Network Coded Wireless Architecture

Sachin Rajsekhar Katti

Network Coded Wireless Architecture

by

Sachin Rajsekhar Katti

M.S., Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 2005
B.Tech., Electrical Engineering, Indian Institute of Technology, Bombay, 2003

Submitted to the
Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Computer Science and Engineering

at the

Massachusetts Institute of Technology

September 2008

Author . . . .
Department of Electrical Engineering and Computer Science
August 29, 2008

Certified by . . . .
Dina Katabi
Associate Professor of Computer Science and Engineering
Thesis Supervisor

Accepted by . . . .
Terry P. Orlando
Chairman, Department Committee on Graduate Students

Network Coded Wireless Architecture

by

Sachin Rajsekhar Katti

Submitted to the Department of Electrical Engineering and Computer Science
on August 29, 2008, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Computer Science and Engineering

Abstract

Wireless mesh networks promise cheap Internet access, easy deployment, and extended range. In their current form, however, these networks suffer from both limited throughput and low reliability; hence they cannot meet the demands of applications such as file sharing, high definition video, and gaming. Motivated by these problems, we explore an alternative design that addresses these challenges.

This dissertation presents a network coded architecture that significantly improves throughput and reliability. It makes a simple yet fundamental switch in network design: instead of routers just storing and forwarding received packets, they mix (or code) packets' content before forwarding. We show through practical systems how routers can exploit this new functionality to harness the intrinsic characteristics of the wireless medium to improve performance. We develop three systems; each reveals a different benefit of our network coded design. COPE observes that wireless broadcast naturally creates an overlap in packets received across routers, and develops a new network coding algorithm to exploit this overlap to deliver the same data in fewer transmissions, thereby improving throughput. ANC pushes network coding to the signal level, showing how to exploit strategic interference to correctly deliver data from concurrent senders, further increasing throughput. Finally, MIXIT presents a symbol-level network code that exploits wireless spatial diversity, forwarding correct symbols even if they are contained in corrupted packets to provide high throughput reliable transfers.

The contributions of this dissertation are multifold. First, it builds a strong connection between the theory of network coding and wireless system design. Specifically, the systems presented in this dissertation were the first to show that network coding can be cleanly integrated into the wireless network stack to deliver practical and measurable gains. The work also presents novel algorithms that enrich the theory of network coding, extending it to operate over multiple unicast flows, analog signals, and soft-information. Second, we present prototype implementations and testbed evaluations of our systems. Our results show that network coding delivers large performance gains ranging from a few percent to several-fold depending on the traffic mix and the topology. Finally, this work makes a clear departure from conventional network design. Research in wireless networks has largely proceeded in isolation, with the electrical engineers focusing on the physical and lower layers, while the computer scientists worked up from the network layer, with the packet being the only interface. This dissertation pokes a hole in this contract, disposing of artificial abstractions such as indivisible packets and point-to-point links in favor of a more natural abstraction that allows the network and the lower layers to collaborate on the common objectives of improving throughput and reliability using network coding as the building block. At the same time, the design maintains desirable properties such as being distributed, low-complexity, implementable, and integrable with the rest of the network stack.

Thesis Supervisor: Dina Katabi
Title: Associate Professor of Computer Science and Engineering

To Shankar and Gourawwa Wali

Acknowledgments

I arrived at MIT five years ago quite without a clue, not sure what I as an electrical engineer was going to do in the CS department's networks and systems group. I am deeply indebted to my advisor, Dina Katabi, for making this difficult transition seamless and a lot of fun. She has given me a tremendous amount, I can only thank her for a subset. First, her advice and attention. It has ranged from a friendly "What's up?" to the ominous "Can I expect this working tomorrow?", without which I would not have done most of what I did in graduate school. Second, our joint work. For every hour I put into my research, I think Dina put two. Her insight, insistence on simplicity and tremendous patience with my writing have made sure I didn't stray. Finally, Leo. Leo's time at MIT overlaps mine, and his "antics" have been an endless source of mischief and fun during long nights close to deadlines.

I am likewise massively indebted to Muriel Medard and Hari Balakrishnan. I do not understand how someone can accomplish so much and manage so many things, yet remain so calm and relaxed, but Hari and Muriel manage to pull this feat off without breaking a sweat. Muriel, with her constant theoretical insights has provided many a "Ah, this is how I can simplify this complex theoretical idea and use it to solve my problem" moments, while Hari has done the same from the systems side, molding my frenzied thought process into simple, clear and logical ideas. This dissertation reflects both their philosophies, and is all the better due to this union.

In addition to so generously serving on both my RQE and thesis committees, Robert Morris taught me the importance of being precise. With his friendly skepticism and insistence on detail, Robert's sharp questions forced me to think harder about the systems I built and how I evaluated them, and more importantly influenced my research taste.

Co-author Shyamnath Gollakota is the reason that our SIGCOMM paper on ANC was submitted on time. Under extreme pressure and without much prior experience during his first year, Shyam helped me finish the ANC paper in under a month. Co-author Hariharan Rahul delivered analogously for the SIGCOMM paper on COPE, spending many sleepless months wrestling with TCP. Szymon Chachulski and Michael Jennings collaborated on the MORE [18] paper which provided the genesis of the MIXIT system. Sidharth Jaggi stimulated with his sharp intuition and entertained with his jokes, while drilling in me the importance of accurate bibliography.

This dissertation builds on several papers [72, 69, 70]. Those papers were much improved by comments from, and conversations with, the following people. Michael Walfish, Samuel Madden, Hariharan Rahul, Nate Kushman, Srikanth Kandula, Mythili Vutukuru, Kyle Jamieson, Shyamnath Gollakota, Maxwell Krohn, Jeremy Stribling, James Cowling, Asfandyar Qureshi, Bret Hull, Ramakrishna Gummadi, Rohit Singh and the anonymous reviewers at Hotnets, SIGCOMM and NSDI.

I am fortunate to have been at MIT for the past five years. It is not hard to find people smarter than you are and I have profited immensely from their suggestions, high standards and friendly company. Hariharan Rahul, Nate Kushman, Shyamanth Gollakota, Srikanth Kandula, Jaeyeon Jung and Grace Woo deserve credit for tolerating me as their office mate at various points. Rahul, thank you for those comments on papers, talks and most importantly titles! Nate, thank you for splitting your hair so often for me, talks will never be the same without that.

I have learned much and received helpful feedback from many others at MIT, including Dan Abadi, Magdalina Balazinska, Manish Bharadwaj, Micah Brodsky, Vladmir Bychkovsky, Jennifer Carlisle, Frank Dabek, Bryan Ford, Michel Goraczko, Frans Kaashoek. Maxwell Krohn, Chris Lesniewski-Laas, Ben Leong, Jinyang Li, Desmond Lun, Samuel Madden, Athicha Muthitacharoen, Stan Rost, Eugene Shih, Ali Shoeb and Jeremy Stribling.

There is more to life than just work. Rohit, Srikanth, Sid, Shashi, Parikshit, Sujay, Lakshmi, Mukul have been very good friends, allowing me to vent and providing refreshing company after long days and weeks. Tang has been home for the last five years, with the THRA providing a wonderful community. I was fortunate to be part of a great softball team (Exponential Run Time), which has provided much needed relief from work and introduced me to the passion of baseball in Boston.

There are many people who influenced me to enter graduate school in the first place and who equipped me with the needed skills. Ian Pratt and Jon Crowcroft showed me how research could be fun and were the primary reasons I even decided to apply for graduate school. The professors at IIT Bombay taught me the basic skills I have depended on to succeed in my graduate career. Going back even further, Father Robert and Geeta Naidu at St. Xaviers taught me the joy of math and related disciplines, and tried hard to teach me how to write.

Seema has been a rock behind me these past five years. Her selfless love and encouragement make me want to excel. I am grateful to her for enriching my life. Words cannot express my gratitude to my parents and my brother Amit for their support and love throughout my life. My parents' sacrifices for educating me, their constant encouragement from overseas and their insistence on aspiring for and working towards the best one could hope for have been instrumental to what I have accomplished.

And finally, yet most importantly, my grandparents, the Walis and Kattis. Providing nature and nurture through my early years, they taught me the importance of hard work and persistence. Without their love, care and encouragement, I would not have come this far. This thesis is dedicated to them.

Previously Published Material

Chapter 3 revises a previous publication [72]: S. Katti, H. Rahul, D. Katabi, W. Hu, M. Medard and J. Crowcroft. XORs in the Air: Practical Wireless Network Coding, *Proc. ACM SIGCOMM*, Sept. 2006.

Chapter 4 revises a previous publication [69]: S. Katti, S. Gollakota and D. Katabi. Embracing Wireless Interference: Analog Network Coding, *Proc. ACM SIGCOMM*, Aug. 2007.

Chapter 5 revises a previous publication [70]: S. Katti, D. Katabi, H. Balakrishnan and M. Medard. Symbol-level Network Coding for Wireless Mesh Networks, *Proc. ACM SIGCOMM*, Aug. 2008.

Contents

List of Figures

List of Tables

Chapter 1
Introduction

Wireless, in its various forms, is an increasingly dominant communication medium. It provides the means for mobility, city-wide Internet connectivity, distributed sensing, etc. Most wireless networks, however, offer only one hop wireless connectivity through access points, which have to be connected by wired links to the Internet. Hence, to achieve blanket coverage, entire buildings and neighborhoods have to be wired, which is typically quite expensive and hard to maintain.

Wireless mesh networks have been proposed to achieve the vision of wireless everywhere. In mesh networks, multiple radios are deployed in a neighborhood or a building. The radios connect to each other via wireless links to form a multi-hop wireless network, with a few of them acting as gateway nodes that connect the wireless network to the Internet. Packets traverse multiple wireless links before reaching the gateway and finally the wired Internet. Mesh networks extend the coverage area without expensive wiring, offering cheap and moderately fast connectivity, sufficient for accessing the Internet assuming normal browsing habits.

Next generation applications, however, are highly demanding. Consumers expect to be able to access video over the Internet, share large files, ship high definition multimedia to entertainment devices in their homes, among other things. These activities require high throughput, which current wireless mesh networks struggle to provide.

Motivated by these problems, in this dissertation, we propose a new design to build high performance wireless mesh networks. The key idea underlying our design is to give routers the ability to “code” packets before forwarding them, i.e., to perform network coding. Traditionally, routers operate using a “store and forward” approach; they buffer received packets and forward them unmodified to the next router along the path. In this dissertation, we break this convention. Routers could, if they wish, combine information from different packets at the packet, symbol, or signal level. We show that this simple yet fundamental shift in router operation can yield large throughput gains over the current state of the art.

Our work builds on the theory of network coding. Network coding was first introduced in a landmark paper by Ahlswede et al. [2], which showed that having the routers mix information in different messages allows the communication to achieve multicast capacity. Subsequent work demonstrated that, for multicast traffic, linear codes are sufficient to achieve the maximum capacity bounds [85], and developed polynomial time algorithms [63] as well as distributed random linear codes [58] for encoding and decoding. This work as well as more recent papers on attack resilience [62, 76, 56], and coding graphs that minimize a cost metric [90, 91] have theoretically established that network coding improves network throughput and reliability.

To ensure analytical tractability, however, prior work tends to ignore practical constraints. The usual approach is to model the network as a graph and assume multicast communications (since network coding for unicast is a largely unknown territory). It is also common to assume that the senders and receivers are fixed and given, and that the traffic rates are known, and do not change. In this framework, prior work shows how to run a min-cost flow optimization to find the optimal subgraph (say one that minimizes the number of transmissions) [91, 100, 90]. The subgraph dictates the coding: each node generates linear combinations of the packets on its incoming edges and broadcasts them to neighbors on the outgoing edges.

Unfortunately, the reality of wireless mesh networks is quite different. First and most importantly, traffic is unicast. Second, senders and receivers are unknown *a priori*; they do not signal their desire to communicate, but just start sending packets. Traffic is usually bursty, and the sending rate is unknown in advance even to the sender itself, and varies over time. Also, connectivity in a wireless network is highly variable due to changing channel conditions and interference from nearby transmitters. Thus, in practice, the wireless medium is highly unpredictable and inhospitable to applying the existing analytical network coding algorithms.

