knightly, “A measurement study of multiplicative overhead effects in wireless networks,” in *Proc. of IEEE INFOCOM*, 2008.

[71] S. M. Das, H. Pucha, K. Papagiannaki, and Y. C. Hu, “Studying Wireless Routing Link Dynamics,” in *Proc. of ACM SIGCOMM/USENIX IMC*, 2007.

[72] R. Draves, J. Padhye, and B. Zill, “Routing in multi-radio, multi-hop wireless mesh networks,” in *Proc. of ACM MobiCom*, September 2004.

[73] L. Tassioulas and A. Ephremides, “Stability properties of constrained queing systems and scheduling for maximum throughput in multihop radio networks,” *IEEE Transactions on Automatic Control*, vol. 37, no. 12, 1992.

[74] R. K. Jain, D.-M. W. Chiu, and W. R. Hawe, “A quantitative measure of fairness and discrimination for resource allocation in shared computer systems,” tech. rep., Digital Equipment Corporation, September 1984.

[75] A. Kamerman and G. Aben, “Net throughput with IEEE 802.11 wireless LANs,” in *Proc. of IEEE WCNC*, 2000.

[76] J. Broch, D. A. Maltz, D. B. Johnson, Y.-C. Hu, and J. Jetcheva, “A Performance Comparison of Multi-Hop Wireless Ad Hoc Network Routing Protocols,” in *Proc. of ACM MobiCom*, October 1998.

[77] S. R. Das, C. E. Perkins, and E. M. Royer, “Performance comparison of two on-demand routing protocols for ad hoc networks,” in *Proc. of IEEE INFOCOM*, March 2000.

[78] B. Scheuermann, W. Hu, and J. Crowcroft, “Near-optimal coordinated coding in wireless multihop networks,” in *Proc. of CoNEXT*, 2007.

[79] 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.

[80] Y. Yuan, H. Yang, S. H. Wong, S. Lu, and W. Arbaugh, “Romer: Resilient opportunistic mesh routing for wireless mesh networks,” in *Proc. of 1st IEEE Workshop on Wireless Mesh Networks (WiMesh)*, 2005.

[81] M. Gerla, R. Bagrodia, L. Zhang, K. Tang, and L. Wang, “TCP over wireless multi-hop protocols: simulations and experiments.,” in *IEEE International Conference on Communications (ICC)*, 1999.

[82] R. de Oliveira and T. Braun, “A dynamic adaptive acknowledgement strategy for TCP over multihop wireless networks.,” in *IEEE Infocom*, 2005.

[83] K. Chen, Y. Xue, and K. Nahrstedt, “On setting TCP’s congestion window limit in mobile ad hoc networks.,” in *IEEE International Conference on Communications (ICC)*, 2003.

[84] Z. Fu, P. Zerfos, H. Luo, S. Lu, L. Zhang, and M. Gerla, “The impact of multihop wireless channel on TCP throughput and loss.,” in *IEEE Infocom*, 2003.

[85] V. Kawadia and P. Kumar, “Experimental investigations into TCP performance over wireless multihop networks.,” in *SIGCOMM E-WIND Workshop*, 2005.

[86] A. Gupta, Wormsbecker, and C. Williamson, “Experimental evaluation of TCP performance in multi-hop wireless ad hoc networks.,” in *Proc. of MASCOTS*, 2004.

[87] T. Kuang, F. Xiao, and C. Williamson, “Diagnosing wireless TCP performance problems: A case study.,” in *Proc. of SPECTS*, 2003.

[88] G. Anastasi, E. Ancilloti, M. Conti, and A. Passarella, *Experimental analysis of TCP performance in static multi-hop Ad hoc networks.* Nova Science Publisher, 2007.

[89] S. Xu and T. Saadawi, “Performance evaluation of TCP algorithms in multi-hop wireless packet networks.,” *Wireless Communications and Mobile Computing*, vol. 2, no. 1, 2001.

[90] D. Koutsonikolas, J. Dyaberi, P. Garimella, S. Fahmy, and Y. C. Hu, “On tcp throughput and window size in a multihop wireless network testbed,” in *Proc. of ACM WiNTECH*, 2007.

[91] X. Lin and N. Shroff, “The impact of imperfect scheduling on cross-layer rate control in multihop wireless networks,” in *Proc. of IEEE INFOCOM*, 2005.

[92] M. Kim, D. Lucani, X. Shi, F. Zhao, and M. Medard, “Network coding for multi-resolution multicast,” in *Proc. of IEEE INFOCOM*, 2010.

[93] 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.

[94] S. Gollakota and D. Katabi, “ZigZag Decoding: Combating Hidden Terminals in Wireless Networks,” in *Proc. of ACM SIGCOMM*, 2008.

[95] P. Larsoon and N. Johansson, “Multi-User ARQ,” in *Proc. of IEEE VTC-Spring*, 2006.

[96] E. Rozner, A. P. Iyer, Y. Mehta, L. Qiu, and M. Jafry, “ER: Efficient retransmission scheme for wireless lans,” in *Proc. of CoNEXT*, 2007.