This dissertation provides a practical design for wireless network coding. The key difference from past work is to take an opportunistic approach to algorithm and system design. We design new network coding algorithms that are opportunistic, i.e., they make no assumptions on the traffic or the underlying topology, but identify coding opportunities as they arise, and exploit them to increase throughput and reliability. Unlike most prior work, the algorithms are designed for unicast flows. Being opportunistic, our algorithms can cope with changing traffic and medium state, and handle applications with varying or bursty sending rates.

We introduce three new systems: COPE, ANC and MIXIT. COPE is a novel packet-level network coding technique that opportunistically exploits the shared redundancy created by wireless broadcast to compress transmissions and hence increase throughput. ANC performs network coding on analog signals, showing that interference need not be harmful, and can rather be harnessed to increase throughput. Finally, MIXIT introduces a symbol-level network coding technique that takes advantage of wireless spatial diversity to perform opportunistic routing on symbols, increase concurrency, and thereby deliver high throughput without compromising end-to-end reliability.

The systems in this dissertation make a strong case for network coding as an alternative design for wireless mesh networks, a design that can deliver significant throughput and reliability gains. We show that network coding can be seamlessly integrated into the current network stack at various layers. COPE introduces a coding shim between the IP and MAC layers, while MIXIT is a cross-layer approach where the physical and network layers co-operate using clean interfaces to build a symbol-level wireless network. ANC shows how to push network coding all the way to the analog domain, operating over signals instead of bits. All three systems were implemented and evaluated in actual wireless testbeds, and our experimental results demonstrate that they deliver large gains in practice. We describe each of them briefly below.

■ 1.0.1 COPE

First, COPE, the subject of Chapter 3, designs a new packet-level network coding technique that improves the throughput of wireless mesh networks with multiple unicast flows. With COPE, a router uses network coding to perform in-network compression of packets in its

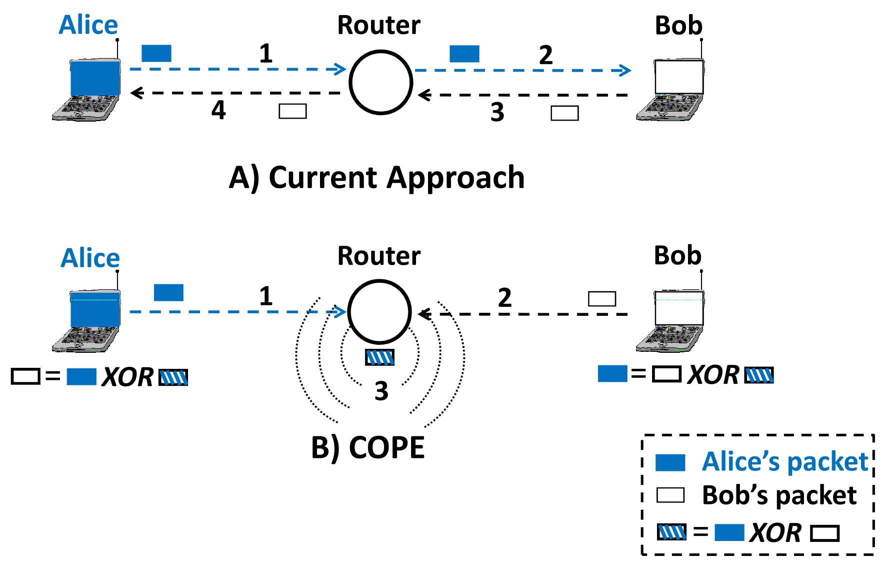


Figure 1-1: A simple example showing how COPE increases throughput. It allows Alice and Bob to exchange a pair of packets using 3 transmissions instead of 4 (numbers on arrows show the order of transmission).

outbound queue; it intelligently combines packets using the simple XOR operator and broadcasts the coded packet to multiple next hops in a single transmission. The next hops decode the coded packet and repeat the same process until each packet reaches its destination. The key insight behind this technique is that, due to multi-hop routing and the broadcast nature of wireless, nodes surrounding a router may have already overheard many of the packets in the router's queue, creating a pool of shared redundancy in that neighborhood. The router can exploit this shared redundancy to perform in-network compression of outgoing packets, delivering multiple packets in a single transmission and consequently increasing throughput.

COPE is best demonstrated through an example. Consider the scenario in Fig. 1-1, where Alice and Bob want to exchange a pair of packets via a router. In the current forwarding architecture, Alice sends her packet to the router, which forwards it to Bob, and Bob sends his packet to the router, which forwards it to Alice. This process requires four time slots. COPE employs network coding to reduce the number of time slots needed to deliver the two packets. In COPE, Alice and Bob send their packets to the router, one after the other. The router bit-wise XORs the two packets and broadcasts the XOR-ed version. Alice and Bob obtain each other's packet by XOR-ing the coded packet received from the router with the packet they transmitted earlier. This process requires three time slots in-

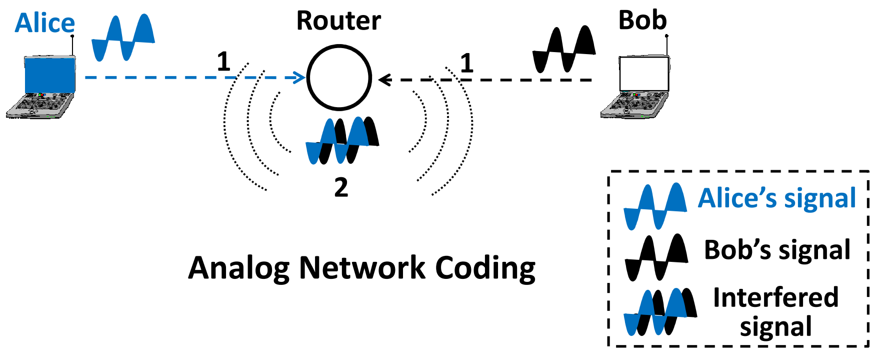


Figure 1-2: A simple example showing how ANC increases throughput. It allows Alice and Bob to exchange a pair of packets in two time slots instead of the four time slots needed in the current approach (numbers on arrows show the order of transmission).

stead of four. The freed slot can be used to send new data, increasing wireless throughput.

Chapter 3 generalizes this idea to arbitrary topologies and traffic patterns and integrates it into the current network stack, showing even larger throughput gains than are apparent from this simple example.

■ 1.0.2 ANC

The second system in this dissertation, presented in Chapter 4, is Analog Network Coding (ANC), which improves network throughput by increasing the number of concurrent transmissions. ANC uses the wireless channel itself to perform network coding over analog signals, thus encouraging certain nodes to transmit concurrently and strategically interfere. Routers themselves do not attempt to decode packets from the interfered signal, but amplify the interfered signal and forward it. At the destinations, nodes use a novel decoding algorithm that exploits network-layer information to cancel out the interference and recover the signal the node wants, and consequently the packet.

We demonstrate the operation of ANC with the same example we used for COPE, shown in Figure 1-2. As discussed above, Alice and Bob wish to exchange a pair of packets, and they require four time slots with the current architecture and three with COPE. ANC does even better; it accomplishes the exchange in two time slots. In ANC, Alice and Bob transmit their packets *simultaneously*, allowing their transmissions to interfere at the router. This consumes a single time slot. Due to interference, the router receives the sum of Alice's and Bob's signals, which it cannot decode. The router simply amplifies and forwards the received interfered signal. This consumes a second time slot. Since Alice knows

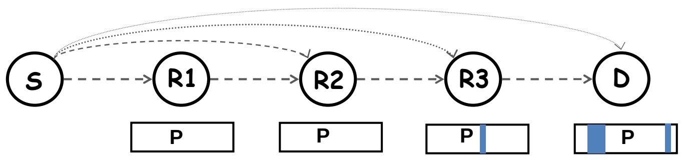


Figure 1-3: A simple example showing how MIXIT increases throughput. The source, S, wants to deliver a packet to the destination, D. The figure shows the receptions after S broadcasts its packet, where dark shades in a packet refer to erroneous bits. The best path traverses all routers $R1$, $R2$ and $R3$. Traditional routing makes $R1$ transmit the packet ignoring any opportunistic receptions. Packet-level opportunistic routing exploits the reception at $R2$ but ignores that most of the bits have made it to $R3$ and D. MIXIT exploits correctly received bits at R3 and D, benefiting from the longest links.

the packet she transmitted, she also knows the analog signal corresponding to her packet. She can therefore subtract her known signal from the received interfered signal to recover Bob's signal, and then decode Bob's packet. Bob can similarly recover Alice's packet. Thus, compared to the traditional approach, ANC reduces the required time slots from four to two, doubling the wireless throughput.

Chapter 4 transforms this idea into a practical design, addressing the problems of channel distortion and the lack of synchronization between Alice's and Bob's transmissions. It actually exploits the asynchrony that naturally exists between concurrent senders to estimate wireless channel distortion and compensate for it before disentangling interfered signals. Chapter 4 also extends the idea to more topologies and traffic patterns and provides a prototype implementation and a testbed evaluation.

■ 1.0.3 MIXIT

The third system in this dissertation, presented in Chapter 5, is MIXIT, which improves network performance by exploiting wireless spatial diversity. In both traditional routing protocols as well as more recent opportunistic approaches [65, 12, 18], each intermediate node forwards a packet only if it has no errors. In contrast, MIXIT takes a much looser approach, a forwarding node does not attempt to recover from any errors, or even bother to apply an error detection code (like a CRC). Instead MIXIT uses physical layer hints [65] (these hints reflect the PHY's confidence in its demodulation and decoding) to filter out incorrect bits in a received packet, and opportunistically route the correctly received bits to their destination. Thus, surprisingly, in MIXIT, the destination can receive a fully correct packet even though no intermediate node may have received the packet correctly.

MIXIT is best demonstrated through an example. Consider Fig. 1-3, where a source, S, tries to deliver a packet to a destination, D, using the chain of routers $R1$, $R2$, and $R3$. It is possible that when the source broadcasts its packet, $R1$ and $R2$ hear the packet correctly, while $R3$ and D hear the packet with some bit errors. Traditional routing ignores the "lucky" reception at $R2$ and insists on delivering the packet on the predetermined path, i.e., it makes $R1$ forward the packet to $R2$ again. In contrast, recent opportunistic routing protocols [12, 18] capitalize on such lucky receptions (at $R2$) to make long jumps towards the destination, saving transmissions. However, by insisting on forwarding fully correct packets, current opportunistic protocols miss many opportunities to save transmissions and increase throughput. In particular, they do not take advantage of the correct bits that already made it to $R3$ and even to the destination, D. Note that because of spatial diversity [96, 124], the corrupted bits at $R3$ and D are likely in different positions. Thus, in MIXIT, $R3$ only transmits the bits that D did not receive correctly, delivering the entire correct packet to D with far fewer transmissions. MIXIT therefore significantly increases throughput without compromising end-to-end reliability.

The principal component of MIXIT is a new *symbol-level network code* that operates on small groups of bits called symbols. This new code allows the network to route groups of bits to their destination with low overhead. It also works as an error correcting code, enabling the destination to correct any incorrect bits that might seep through.

Chapter 5 describes the design and implementation of MIXIT, including the symbol-level network code and new MAC and forwarding protocols built on the symbol-level substrate. It also provides a prototype implementation in software radios and a testbed evaluation.

■ 1.1 Contributions

The high-level contribution of this dissertation is to build a strong connection between theoretical network coding and wireless systems, with novel algorithms that strengthen the theory as well as prototype implementations and evaluations that deliver large gains in practice. The specific contributions of this dissertation highlight different aspects of this connection and are described below.

1.1.1 Network Coding: An Alternative Design for Wireless Mesh Networks

This dissertation establishes network coding as a basic building block for an alternative design of wireless mesh networks. The systems in this dissertation demonstrate how network coding provides a framework to exploit the fundamental properties of wireless systems. For example, in COPE, network coding gives us the ability to exploit the shared redundancy created by broadcast. In ANC, analog network coding provides the engineering tools needed to take advantage of strategic interference. Finally, in MIXIT, our symbol-level network code enables us to leverage wireless spatial diversity at a low overhead. Further, our implementations and evaluations in actual testbeds demonstrate the large gains network coding provides in practice, making a strong case for network coding as an elegant and practical engineering tool for designing wireless mesh networks.

1.1.2 Novel Network Coding Algorithms

The systems in this dissertation present novel opportunistic network coding algorithms that enrich the existing theory. We highlight the main algorithmic contributions below:

- COPE introduces a general algorithm for network coding multiple unicast sessions. It shows that a simple yet effective technique to build network coding algorithms for multiple unicast sessions is to ensure that every coded packet can be decoded at its next hop. This avoids the problem of data from independent sessions getting mixed up and being undecodable eventually at the destination.

- ANC extends network coding to the signal-level, providing a practical decoding algorithm to disentangle interfered signals. It shows that interference can often be viewed as a network code, which we can decode using network-layer information. The key insight behind the decoding algorithm is that asynchrony between transmitters is a blessing in disguise, and could be exploited to estimate channel distortion and disentangle signals.

- MIXIT introduces new network coding algorithms that take advantage of soft information, allowing the physical and network layers to cooperate instead of working in isolation. Previous network coding techniques work on correctly decoded packets. MIXIT extends network coding to work on likely-correct high-confidence symbols without compromising end-to-end reliability. Furthermore, MIXIT shows how network coding ideas can be used to bring the theoretical ideas of co-operative diversity [79] and backpressure routing [123] to practice.

■ 1.1.3 Integration of Network Coding Into the Network Stack

This dissertation shows how network coding can be integrated into the current network stack at different layers. COPE inserts a coding shim between the routing and MAC layers to exploit wireless broadcast. The shim is transparent to layers above and below. ANC on the other hand works at the signal level below the physical layer. Finally, MIXIT is a cross-layer approach where the physical and network layers collaborate on improving throughput and reliability.

Our work gives network designers the flexibility to deploy network coding at various granularities depending on their constraints. If they are designing networks using off-the-shelf 802.11 hardware, where only the layers above the MAC are at their control, they can deploy network coding at the packet level [72, 18] (like in COPE) to take advantage of wireless broadcast. If the link layer can be modified, MIXIT can be deployed at the symbol level to harvest higher gains. Finally, if the wireless hardware itself can be modified, or if the PHY layer is implemented in software, network designers can deploy ANC to exploit strategic interference, as well as the other two techniques at the higher layers. Generally, higher flexibility in network design provides better performance, but even when network coding can only be deployed in software at the packet level, the gains are significant.

■ 1.1.4 Practical Evaluations Demonstrating Large Gains

All three systems presented in this dissertation have been deployed and evaluated in actual testbeds. The evaluations reveal that each system provides large throughput increases, but the exact gain depends on the topology and traffic mix. We summarize the major experimental contributions below:

- We evaluate COPE in a 20-node 802.11 testbed and find that COPE largely increases network throughput. The gains vary from a few percent to several fold depending on the traffic pattern, congestion level and transport protocol used. When the wireless medium is congested and the traffic consists of many random UDP flows, COPE increases the throughput of our testbed by $3\times - 4\times$. With TCP, however, hidden terminals create a high collision rate which prevents TCP from sending enough to utilize the medium and thus does not create coding opportunities. With no hidden terminals, TCP's throughput increases by an average of 38%.

- We evaluate ANC in a testbed of GNURadio [41] software radios. Empirical results show that our analog network coding technique decodes interfered signals with an average bit error rate as low as $2-4\%$, which is masked by the use of error-correcting codes. As for throughput, it increases by 70% in comparison to traditional wireless routing.
- We evaluate MIXIT using a GNURadio [41] software radio implementation on a 25-node testbed running the Zigbee (802.15.4) protocol. The experiments show that MIXIT provides a $2.8\times$ gain over state-of-the-art packet based opportunistic routing protocols [12, 18] under moderate load and $2.1\times$ gain under light load. The gain over current routing is even higher and reaches $3.9\times$.

■ 1.2 How to Read This Dissertation?

This dissertation can be divided and read along multiple axes depending on what the reader is looking for. The chapters themselves have been ordered from a pedagogical point of view. We start with COPE, described in Chapter 3, which contains a simple, elegant idea and offers a gentle and intuitive introduction to wireless network coding. ANC, described in Chapter 4, presents a similar idea but at the level of signals, and introduces the reader to the intricacies of wireless signal processing. Finally, MIXIT, described in Chapter 5, is a sophisticated system that marries the concepts of network coding, opportunistic routing, flexible MAC protocols, and soft information into a single complete system.

The same systems can be viewed through a network coding lens as inter-flow and intra-flow coding techniques. COPE is an inter-flow network coding technique that sheds insight on the problem of applying network coding to multiple unicast sessions. ANC has both intra and inter-flow network coding, and further extends network coding to the signal level. Finally, MIXIT is an intra-flow network coding technique that leverages physical-layer soft information. Further, the chapter on MIXIT shows how ideas from co-operative diversity and backpressure routing can be naturally combined with network coding to improve performance.

The dissertation caters to both the theoretician and the practitioner. Chapter 3 develops a new network coding algorithm which has connections to the index coding problem [5]. It also provides new problems for theoretical investigations, especially the question of capacity scaling when network coding is used. At the same time, the idea of shared redundancy presented in the chapter is quite general, and can be found in a variety of practical settings [36], where systems designers could possibly use the COPE coding algorithm with some modifications. Chapter 4 has a capacity analysis that sheds light on the theoretical improvements possible with analog network coding, as well as gives insight to system designers on the practical situations where analog network coding is beneficial and where it is not. Chapter 5 has a multitude of ideas which draw inspiration from theoretical literature and applies them to practical settings. It uses a new load-aware routing algorithm that has connections to theoretical work on congestion control and backpressure [28]. Further, the error correcting code used is based on fairly recent theoretical techniques built on rank distance codes [118].

Chapter 2
Background

In this chapter, we provide an overview of network coding and current wireless mesh network design. We begin in Section 2.1 with an introduction to network coding. Section 2.2 discusses the benefits of network coding, while Section 2.3 discusses prior work in network coding relevant to this dissertation. Section 2.4 discusses the present architecture of wireless mesh networks and how they are built. It also describes prior work on improving performance at various layers in the mesh network architecture.

■ 2.1 What is Network Coding?

All communication networks today make a basic assumption that *information is separate*. Thus, whether it is packets in a network, or signals on a phone network, information is transmitted in the same way as cars share a highway or fluids share pipes. That is, independent data streams may share network resources, but the information itself is separate. Routing, data storage, error control and generally most network functions are based on this assumption.

Network coding breaks this assumption. Instead of just storing and forwarding data, nodes may combine several input packets into one or several output packets. Network coding is best demonstrated through the famous butterfly example shown in Figure 2-1. The source S_1 wants to deliver a packet P_1 to both D_1 and D_2, and source S_2 wants to send packet P_2 to the same two receivers. Assume all links have a capacity of one packet

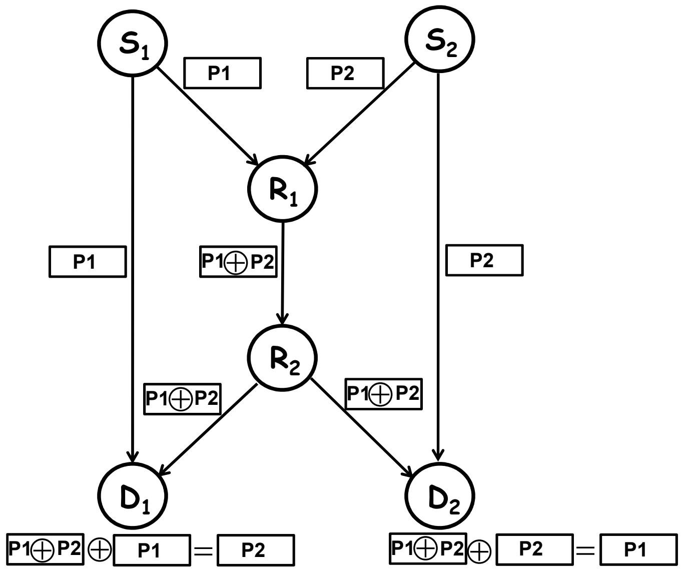


Figure 2-1: A simple scenario showing how network coding improves throughput. All links have a capacity of one message per unit of time. By sending the XOR of P_1 and P_2 on the middle link, we can deliver two messages per unit of time to both receivers.

per second. If routers R_1 and R_2 only forward the packets they receive, the middle link will be a bottleneck. The two routers, for every second, can either deliver P_1 to D_2 or P_2 to D_1. In contrast, if the router feeding the middle link XORs the two packets and sends $P_1 \oplus P_2$ (or any linear combination of P_1 and P_2), as shown in the figure, both receivers can obtain both packets. D_1 would get P_2 by XORing packet P_1 which it received on the direct link from S_1 with $P_1 \oplus P_2$ and similarly D_2 recovers P_1. Thus, network coding can obtain a multicast throughput of two packets/second, strictly better than the routing approach which can at best achieve 1.5 packets/second.

Linear network coding, is in general, similar to this example, with the difference that the XOR operation is replaced by a linear combination of the data, interpreted as numbers over some finite field. This allows for a much larger degree of flexibility in the way packets can be combined. Thus, routers instead of forwarding packets, create linear combinations of incoming packets to create coded packets and then forward them on their outgoing links. We briefly describe the encoding and decoding process in the following sections.

2.1.1 Encoding

Assume that each packet consists of L bits. When the packets to be combined do not have the same size, the shorter ones are padded with trailing 0s. We can interpret s consecutive bits of a packet as a symbol over the finite field F_{2^s}, thus each packet is a vector of L/s symbols. With linear network coding, outgoing packets are linear combinations of the original packets, where addition and multiplication are performed over the finite field F_{2^s}.

Let $P_1, P_2, \ldots, P_n$ be the n original packets generated by one or several sources. In linear network coding, each packet X in the network is associated with a vector of coefficients $\vec{g} = g(1), g(2) \ldots g(n)$ in F_{2^s} called the *code vector*. The code vector tells us how packet X can be derived form the original packets,

$$X = \sum_{i=1}^{n} g(i) P_i \tag{2.1}$$

The summation has to occur for every symbol position, i.e., $X(k) = \sum_{i=1}^{n} g(i) P_i(k)$ where $P_i(k)$ and $X(k)$ are the k'th symbols of P_i and X respectively. In the example of Figure 2-3, the field is $F_2 = \{0, 1\}$, a symbol is a bit, and the linear combination sent by R_1 after receiving P_1 and P_2 is $P_1 \oplus P_2$. The code vector is carried in the header of the encoded packet X. The vector is used by recipients to decode the data, as explained later.

Encoding can be performed recursively, namely, with already encoded packets. Consider a node that has received and stored a set $(\vec{g}_1, X_1), \ldots, (\vec{g}_m, X_m)$ of encoded packets where $\vec{g}_j$ is the code vector for packet X_j. This node may generate a new encoded packet $(\vec{g}', X')$ by picking a set of coefficients $\vec{h} = h(1), \ldots, h(m)$ and computing the linear combination $X' = \sum_{j=1}^{m} h(j) X_j$. The corresponding code vector $\vec{g}'$ is not simply equal to $\vec{h}$, since the coefficients are not with respect to the original packets $P_1, \ldots, P_n$; but with some straightforward algebra we can see that it is given by $\vec{g}' = \sum_{j=1}^{m} h(j) \vec{g}_i$. This operation may be repeated at several nodes in the network.

2.1.2 Decoding

The destination receives coded packets, $X_1, \ldots, X_m$ with the corresponding code vectors $\vec{g}_1, \ldots, \vec{g}_m$. As discussed before, each coded packet is a linear combination of the original packets. Hence, in order to retrieve the original packets, it needs to solve the system of equations:

$$X_j = \sum_{i=1}^{n} g_j(i) P_i \tag{2.2}$$

where the unknowns are the original set of packets P_i. This is a linear system with m equations and n unknowns. As long as $m \geq n$ and there are at least n linearly independent combinations, then the system of equations is solvable and the original packets can be decoded. Hence, the network has to ensure that it delivers at least n linearly independent packets to the destination. This is easy to ensure, as we discuss next.

■ 2.1.3 How to select the linear combinations?

The problem of network code design is to select what linear combinations each node of the network performs to ensure that the destination node receives at least n linearly independent combinations from which it can decode the original packets. A simple algorithm is to have each node in the network select uniformly at random the coefficients over the field F_{2^s}, in a completely independent and decentralized manner [58]. With random network coding there is a certain probability of selecting linearly dependent combinations [58]. This probability is related to the field size 2^s. Simulation results indicate that even for small field sizes (for example, $s = 8$) this probability becomes negligible [133].