# VITA

# VITA

Dimitrios Koutsonikolas was born in Amaliada, Greece on May 15th, 1981. He graduated from the 2nd General Lyceum of Trikala in 1999. He received a Diploma (5-year degree) in Electrical and Computer Engineering from the National Technical University of Athens, Greece, in 2004. He is currently a PhD student in the School of Electrical and Computer Engineering at Purdue University working with Prof. Y. Charlie Hu. In Summer 2007, Dimitrios did an internship with Thomson Paris Research Lab (now Technicolor Research and Innovation Lab).

Dimitrios received a Ross Fellowship from Purdue University in 2004, and a Tellabs Fellowship from the Center for Wireless Systems and Applications (CWSA) at Purdue University in 2006. He received a Best Paper Award in SENSORCOMM 2007, and won the first place in the ACM Student Research Competition (SRC), in MOBICOM 2009 and the third place in the same contest, in MOBICOM 2008. Dimitrios is a student member of USENIX, ACM, IEEE, IEEE Computers Society, and IEEE Communications Society.

Dimitrios' research interests are broadly in experimental wireless networking and mobile computing, with a focus on high performance protocol design and implementation, testbed prototyping, network measurements, and performance evaluation. His doctoral work has resulted in more than 20 research publications including 7 journal papers, 1 book chapter, and 14 conference and workshop papers in premier forums on wireless networking. Dimitrios' work can be found in the following publications:

## Journal Publications

[1] Dimitrios Koutsonikolas, Saumitra Das, Y. Charlie Hu, Ivan Stojmenovic. Hierarchical Geographic Multicast Routing for Wireless Sensor Networks. In *ACM Wireless Networks*, Vol. 16(2), pp. 449-466, February 2010.

[2] Dimitrios Koutsonikolas, Y. Charlie Hu. Exploring the Design Space of Reliable Multicast Protocols for Wireless Mesh Networks. In *Ad Hoc Networks (Elsevier) Journal (AdHoc)*, Vol. 7 (5), pp. 932-954, July 2009.

[3] Sabyasachi Roy, Dimitrios Koutsonikolas, Saumitra Das, and Y. Charlie Hu. High-Throughput Multicast Routing Metrics in Wireless Mesh Networks. In *Ad Hoc Networks (Elsevier) Journal (AdHoc)*, Volume 6 (6), pp. 878-899, August 2008.

[4] Dimitrios Koutsonikolas, Saumitra Das, and Y. Charlie Hu. An Interference-Aware Fair Scheduling for Multicast in Wireless Mesh Networks. In *Elsevier Journal of Parallel and Distributed Computing (JPDC), Special Issue on Behavior, Artefacts, and Solutions in Wireless Mesh Networks*, Vol. 68 (3), pp. 372-386, March 2008.

[5] Dimitrios Koutsonikolas, Saumitra Das, Y. Charlie Hu, Yung-Hsiang Lu, and C.S. George Lee. CoCoA: Coordinated Cooperative Localization for Mobile Multi-Robot Ad Hoc Networks. In *Ad Hoc and Sensor Wireless Networks (AH-SWN) Journal*, Vol. 3(4), pp. 331-352, 2007.

[6] Dimitrios Koutsonikolas, Saumitra Das, and Y. Charlie Hu. Path Planning of Mobile Landmarks for Localization in Wireless Sensor Networks. In *Elsevier Journal of Computer Communications (COMCOM), Special Issue on Sensor-Actuator Networks (SANETs)*, Vol 30 (13), pp. 2577-2592, September 2007.

[7] Saumitra Das, Himabindu Pucha, Dimitrios Koutsonikolas, Y. Charlie Hu, and Dimitrios Peroulis. DMesh: Incorporating Practical Directional Antennas in Multi-Channel Wireless Mesh Networks. In *IEEE Journal on Selected Areas in Communications (JSAC), Special Issue on Multi-Hop Wireless Mesh Networks*, Vol 24 (11), pp. 2028-2039, November 2006.

**Book Chapter**

[8] Saumitra Das, Dimitrios Koutsonikolas, Y. Charlie Hu. Measurement-based Characterization of a Wireless Mesh Network. Book Chapter, in *Handbook of Wireless Mesh and Sensor Networking, McGraw-Hill International*, New York, 2008.

**Conference and Workshop Publications**

[9] Dimitrios Koutsonikolas, Chih-Chun Wang, and Y. Charlie Hu. CCACK: Efficient Network Coding Based Opportunistic Routing Through Cumulative Coded Acknowledgments. In Proceedings of *IEEE INFOCOM 2010*, San Diego, CA, March 15-19, 2010. (acceptance rate 17.5

[10] Dimitrios Koutsonikolas, Y. Charlie Hu. On the feasibility of bandwidth estimation in 1x EVDO networks. In Proceedings of the *ACM Mobicom International Workshop on Mobile Internet Through Cellular Networks (MICNET 2009)*, Beijing, China, September 21, 2009.