Alternatively, we can use deterministic algorithms to design network codes. The polynomial-time algorithm for multicasting in [63], sequentially examines each node of the network, and decides what linear combinations each node performs. Since each node uses fixed linear coefficients, the packets do not need to carry the code vector. There also exist deterministic decentralized algorithms that apply to restricted families of network configurations.

■ 2.1.4 Practical Issues

Decoding

Decoding requires solving a set of linear equations, which can be accomplished efficiently using Gaussian elimination. A node stores the code vectors it receives as well as the corresponding packets, row by row, in a so-called decoding matrix. Initially, this matrix is empty. When an encoded packet is received, it is inserted as the last row into the decoding matrix and Gaussian elimination is performed to transform it to a triangular matrix.

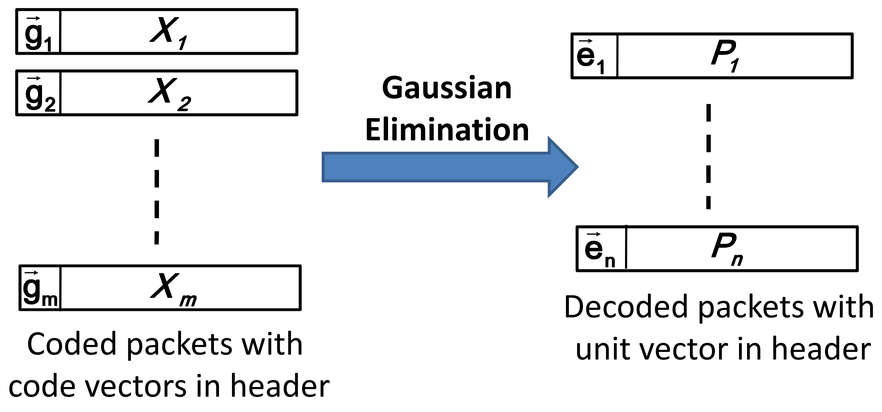


Figure 2-2: Decoding Network Codes. The destination performs Gaussian elimination on the $m > n$ received coded packets. When the code vector portion of the packet has been transformed to a unit vector, the corresponding original packet is recovered.

A received packet is called *innovative* if it increases the rank of the matrix. If a packet is non-innovative, it is reduced to a row of 0s by Gaussian elimination and is ignored. As soon as the code vector part of the matrix contains a row of the form e_i (a unit vector with a single one at the i'th position), this node knows that X is equal to the original packet P_i. This occurs at the latest when n linearly independent code vectors are received. Note that decoding does not need to be performed at all nodes of the network, but only at the receivers. Figure 2-2 shows how Gaussian elimination transforms the m received coded packets into the original n packets.

Batches

For practical purposes, the size of the matrices with which network coding operates has to be limited. This is straightforward to achieve by grouping packets into *batches*, and mandating that only packets in the same batch can be combined. The size of the batch has an impact on the performance of network coding, and is related to the size of the finite field. Typically, a small finite field increases the probability of non-innovative transmissions and reduces performance. But for almost all practical systems where the typical field size is 256 (i.e., each finite field element is a byte in a packet), batch size is not a concern.

Finite-field operations

Network coding requires operations in F_{2^s} , i.e., operations on strings of s bits. Addition is the standard bitwise XOR. For multiplication, one interprets a sequence $b_0, \ldots, b_s$ of s bits as the polynomial $b_0 + b_1x + \ldots + b_{s-1}x^{s-1}$. Then one picks a polynomial of degree s that is irreducible over F_2 (there are several of them, and each gives a different representation of F_{2^s}). Multiplication is obtained by first computing the usual product of two polynomials (which gives a polynomial of degree possibly larger than s), and then computing the remainder modulo the chosen irreducible polynomial. Division is computed by the Euclidian algorithm. Both multiplication and division can be implemented efficiently with s shifts and additions.

If s is small (e.g., $s = 8$), a faster alternative is to use discrete logarithms. In a finite field there exists at least one special element α, called generator (for example, $\alpha = 1 + x$ is a generator in Rijndael's representation of F_{2^8}), with the property that any non-zero x can be written in a unique way $x = \alpha^{\log(x)}$; where $\log(x)$ is called the logarithm. Since $\log(xy) = \log(x) + \log(y)$, multiplication and division can be implemented by looking up the two tables that map x to $\log(x)$ and vice versa. For $s = 8$ these are two tables of size 255 bytes each.

■ 2.2 What are the Benefits of Network Coding?

Network coding can be applied to a wide variety of scenarios, from static wired/wireless networks to ad-hoc mobile wireless networks. It provides provable throughput benefits, and also achieves those gains with distributed algorithms which simplify network operation.

■ 2.2.1 Theoretical Throughput Gains

Network coding achieves the optimal network capacity for multicast flows. Specifically, consider a network that can be represented as a directed graph (typically, this is a wired network). The vertices of the graph correspond to nodes, and the edges of the graph corresponds to links. Assume that we have N sources, each sending information at some given rate, and M receivers. All receivers are interested in receiving all packets. Ahslwede et al. [2] showed that network coding gives the following powerful guarantee *"assume that the source rates are such that, without network coding, the network can support each receiver in isolation (i.e. each receiver can decode all sources when it is the only receiver in the network). With an appropriate choice of linear coding coefficients, the network can support all receivers simultane-*

ously.”

In other words, when the N receivers share the network resources, each of them can receive the maximum rate it could hope to receive, even if it were using all the network resources by itself. Thus, network coding can help to better share the available network resources. There exist directed graphs where the throughput gains of network coding for multicasting can be very significant [115]. However, in undirected graphs (e.g., a wired network where all links are full-duplex) the throughput gain is at most a factor of two [20].

Network coding may offer throughput benefits not only for multicast flows, but also for other traffic patterns, such as unicast. Consider again Figure 2-1 but assume now that source S_1 transmits packets to destination D_2 and S_2 to D_1. We can use the same network code as in the multicast example to achieve a unicast rate from each source to its corresponding receiver of 1 packet/second. Without network coding, sources can only transmit at a rate of $1/2$ packets/second to their corresponding receiver.

An interesting point is that network coding allows to achieve the optimal throughput when multicasting using polynomial time algorithms. In contrast, achieving the optimal throughput with routing (which is often far less than the optimal throughput achievable with network coding) is NP-complete: this is the problem of packing Steiner trees [35]. Thus, even when the expected throughput benefits of network coding are not large, we expect to be able to achieve them using simpler algorithms. We expand on this point in the following.

■ 2.2.2 Robustness and Adaptability

The most compelling benefits of network coding might be in terms of robustness and adaptability. Intuitively, we can think that network coding, similar to traditional coding, takes information packets and produces encoded packets, where each encoded packet is equally important. Provided we receive a sufficient number of encoded packets, no matter which, we are able to decode. The new twist that network coding brings, is that linear combining is performed opportunistically all over the network, not just at the source node, and thus it is well suited for the (typical) cases where nodes only have incomplete information about the global network state.

Consider the scenario in Figure 2-3 where the basestation S is broadcasting packets to A and B. Assume that A and B may go into sleep mode (or may move out of range) at

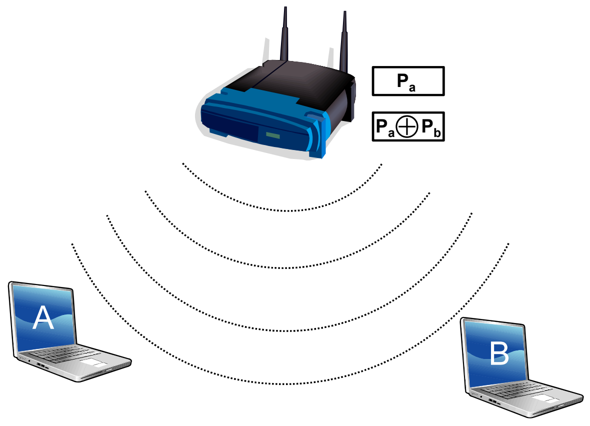


Figure 2-3: A simple scenario showing how network coding improves robustness. The access point can keep transmitting linear combinations of P_a and P_b until both A and B decode, simplifying network operation.

random and without notifying the base station S. If the base station S broadcasts P_a (or P_b), the transmission might be completely wasted, since the intended destination might not be able to receive. However, if the base station broadcasts $P_a \oplus P_b$, or more generally, random linear combinations of the two packets, each transmission will bring new information to all active nodes.

Simplified Content Distribution

The previous example also applies in a general network setting. Consider a Bittorrent network, where a group of nodes want to download a file. The file is split into $O(n)$ packets, and for simplicity, lets say there are n nodes interested in downloading the file. This is related to the classic coupon collector problem [30] in theoretical literature. With a centralized design, the optimal protocol can distribute the entire file to all nodes in $\theta(n)$ rounds. A decentralized approach, however, would need $\theta(n \log(n))$ rounds. The extra $\log(n)$ factor is due to the fact that a specific message may become rare, i.e., hard to find. Instead, if nodes use network coding and transmit linear combinations of packets, all packets are equal and it is sufficient for each node to receive any n coded packets. Hence, with network coding we can distribute the entire file in $\theta(n)$ rounds. The key point here is how network coding simplifies network operation, achieving the optimal performance of a centralized approach using a simple decentralized algorithm.

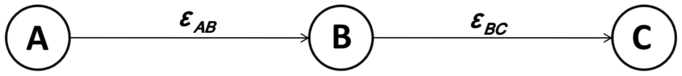


Figure 2-4: Network Coding combats packet loss. A **wishes to send packets to** C **through the router** B**. The links** AB **and** BC **have packet loss probabilities of** ε_{AB} **and** ε_{BC} **respectively. With network coding we can achieve a throughput of** $\min\{(1-\varepsilon_{AB}),(1-\varepsilon_{BC})\}$**, while an ARQ or source coding approach achieves** $(1-\varepsilon_{AB})(1-\varepsilon_{BC})$**.**

Combating Packet Loss

Network coding optimally combats packet loss. Consider the example in Figure 2-4, where the source A intends to transmit packets to destination C with the help of router B. The links are not perfect, link AB drops packets with probability ε_{AB} and link BC drops packets with probability ε_{BC}. Using end-to-end FEC (like fountain codes [13]) or an ARQ protocol like TCP, the source can achieve a maximum throughput of $(1-\varepsilon_{AB})(1-\varepsilon_{BC})$ (assuming each link can transmit one packet per second). Intuitively, the throughput is the probability that a transmitted packet is not lost on either of the two links before it reaches the destination.

Network coding improves throughput in this case by decoupling the links. If we allow the router B to use network coding, i.e., send random linear combinations of the packets it receives from A, we can achieve an end-to-end throughput of $\min\{(1-\varepsilon_{AB}),(1-\varepsilon_{BC})\}$ which is higher than the FEC or ARQ based approaches. The reason is that instead of a lost packet having to be retransmitted all the way from the source, all router B has to ensure is that it delivers a sufficient number of linear combinations for the destination to decode. This scheme in its full generality can be applied over an arbitrary network topology, and with diverse traffic loads (multicasting, unicasting, broadcasting, etc.) [99, 90].

■ 2.3 Subsequent Work on Network Coding

Network coding has been an active area of research since its inception. In this section, we discuss prior work in network coding relevant to this dissertation. Specific differences with different components of the dissertation are discussed in the corresponding chapter.

As discussed before, the basic notion of network coding, i.e., of performing coding operations on the contents of packets throughout the network, is attributed to Ahlswede et al. [2]. This was quickly followed by other work, by Li et al. [85], and by Koetter and Medard [77], that showed that codes with a simple linear structure were sufficient to achieve capacity in the multicast problem. This result put structure on the codes and paved the way for subsequent practical capacity achieving network code designs.

The subsequent growth in network coding was explosive. Practical capacity achieving codes were quickly proposed by Jaggi et al. [63], Ho et al. [58], and Fragouli and Soljanin [39]. A wide variety of applications including network management [57], network tomography [37], overlay networks [47, 64, 138] and wireless networks [49, 73, 114, 135, 134] were studied. The capacity of random networks [108] and undirected networks [87, 88] were investigated, and security aspects were evaluated [10, 14, 34, 56, 61]. Also extensions of network coding to non-multicast problems were proposed [29, 94, 83, 113] and further code constructions based on convolutional codes and other notions were proposed [22, 31, 32, 38, 54, 81]. On the systems side, network coding has been adopted as a core technique in Microsoft's Avalanche project [47], a replacement for content distribution systems like Bittorrent.

Lun et al. [90] analyzed network coding from a theoretical networking perspective. They present a theoretical network coding framework which introduces the notion of cost (in terms of number of transmissions). They proposed minimum cost network coding algorithms, similar to shortest path routing algorithms for non network coded systems. Another significant thread of work concerns network coding and security. Network coding is vulnerable to corruption attacks, where a rogue router mixes corrupt data with correct packets, rendering subsequent decoding erroneous. Ho et al. [56] showed how such modifications can be detected. Subsequently, Jaggi et al. [62] designed polynomial time network codes which were optimally resistant to such jamming attacks, allowing destinations to recover original packets even if they were mixed up with corrupted data. Koetter et al. [76] studied the same problem from a geometric perspective, proposing elegant Reed-Solomon style codes which worked on vector spaces to combat adversarial errors.

■ 2.4 Wireless Mesh Networks

A wireless mesh network is a network of wireless routers connected to each other via multi-hop wireless links. Typically, they are self-organized and adaptive, using distributed algorithms to maintain mesh connectivity. A packet on a mesh network travels multiple

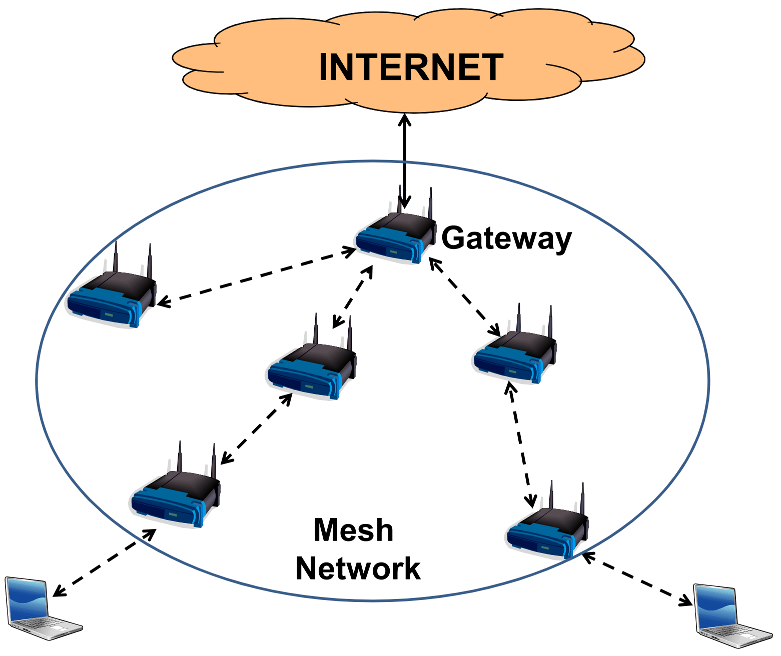


Figure 2-5: Typical Mesh Architecture. Routers form multi-hop wireless paths leading to the gateway which is connected to the Internet. Clients connect to the closest wireless router. Dashed lines represent wireless links and solid lines represent wired links.

wireless links before reaching a gateway node, which is connected to the wired Internet. The major appeal of mesh networks is that because of their multi-hop nature, they can achieve the same coverage as single-hop access point based networks, either with much lower transmission power or significantly lower deployment costs (since wiring every access point is often expensive and hard to maintain). Figure 2-5 shows the typical architecture of mesh networks.

The wireless routers in a mesh network form the backbone for clients. Though, clients can also work as routers and help in forwarding packets, typically they are end-user nodes with simpler wireless hardware and software. Because of their attractive properties: low up-front costs, easy network maintenance, robustness, bigger coverage area etc., a number of research and commercial mesh networks have been deployed [1, 60], with the goal of offering cheap and moderately fast Internet connectivity. But mesh networks figure to be even more ubiquitous in the future, promising completely unwired homes, high throughput city-wide Internet connectivity and fast, self-forming local ad-hoc networks of personal devices, among other things.

2.4.1 Physical Layer

The physical layer of most mesh networks have been built using standard off-the-shelf 802.11 hardware. These radios are able to support multiple transmission rates by a combination of different modulation and coding rates. The original 802.11 standard [24] specified Direct Sequence Spread Spectrum radios that operate at 1 megabit in the 2.4 gigahertz frequency range. 802.11b added additional higher bit-rates, and 802.11g added bit-rates that used Orthogonal Frequency Division Multiplexing (OFDM). 802.11a allows use of frequencies at 5.8 gigahertz using only the OFDM bit-rates.

Prior work [11] has looked at providing adaptive error resilience using bit-rate adaptation algorithms. Complementary work includes channel selection algorithms [74] to balance load and avoid inter-cell interference. Recent research has also followed a parallel thread of investigating new hardware level techniques such as MIMO smart antennas [124], interference cancellation [124] and ultra-wide band (UWB) [6] to support high throughput. Although some of these physical-layer techniques have already been deployed in wireless access points, it is a more challenging problem to adapt these techniques for mesh networks. For example, mesh networking among multiple nodes introduces complicated interactions between links, making the system model much more complicated than that of a conventional MIMO system in wireless LANs or cellular networks.

2.4.2 Link Layer

The MAC layer for wireless mesh networks is harder to design than ones for conventional wireless access point based networks. Present MAC protocols for mesh networks are based on the standard IEEE 802.11 MAC which was designed for access points. The protocol is based on Carrier Sense (CS) with collision avoidance. Nodes perform carrier sense to check if a transmission is active in the medium, and if not, they go ahead and transmit. They also expect to receive a synchronous Ack from the receiver, and if they don't, they assume that the packet has been lost (either due to a bad channel or collision from an interfering transmission) and back off and retransmit. A large body of work has investigated adapting the CSMA/CA protocol for mesh networks. Typical research [9] in this space has included adjusting MAC parameters such as contention window size, backoff windows, carrier sense thresholds etc. However, all these solutions still achieve relatively low throughput since the carrier sense mechanism is not very adept at either detecting hidden terminals, or opportunities for concurrent transmissions.

Recent research has proposed a number of new MAC protocols. CMAPS [127] proposes a MAC protocol which builds a distributed map of possible concurrent transmissions based on empirical, online packet loss measurements. Physical layer techniques such as interference cancellation [48, 52] have been proposed to disentangle collisions into their constituent packets. Such techniques enable higher concurrency, but building a practical MAC protocol based on these physical layer techniques still remains an open problem. Prior work [23] has also examined systems equipped with directional antennae. This capability reduces exposed terminals assuming perfect antenna beamforming. However, due to directional transmissions, more hidden nodes are produced. These schemes also face other difficulties such as cost, system complexity, and practicality of fast steerable directional antennas. Prior work [66] has also looked at MAC protocols which employ power control. This reduces exposed nodes, especially in a dense network, and thus improves the spectrum spatial reuse factor in mesh networks. However, the issue of hidden nodes may become worse because a lower transmission power level reduces the possibility of detecting a potential interfering node.

■ 2.4.3 Network Layer

Routing in mesh networks is complex due to the subtle interactions between adjacent links. Routing has been an active area of research and a large number of protocols have been proposed which we review below.

Routing Protocols with Various Performance Metrics

These are link-state routing protocols which differ in the way they compute the link metric. Typical metrics include hop count, expected transmission count (ETX) [27], per-hop RTT etc. The expected transmission count (ETX) for a link is defined as the inverse of the packet delivery probability on the link. The ETX for a path is the sum of the ETX values for the constituent links. The routing protocols then compute the shortest paths between all pairs of nodes according to the corresponding metric. Prior work [105] has shown that the ETX metric performs best for stationary mesh networks.

Multi-path Routing

The main objectives of using multi-path routing are to perform better load balancing and to provide high fault tolerance. When a link is broken on a path due to a bad channel etc., another path in the set of existing paths can be used. Thus, without waiting to set up a new routing path, the end-to-end delay, throughput, and fault tolerance can be improved. However, the improvement depends on the availability of node disjoint routes between source and destination. Another drawback of multi-path routing is its complexity.

Ganesan's braided multi-path routing [45] identifies multiple routes, using one as a primary and switching if the primary fails. Opportunistic Multipath Scheduling (OMS) [17] splits traffic over multiple paths, adaptively favoring paths that provide low delay or high throughput. Tsirigos and Haas [125] propose sending erasure-coded fragments of each packet over disjoint paths in a mobile ad-hoc network, in order to tolerate loss of some fragments due to fading or node movement.

Opportunistic routing protocols such as ExOR and MORE [12, 18] exploit wireless spatial diversity to perform opportunistic multi-path routing. In these schemes, the choice of the next hop is made after the packet transmission. ExOR [12] chooses this next hop based on expected distance from the destination. MORE [18] uses a similar approach, but employs packet-level network coding to remove the co-ordination overhead. These protocols are related to theoretical work on exploiting co-operative diversity in wireless networks [79]. The focus in co-operative diversity is to improve end-to-end reliability, and hence nodes forward every packet they overhear. Thus reliability is high, but there are a lot of wasted transmissions.

Geographic Routing

Compared to topology-based routing schemes, geographic routing schemes forward packets by only using the position information of nodes in the vicinity and the destination node [75]. Thus, topology change has less impact on geographic routing than other routing protocols. Early geographic routing algorithms were a type of single-path greedy routing schemes in which the packet forwarding decision is made based on the location information of the current forwarding node, its neighbors, and the destination node. However, all greedy routing algorithms have a common problem, i.e., delivery is not guaranteed even if a path exists between source and destination. In order to guarantee delivery, planar-graph based geographic routing algorithms [68] have been proposed recently. However, these algorithms usually have much higher communication overhead than the single-path greedy routing algorithms.

■ 2.4.4 Transport Layer

Traditionally, TCP and UDP are used for the transport layer without any modifications. But wireless networks pose a number of challenges which has necessitated research on modified transport protocols [19]. TCP cannot distinguish between congestion and non-congestion losses. But, due to the high BER, non-congestion losses are quite common in wireless. This results in TCP backing off, reducing network throughput. Further, wireless mesh paths can be asymmetric due to asymmetric channels and interference, which affects Ack delivery and consequently TCP performance. A large body of prior work [4, 3, 110, 92] has proposed a variety of mechanisms to improve TCP performance in wireless networks.

CHAPTER 3

COPE: Packet-level Network Coding

This chapter describes the design, implementation and evaluation of COPE, a new forwarding architecture that uses packet-level network coding to significantly improve wireless throughput. COPE inserts a coding shim between the IP and MAC layers, which identifies coding opportunities and benefits from them by forwarding multiple packets in a single transmission.

COPE takes advantage of the broadcast nature of the wireless medium, which for free, allows multiple nearby nodes to overhear a packet when it is broadcast from a node. This has been typically considered harmful, since neighboring nodes have no use for the overheard packets not meant for them and have to simply throw them away, wasting bandwidth. The broadcast nature has been considered to be one of the primary scaling limitations of multi-hop wireless networks [84, 50].

Instead we show how network coding can be used to profitably exploit the broadcast nature to increase network throughput. Due to the broadcast nature, wireless networks exhibit significant redundancy, i.e., there is a large overlap in the information available to the nodes. First, as a packet travels multiple hops, its contents become known to many nodes. Further, wireless broadcast amplifies this redundancy because at each hop, it delivers the same packet to multiple nodes within the transmitter's radio range. COPE exploits this redundancy to compress data, increasing the information flow per transmission, and thus improves the overall network throughput. We demonstrate COPE's operation through a simple example.