[11] Dimitrios Koutsonikolas, Y. Charlie Hu, and Chih-Chun Wang. Pacifier: High-Throughput, Reliable Multicast without Crying Babies in Wireless Mesh Networks. In Proceedings of *IEEE INFOCOM 2009*, Rio de Janeiro, Brazil, April 19-25, 2009.

[12] Dimitrios Koutsonikolas, Y. Charlie Hu, and Chih-Chun Wang. An Empirical Study of Performance Benefits of Network Coding in Multihop Wireless Networks. In Proceedings of *IEEE INFOCOM 2009 Mini-Conference*, Rio de Janeiro, Brazil, April 20, 2009.

[13] Dimitrios Koutsonikolas, Theodoros Salonidis, Henrik Lundgren, Pascal LeGuyadec, Y. Charlie Hu, and Irfan Sheriff. TDM MAC Protocol Design and Implementation for Wireless Mesh Networks. In Proceedings of *the 4th ACM SIGCOMM International Conference on emerging Networking EXperiments and Technologies (CoNEXT 2008)*, Madrid, Spain, December 9-12, 2008.

[14] Dimitrios Koutsonikolas, Y. Charlie Hu, Konstantina Papagiannaki. How To Evaluate Exotic Wireless Routing Protocols? In Proceedings of *the 2008 ACM Workshop on Hot Topics in Networking (HotNets-VII)*, Calgary, Alberta, Canada, October 6-7, 2008.

[15] Che-Wei Chang, Akshay Kothari, Syed Ali Raza Jafri, Dimitrios Koutsonikolas, Dimitrios Peroulis, Y. Charlie Hu. Radiating Sensor Selection for Distributed Beamforming in Wireless Sensor Networks. In Proceedings of *IEEE Military Communications Conference (MILCOM 2008)*, San Diego, CA, November 17-19, 2008.

[16] Saumitra Das, Dimitrios Koutsonikolas, and Y. Charlie Hu. Practical Service Provisioning for Wireless Meshes. In Proceedings of *the 3rd ACM SIGCOMM International Conference on emerging Networking EXperiments and Technologies (CoNEXT 2007)*, New York, NY, December 10-13, 2007.

[17] Dimitrios Koutsonikolas, Jagadeesh Dyaberi, Prashant Garimella, Sonia Fahmy, and Y. Charlie Hu. On TCP Throughput and Window Size in a Multihop Wireless Network Testbed. In Proceedings of *ACM Mobicom International Workshop on Wireless Network Testbeds, Experimental evaluation and CHaracterization (WiNTECH 2007)*, Montreal, QC, Canada, September 10, 2007.

[18] Dimitrios Koutsonikolas, Saumitra Das, Y. Charlie Hu, and Ivan Stojmenovic. Hierarchical Geographic Multicast Routing for Wireless Sensor Networks. (Best Paper Award) In Proceedings of *International Conference on Sensor Technologies and Applications (SENSORCOMM 2007)*, Valencia, Spain, October 14-20, 2007.

[19] Dimitrios Koutsonikolas, Y. Charlie Hu. The case for FEC-based Reliable Multicast in Wireless Mesh Networks. In Proceedings of *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2007)*, Edinburgh, UK, June 25-28, 2007.

[20] Saumitra Das, Dimitrios Koutsonikolas, Y. Charlie Hu, and Dimitrios Peroulis. Characterizing Multi-Way Interference In Wireless Mesh Networks. In Proceedings of *ACM Mobicom International Workshop on Wireless Network Testbeds, Experimental evaluation and CHaracterization (WiNTECH 2006)*, Los Angeles, CA, September 29, 2006.

[21] Dimitrios Koutsonikolas, Saumitra Das, Y. Charlie Hu, Yung-Hsiang Lu, and C.S. George Lee. CoCoA: Coordinated Cooperative Localization for Mobile Multi-Robot Ad Hoc Networks. In Proceedings of *ICDCS International Workshop on Dynamic Distributed Systems (IEEE IWDDS 2006)*, Lisboa, Portugal, July 4-7 2006.

[22] Dimitrios Koutsonikolas, Saumitra Das, and Y. Charlie Hu. Path Planning of Mobile Landmarks for Localization in Wireless Sensor Networks. In Proceedings of *ICDCS International Workshop on Ad hoc and Sensor Networks (IEEE WWASN 2006)*, Lisboa, Portugal, July 4-7 2006.

[23] Sabyasachi Roy, Dimitrios Koutsonikolas, Saumitra Das, and Y. Charlie Hu. High-Throughput Multicast Routing Metrics in Wireless Mesh Networks. In Proceedings of *IEEE International Conference on Distributed Computing Systems (ICDCS 2006)*, Lisboa, Portugal, July 4-7 2006.

[24] Dimitrios Koutsonikolas, Saumitra Das, Himabindu Pucha, and Y. Charlie Hu. On optimal TTL sequence-based route discovery in MANETs. In Proceedings of the *2nd ICDCS International Workshop on Wireless Ad Hoc Networking (IEEE WWAN 2005)*, Columbus, Ohio, June 6-9, 2005.