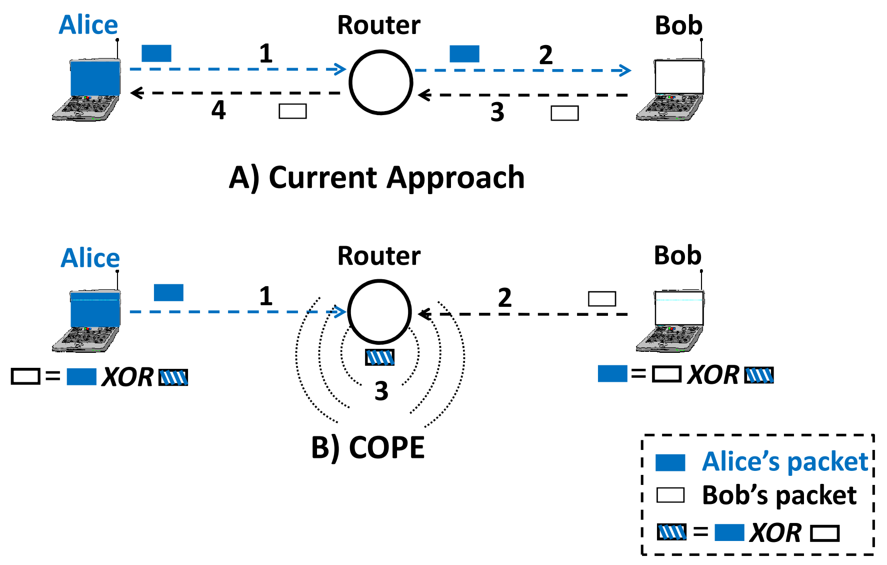


Figure 3-1: A simple example showing how COPE increases throughput. It allows Alice and Bob to exchange a pair of packets using 3 transmissions instead of 4 (numbers on arrows show the order of transmission).

Consider the scenario in Fig. 3-1, where Alice and Bob want to exchange a pair of packets. Since they are not within hearing range of each other, the router in the middle assists them by forwarding their packets to their respective destinations. In the current forwarding architecture, Alice sends her packet to the router, which forwards it to Bob, and Bob sends his packet to the router, which forwards it to Alice. This process requires 4 transmissions.

COPE employs network coding to reduce the number of transmissions needed to deliver the two packets. In COPE, Alice and Bob send their respective packets to the router in separate time slots. The router, instead of forwarding the two packets separately, applies network coding on them. The coding operation is a simple bitwise XOR of the two packets. The router now broadcasts the XOR-ed packet in a single transmission to Alice and Bob. Note that Alice and Bob already have the packets they transmitted initially to the router. Hence, they can obtain each other's packet by XOR-ing the received packet from the router with the packet they initially transmitted. COPE thus delivers the two packets to Alice and Bob in three time slots, as opposed to the four needed by the current architecture. Saved time slots can be used to send new data, increasing the wireless throughput.

In fact, COPE leads to larger bandwidth savings than are apparent from this example. COPE exploits the shared nature of the wireless medium which, for free, broadcasts each packet in a small neighborhood around its path. Each node stores the overheard packets for a short time. It also tells its neighbors which packets it has heard by annotating the packets it sends. When a node transmits a packet, it uses its knowledge of what its neighbors have heard to perform *opportunistic coding*; the node XORs multiple packets and transmits them as a single packet if each intended nexthop has enough information to decode the encoded packet. This extends COPE beyond two flows that traverse the same nodes in reverse order (as in the Alice-and-Bob example), and allows it to XOR more than a pair of packets.

Designing a network coding based forwarding architecture poses several challenges beyond designing an efficient coding and decoding algorithm. First, to code the right set of packets together, nodes have to learn what packets its neighboring nodes have overheard without incurring excessive overhead. Second, since a coded packet is intended for at least two next hops, the node has to ensure reliable delivery of their respective information to *all* the next hops. Getting link-layer feedback on lost/delivered packets from each next hop can get expensive, hence, we have to design an efficient acking and retransmission scheme which take into account the fact that lost information can be coded together again during retransmissions. This chapter tackles these challenges and makes the following contributions:

1. We present the first system architecture for packet-level wireless network coding. This chapter articulates a full-fledged design that integrates seamlessly into the current network stack, works with both TCP and UDP flows, and runs real applications.

2. COPE's coding technique provides a fresh angle for attacking the open theoretical problem of network coding in the presence of multiple unicast sessions. The key insight behind COPE is *local network coding*, where routers locally mix packets such that they can be decoded when the paths of the unicast flows diverge. This ensures that information not intended for a particular part of the network does not get forwarded there and avoids wasting capacity.

3. We present an implementation of the new forwarding architecture in the Linux kernel and the Roofnet platform [1]. The implementation is deployed on a twenty node wireless testbed, creating the first deployment of network coding in a wireless network.

4. We present a detailed evaluation of the performance of COPE in a twenty node wireless testbed. The evaluation reveals interesting interactions between COPE and the physical, routing and higher layer protocols. Our findings can be summarized as follows:

   - Network coding does have practical benefits, and can substantially improve wireless throughput.
   - When the wireless medium is congested and the traffic consists of many random UDP flows, COPE increases the throughput of our testbed by 3-4x.
   - If the traffic does not exercise congestion control (e.g., UDP), COPE's throughput improvement may substantially exceed the expected theoretical coding gain. This additional gain occurs because coding makes a router's queue smaller, reducing the probability that a congested downstream router will drop packets that have already consumed network resources.
   - For a mesh network connected to the Internet via an access point, the throughput improvement observed with COPE varies depending on the ratio between total download and upload traffic traversing the access point, and ranges from 5% to 70%.
   - Hidden terminals create a high collision rate that cannot be masked even with the maximum number of 802.11 retransmissions. In these environments, TCP does not send enough to utilize the medium, and thus does not create coding opportunities. With no hidden terminals, TCP's throughput increases by an average of 38% in our testbed.

From a design point of view, COPE disposes of the point-to-point abstraction of the wireless link and embraces the broadcast nature of the medium. Historically, network designers abstracted the wireless channel as a point-to-point link (a holdover from the design of the wired Internet), and adapted forwarding and routing techniques designed for wired networks for wireless. In contrast, COPE exploits the broadcast property of radios instead of hiding it under an artificial abstraction.

The rest of this chapter proceeds as follows. Section 3.1 describes the high level design of COPE, while Sections 3.2 and 3.3 present the detailed architecture and implementation of COPE. Section 3.4 presents a theoretical analysis of COPE, while Section 3.5 presents a

Term	Definition
Native Packet	A non-encoded packet
Encoded or XOR-ed Packet	A packet that is the XOR of multiple native packets
Nexthops of an Encoded Packet	The set of nexthops for the native packets XOR-ed to generate the encoded packet
Packet Id	A 32-bit hash of the packet's IP source address and IP sequence number
Output Queue	A FIFO queue at each node, where it keeps the packets it needs to forward
Packet Pool	A buffer where a node stores all packets heard in the past T seconds
Coding Gain	The ratio of the number of transmissions required by the current non-coding approach, to the number of transmissions used by COPE to deliver the same set of packets.
Coding+MAC Gain	The expected throughput gain with COPE when an 802.11 MAC is used, and all nodes are backlogged.

Table 3-1: Definitions of terms used in this chapter.

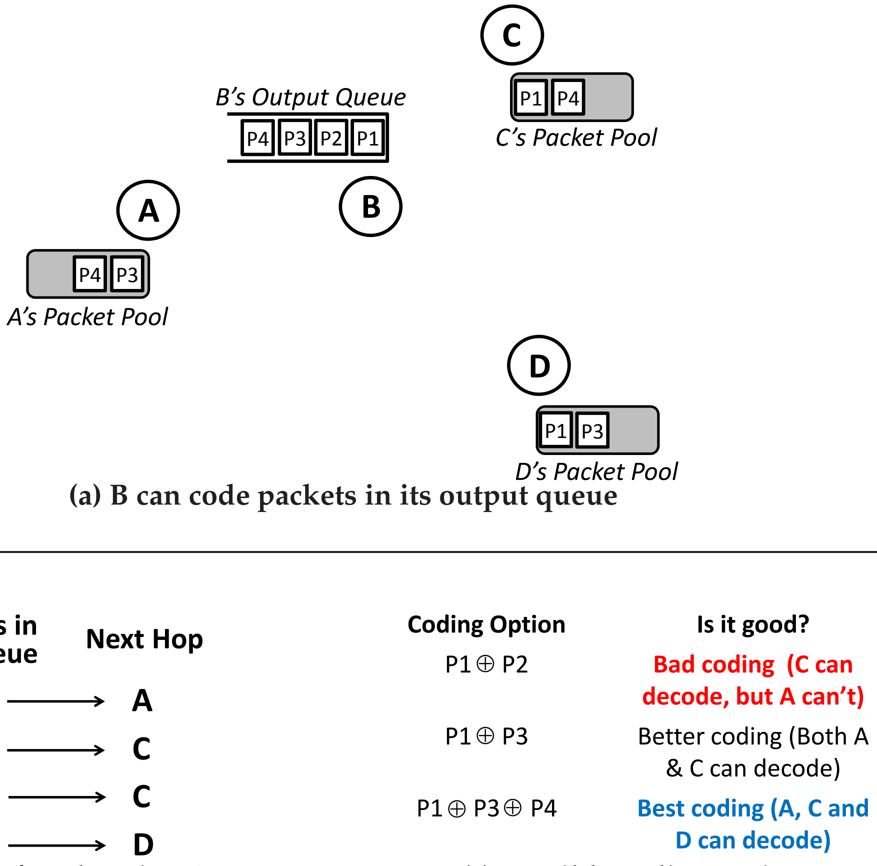
detailed practical system evaluation. Section 3.7 summarizes the take-away lessons and concludes.

■ 3.1 High Level Design of COPE

COPE is a new forwarding architecture for wireless mesh networks built around network coding. It inserts a coding layer between the network and link layers, which detects coding opportunities and exploits them to forward multiple packets in a single transmission. Before delving into details, we refer the reader to Table 3-1, which defines the terms used in the rest of the chapter.

COPE's modular design allows network designers to experiment with a variety of routing protocols, as well as underlying physical layer hardware. COPE can work with a variety of routing protocols, the only condition it imposes is that the forwarding node knows the next hop of each packet in its forwarding queue. Further, some network designers may wish to modify the routing [116], or transport [106] layers to increase coding opportunities and further increase network throughput.

In this section, we describe COPE's high level operational design. COPE has three main components in its operation which can be succinctly summarized as: ***listening, coding*** and



(a) B can code packets in its output queue

(b) Nexthops of packets in B's queue

(c) Possible coding options

Figure 3-2: Example of Opportunistic Coding. Node B has 4 packets in its queue, whose nexthops are listed in (b). Each neighbor of B has stored some packets as depicted in (a). Node B can make a number of coding decisions (as shown in (c)), but should select the last one because it maximizes the number of packets delivered in a single transmission.

learning. The following subsections describe each one of them in detail.

3.1.1 Listening

COPE requires nodes to *listen* and store packets that can be potentially used later for decoding coded packets. There are two sources for such packets:

1. Packets the node itself broadcast. For example, in the Alice-Bob example in Figure. 3-1, Alice and Bob keep a copy of the packets they transmitted to the router. They use these packets later for decoding the coded packet from the router.

2. Packets the node overheard. Since wireless is a broadcast medium, it creates many opportunities for nodes to overhear packets when they are equipped with omni-

directional antennae. Packets are usually addressed to a single next hop, and nodes throw away packets which they overhear but aren't meant for them. In COPE however, nodes are kept in promiscuous mode, and they snoop on all communications over the wireless medium and store the overheard packets for a limited period T. The value of T should be larger than the maximum one-way latency in the network (the default is $T = 0.5s$).

3.1.2 Coding

COPE allows nodes to *code* packets together to maximize throughput. Each node maintains a FIFO forwarding queue, where it keeps the packets that it is supposed to transmit to the next hop. When the wireless channel signals an opportunity to send a packet, the node picks the packet at the head of the queue, and checks whether the packet can be coded with other packets in the queue. The key question is: what packets to code together?

The choices a node faces are best illustrated with an example. In Fig. 3-2(a), node B has 4 packets in its forwarding queue p_1, p_2, p_3, and p_4. Its neighbors have overheard some of these packets, which they have stored in their respective *Packet Pools*. The table in Fig 3-2(b) shows the nexthop of each packet in B's queue. When the MAC permits B to transmit, B takes packet p_1 from the head of the queue and attempts to code it with other packets from the forwarding queue. Note that both coding and decoding are simple XOR operations. Assuming that B knows which packets each neighbor has, it has a few coding options as shown in Fig. 3-2(c)

1. Send $p_1 \oplus p_2$. Since node C has p_1 in store, it could XOR p_1 with $p_1 \oplus p_2$ to obtain the native packet sent to it, i.e., p_2. However, node A does not have p_2, and hence cannot decode the XOR-ed packet. Thus, sending $p_1 \oplus p_2$ allows only one neighbor to benefit from the transmission.

2. Send $p_1 \oplus p_3$. Both next-hops, C and A can decode and obtain their intended packets from a single transmission, since they have the other packet in their *Packet Pool*. This is a better option than the first one, since two neighbors benefit from a single transmission.

3. Send $p_1 \oplus p_3 \oplus p_4$. All three next hops, A, C and D, will be able to recover the packet meant for them, since they have the other two packets in their *Packet Pool*. For exam-

ple, node A can XOR the received coded packet with p_3 and p_4 from its *Packet Pool* to recover p_1, the packet meant for it. This is the best option, since three neighbors benefit from a single transmission.

The example gives us intuition on which option to pick. To maximize throughput a node should aim to *maximize the number of native packets coded in a single transmission, with the constraint that each intended nexthop has enough information to decode the packet meant for it.* This intuition is formalized in the following rule:

> To transmit n packets, $p_1, ..., p_n$, to n nexthops, $r_1, ..., r_n$, a node can XOR the n packets together only if each next-hop r_i has all $n-1$ packets p_j for $j \neq i$.

The rule also starkly illustrates the difference between theoretical network coding [77] and COPE's coding algorithm. Packets from multiple unicast flows may get encoded together at some intermediate hop. But their paths may diverge at the nexthop, at which point they need to be decoded. If not, unneeded data will be forwarded to areas where there is no interested receiver, wasting bandwidth. Hence COPE's coding algorithm ensures that all nexthops of an encoded packet can decode their corresponding packet. Traditional network coding theory advocates randomly combining all packets, which renders them undecodable at the next hop. Therefore traditional network coding has to restrict coding only to packets belonging to the same flow. COPE extends network coding to multiple unicast flows.

■ 3.1.3 Learning

To implement the coding algorithm described above, nodes in COPE need to know what packets their neighbors have in their *Packet Pools*. COPE achieves this through two mechanisms:

1. **Reception Reports**: Nodes broadcast *reception reports* to tell their neighbors which packets they have stored in their *Packet Pool*. Reception reports are sent by annotating the data packets the node transmits. A node that has no data packets to transmit periodically sends the reception reports in special control packets.

2. **Guessing**: If a node hasn't received a reception report for a particular packet in its forwarding queue, it guesses whether a neighbor has it. To guess intelligently, we

leverage the routing computation. Wireless routing protocols compute the delivery probability between every pair of nodes and use it to identify good paths. For e.g., the ETX metric [27] periodically computes the delivery probabilities and assigns to each link a weight equal to *1/(delivery probability)*. These weights are broadcast to all nodes in the network and used by a link-state routing protocol to compute shortest paths. We leverage these probabilities for guessing. In the absence of deterministic information, COPE estimates the probability that a particular neighbor has a packet as the delivery probability of the link between the packet's previous hop and the neighbor.

Guessing allows COPE to operate efficiently when reception reports get lost or are late. For example, at times of severe congestion, reception reports may get lost in collisions, while at times of light traffic, they may not arrive before a node has to make a coding decision on a particular packet. COPE is not handcuffed in such situations, since the guessing method gives a fairly accurate picture of neighbor state. Occasionally, a node may make an incorrect guess, which causes the coded packet to be undecodable at some nexthop. In this case, the relevant native packet is retransmitted, potentially encoded with a new set of native packets.

■ 3.2 System Architecture

In this section, we aim to make practical, the high-level overview in Section. 3.1. As mentioned before, COPE makes no changes to the network layer above or the link layer below. The COPE layer has four major components: *coding/decoding*, *reliability*, *listening* and *guessing* modules. We describe each of them in detail below.

■ 3.2.1 Design Goals

To build a practical system, we have to make a few design decisions.

1. **Transparency:** COPE does not expect or make any changes in the layers above or below. A primary design goal is to plug in seamlessly into any network stack. Specifically, this means that though there are minor benefits to be had from modifying the routing or physical layers [116], COPE chooses not to do so in the interests of modularity and simplicity.

2. **No extra delay:** As we will see below, COPE can increase throughput by delaying packets in its queues. But to avoid unintended consequences on applications expecting a certain best-effort behavior from the network, COPE never intentionally delays a packet.

3. **Reliable delivery:** COPE strives to guarantee the same level of reliability to every packet which is part of a coded packet, as if the packets were transmitted separately.

In the following sections we will see how these design goals affect the design of COPE.

■ 3.2.2 Coding/Decoding Module

The coding/decoding module handles the job of identifying opportunities to code multiple packets together and send them in a single transmission. It also decodes coded packets on the receiving side. It maintains a FIFO output queue, where received uncoded and decoded packets are added. When the MAC layer signals an opportunity to send, if possible, it picks packets from the output queue to code together, and transmits the packet. The primary question for the coding module is: *What packets to code together from the output queue?* Recall from Section 3.1 that to code n packets together, a node has to ensure that each of the n next-hops has the remaining $n-1$ packets apart from the one intended for it. But the choice of packets is more nuanced due to three factors: *design goals, probabilistic neighbor state and variable packet sizes.* We discuss each of them below.

Code opportunistically:
One of the design goals is to not delay packets intentionally. Hence, in COPE, whenever the wireless channel is available, the node takes the packet at the head of its output queue, checks which other packets in the queue may be encoded with this packet, XORs those packets together, and broadcasts the XOR-ed version. If there are no encoding opportunities, our node does not delay the packet to wait for the arrival of another packet which could be coded together with the packet at the head of the queue. COPE therefore lets the node opportunistically overload each transmission with additional information when possible, but does not wait for additional codable packets to arrive.

Code to maximize decoding probability:

COPE tries to ensure that each neighbor to whom a packet is headed has a high probability of decoding its native packet. In particular, suppose the node encodes n packets together. Let the probability that a nexthop has heard packet i be P_i. Then, the probability, P_D, that it can decode its native packet is equal to the probability that it has heard all of the $n-1$ native packets XOR-ed with its own, i.e.,

$$P_D = P_1 \times P_2 \times \ldots \times P_{n-1}.$$

Consider an intermediate step while searching for coding candidates. We have already decided to XOR $n-1$ packets together, and are considering XOR-ing the n^{th} packet with them. The coding algorithm now checks that, for each of the n nexthops, the decoding probability P_D, after XOR-ing the n^{th} packet with the rest stays greater than a threshold G (the default value $G = 0.8$). If the above conditions are met, each nexthop can decode its packet with at least probability G. Finally, we note that for fairness we iterate over the set of neighbors according to a random permutation.

Thus, for each packet in its output queue, we estimate the probability that each of its neighbors has already heard the packet. The coding/decoding module consults the **Learning Module** for this probability. The probability value can be computed from two sources:

1. *Reception Reports:* When the learning module has heard a reception report from a neighbor about a particular packet, or if that neighbor was the previous hop of that packet, then the guessing module can be certain that the neighbor has the packet. Hence, in this case the probability value is 1.0.

2. *Guessing:* When the learning module doesn't have deterministic information, it guesses. For this, it leverages the delivery probabilities computed by the routing protocol. It estimates the probability the neighbor has the packet as the delivery probability between the packet's previous hop and that neighbor. It reports this delivery probability to the coding module, which uses them to ensure that encoded packets are decodable by all of their nexthops with high probability.

An interesting consequence of the above condition is that COPE will *never code together two or more packets headed to the same nexthop*, since the nexthop will not be able to decode them. Hence, while coding, we only need to consider packets headed to different nexthops.

Code packets of similar lengths

COPE *gives preference to XOR-ing packets of similar lengths*, because XOR-ing small packets with larger ones reduces bandwidth savings. Empirical studies show that the packet-size distribution in the Internet is bimodal with peaks at 40 and 1500 bytes [119]. We can therefore limit the overhead of searching for packets with the right sizes by distinguishing between small and large packets. We might still have to XOR packets of different sizes. In this case, the shorter packets are padded with zeroes. The receiving node can easily remove the padding by checking the packet-size field in the IP header of each native packet. COPE therefore maintains two virtual queues per neighbor; one for small packets and another for large packets (The default setting uses a threshold of 100 bytes). When a new packet is added to the output queue, an entry is added to the appropriate virtual queue based on the packet's nexthop and size.

Data Structures and Coding Algorithm

Formally, each node maintains the following data structures.

- Each node has a FIFO queue of packets to be forwarded, which we call *the output queue*.
- For each neighbor, the node maintains two *per-neighbor virtual queues*, one for small packets (e.g., smaller than 100 bytes), and the other for large packets. The virtual queues for a neighbor A contain pointers to the packets in the output queue whose nexthop is A.
- Additionally, the node keeps a hash table, *packet info*, that is keyed on packet-id. For each packet in the output queue, the table indicates the probability of each neighbor having that packet.

Whenever the MAC signals a sending opportunity, the node executes the procedure illustrated in Alg. 1.

Searching for appropriate packets to code is efficient due to the maintenance of virtual queues. When making coding decisions, COPE first dequeues the packet at the head of the FIFO output queue, and determines if it is a small or a large packet. Depending on the size, it looks at the appropriate virtual queues. For example, if the packet dequeued is a small packet, COPE first looks at the virtual queues for small packets. COPE looks only at the heads of the virtual queues to limit packet reordering. After exhausting the virtual queues of a particular size, the algorithm then looks at the heads of virtual queues for packets of the other size. Thus for finding appropriate packets to code, COPE has to look at $2M$ packets in the worst case, where M is the number of neighbors of a node.

Another concern is *packet reordering*. We would like to limit reordering packets from the same flow because TCP mistakes it as a congestion signal. Thus, we always consider packets according to their order in the output queue. Still, reordering may occur because we prefer to code packets of the same size. In practice, this reordering is quite limited because most data packets in a TCP flow are large enough to be queued in the large-packet queue, and thus be considered in order. We will see in Section 3.2.3, however, that reordering might arise from other reasons, particularly the need to retransmit a packet that has been lost due to a mistake in guessing what a neighbor can decode. Thus, we choose to deal with any reordering that might happen inside the network at the receiver. COPE has a module that puts TCP packets in order before delivering them to the transport layer as explained in Section. 3.2.3.

Decoding Algorithm

Packet decoding is simple. When a node receives an encoded packet consisting of n native packets, the node goes through the ids of the native packets one by one. In this set, there are $n-1$ packets apart from the one addressed to it. It attempts to retrieve these $n-1$ packets from the *Listening Module*, and XORs the $n-1$ packets with the received encoded packet to retrieve the native packet meant for it. If the listening module doesn't have the $n-1$ packets, the coded packet is dropped. Note that some optimizations are possible, where a presently undecodable packet may get decoded later when more information is available, but for reasons of simplicity, COPE chooses to drop the packet.

■ 3.2.3 Reliability

COPE strives to provide the same level of reliability to both coded as well as native packets. We will first discuss how 802.11 wireless systems provide link-layer reliability and then show why that isn't sufficient for COPE. We will then explain our solution and discuss

1 Coding Procedure

```
Pick packet $p$ at the head of the output queue.
Natives = $\{p\}$
Nexthops = $\{nexthop(p)\}$
**if** $size(p) > 100$ bytes **then**
   which_queue = 1
**else**
   which_queue = 0
**end if**
**for** Neighbor $i = 1$ to $M$ **do**
   Pick packet $p_i$, the head of virtual queue $Q(i, which\_queue)$
   **if** $\forall n \in$ Nexthops $\cup\{i\}$, Pr[$n$ can decode $p \oplus p_i$] $\geq G$ **then**
      $p = p \oplus p_i$
      Natives = Natives $\cup\{p_i\}$
      Nexthops = Nexthops $\cup\{i\}$
   **end if**
**end for**
which_queue = !which_queue
**for** Neighbor $i = 1$ to $M$ **do**
   Pick packet $p_i$, the head of virtual queue $Q(i, which\_queue)$
   **if** $\forall n \in$ Nexthops $\cup\{i\}$, Pr[$n$ can decode $p \oplus p_i$] $\geq G$ **then**
      $p = p \oplus p_i$
      Natives = Natives $\cup\{p_i\}$
      Nexthops = Nexthops $\cup\{i\}$
   **end if**
**end for**
return $p$
```

other issues.

How 802.11 provides reliability?

The 802.11 MAC has two modes: unicast and broadcast. Specifically, in the 802.11 unicast mode, packets are immediately ack-ed by their intended nexthops. The 802.11 protocol ensures reliability by retransmitting the packet at the MAC layer for a fixed number of times until a synchronous Ack is received. Lack of an Ack is interpreted as a collision signal, to which the sender reacts by backing off exponentially, thereby allowing multiple nodes to share the medium.

In contrast, 802.11 broadcast lacks both reliability and backoff. A broadcast packet has many intended receivers, and it is unclear who should ack. In the absence of the Acks, the broadcast mode offers no retransmissions and consequently very low reliability. Additionally, a broadcast source cannot detect collisions, and thus does not back off. If multiple backlogged nodes share the broadcast channel, and each of them continues sending at the highest rate, the resulting throughput is therefore very poor, due to high collision rates.

Problem

COPE broadcasts encoded packets to their next hops, the natural approach would be to use broadcast. But as discussed before, the broadcast mode in 802.11 lacks both reliability and backoff. Ideally one would design a new MAC suitable for broadcast communication, but we are interested in an implementation of COPE that can be deployed in the near future using off-the-shelf 802.11 products.

A potential solution would be to use the 802.11 unicast mode, which provides synchronous acks, retransmissions and backoff. While this is sufficient for uncoded packets, a coded packet should be delivered reliably to two or more next hops. The 802.11 unicast mode can ensure reliability to only one of them. Hence we need an alternative mechanism to ensure reliability.

COPE solves this problem by two mechanisms: *Pseudo-broadcast* and *Asynchronous Acks and Retransmissions*. We discuss both of them below.

Solution

Pseudo-broadcast: The first part of the solution is *pseudo-broadcast*, which piggybacks on 802.11 unicast and benefits from its reliability and backoff mechanism. Pseudo-broadcast unicasts packets that are meant for broadcast. The link-layer destination field is set to the MAC address of one of the intended recipients. An XOR-header is added after the link-layer header, listing all nexthops of the packet. Since all nodes are set in the promiscuous mode, they can overhear packets not addressed to them. When a node receives a packet with a MAC address different from its own, it checks the XOR-header to see if it is a nexthop. If so, it processes the packet further, else it stores the packet in a buffer as an opportunistically received packet. As all packets are sent using 802.11 unicast, the MAC can detect collisions and backoff properly.

Pseudo-broadcast is also more reliable than simple broadcast. The packet is retransmitted multiple times until its designated MAC receiver receives the packet and acks it, or the number of retries is exceeded. A desirable side effect of these retransmissions is that nodes that are promiscuously listening to this packet have more opportunities to hear it.

Pseudo-broadcast, however, does not completely solve the reliability problem, which we address in the next section.

Asynchronous Acks and Retransmissions: Encoded packets require all nexthops to acknowledge the receipt of the associated native packet for two reasons. First, encoded packets are headed to multiple nexthops, but the sender gets synchronous MAC-layer acks only from the nexthop that is set as the link layer destination of the packet (as explained in the previous section). There is still a probability of loss to the other nexthops from whom it does not get synchronous acks. Second, COPE may optimistically guess that a nexthop has enough information to decode an XOR-ed packet, when it actually does not.

The standard solution to wireless losses is to mask error-induced drops by recovering lost packets locally through acknowledgments and retransmissions [3, 102]. COPE too addresses this problem using local retransmissions; the sender expects the nexthops of an XOR-ed packet to decode the XOR-ed packet, obtain their native packet, and ack it. If any of the native packets is not ack-ed within a certain interval, the packet is retransmitted, potentially encoded with another set of native packets.

How should we implement these hop-by-hop Acks? For non-coded packets, we simply leverage the 802.11 synchronous Acks. Unfortunately, extending this synchronous Ack approach to coded packets is highly inefficient, as the overhead incurred from sending each Ack in its own packet with the necessary IP and WiFi headers would be excessive. Thus, in COPE, encoded packets are ack-ed asynchronously.

When a node sends an encoded packet, it schedules a retransmission event for each of the native packets in the encoded packet. If any of these packets is not ack-ed within T_a seconds, the packet is inserted at the head of the output queue and retransmitted. (T_a is slightly larger than the round trip time of a single link.) Retransmitted packets may get encoded with other packets according to the scheme in Section. 3.2.2.

A nexthop that receives an encoded packet decodes it to obtain its native packet, and immediately schedules an Ack event. Before transmitting a packet, the node checks its pending Ack events and incorporates the pending Acks in the COPE header. If the node has no data packets to transmit, it sends the Acks in periodic control packets–the same control packets used to send reception reports.

Transport Layer Reliability

Asynchronous acks can cause packet reordering, which may be confused by TCP as a sign of congestion. Thus, COPE has an *ordering agent*, which ensures that TCP packets are delivered in order. The agent ignores all packets whose final IP destinations differ from the current node, as well as non-TCP packets. These packets are immediately passed to the next processing stage. For each TCP flow ending at the host, the agent maintains a packet buffer and records the last TCP sequence number passed on to the network stack. Incoming packets that do not produce a hole in the TCP sequence stream are immediately dispatched to the transport layer, after updating the sequence number state. Otherwise, they are withheld in the buffer till the gap in the sequence numbers is filled, or until a timer expires.

■ 3.2.4 Listening Module

The goal of the listening module is to store packets which might be useful for decoding coded packets later. There are two classes of packets:

1. Transmitted packets: These are packets which the node itself transmitted previously.

2. Overheard packets: The node keeps its wireless card in promiscuous mode and overhears and stores packets transmitted from other nodes.

The packets have to be kept as long as they are useful for decoding. This duration is hard to determine exactly, but it is upper-bounded by the maximum end-to-end delay in the mesh network. To simplify design, we keep these packets for 500 ms (which is larger than the end-to-end delay of typical mesh network deployments). Finally, when the listening module overhears a packet, it also broadcasts a *reception report* informing its neighbors that it has this packet.

The packets are kept in a hash table keyed by a unique id. We use the source IP address and the IPID combination as the unique identifier for the hash table. Further, the listening module exports a `lookup` call API to the coding module. The function takes the source IP address and IPID as arguments and returns the corresponding packet if the listening module has it.

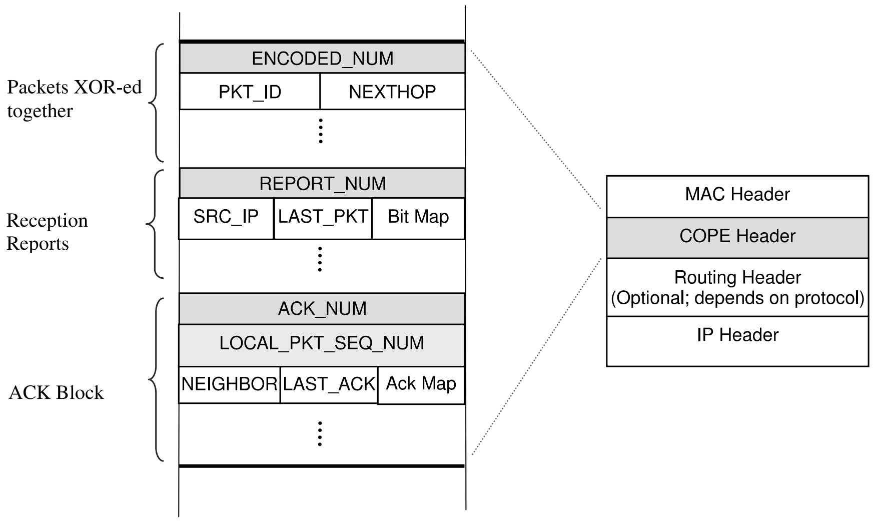


Figure 3-3: COPE Header. The first block identifies the native packets XOR-ed and their next hops. The second block contains reception reports. Each report identifies a source, the last IP sequence number received from that source, and a bit-map of most recent packets seen from that source. The third block contains asynchronous acks. Each entry identifies a neighbor, an end point for the Ack map, and a bit-map of ack-ed packets.

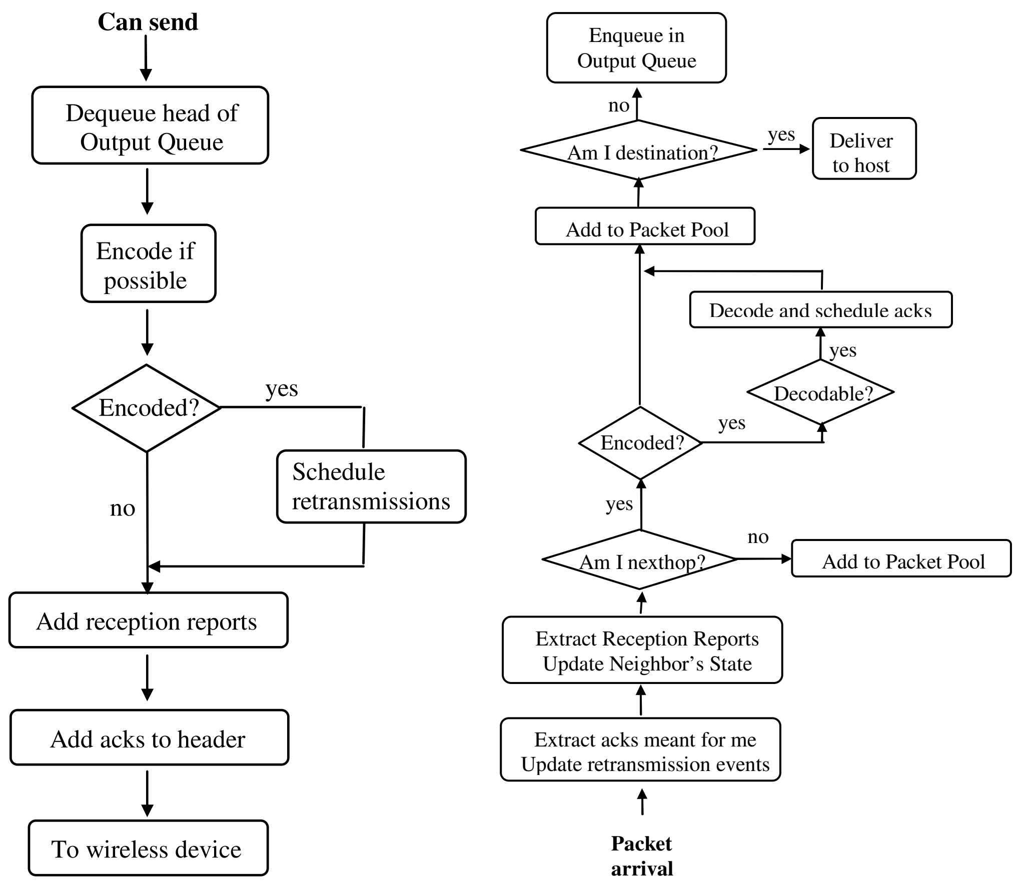
■ 3.2.5 Learning Module

The function of the learning module is to keep track of what packets a node's neighbors have stored in their listening module. This information is necessary to make the right coding decision as discussed in Section 3.2.2. The learning module uses two mechanisms to keep track of this neighbor state:

1. Deterministic information: If a neighboring node sent a reception report about a packet or was the previous hop on that packet's route, then the learning module is certain that the neighboring node has that packet.

2. Probabilistic information: When the learning module doesn't have deterministic information, it guesses. For this, it leverages the delivery probabilities computed by the routing protocol. It estimates the probability the neighbor has the packet as the delivery probability between the packet's previous hop and that neighbor.

The learning module also exposes a lookup API to the coding module. It takes as input the packet id (source IP+IPID), and the neighbor identifier (MAC address) and returns the probability that the neighbor has that particular packet. It keeps this information in a two-level hash table keyed first by the packet id and then by the neighbor id.



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

Figure 3-4: Flow chart for our COPE Implementation.

■ 3.3 Implementation Details

COPE adds special packet headers and alters the control flow of the router to code and decode packets. This section describes both parts.

■ 3.3.1 Packet Format

COPE inserts a variable-length coding header in each packet, as shown in Fig. 3-3. If the routing protocol has its own header (e.g., Srcr [27]), COPE's header sits between the routing and the MAC headers. Otherwise, it sits between the MAC and IP headers. Only the shaded fields in Fig. 3-3 are required in every COPE header. The coding header contains the following 3 blocks.

(a) Ids of the coded native packets: The first block records meta-data to enable packet decoding. It starts with `ENCODED_NUM`, the number of native packets XOR-ed together.

For each native packet, the header lists its `ID`, which is a 32-bit hash of the packet's source IP address and IP sequence number. This is followed by the MAC address of the native packet's `Nexthop`. When a node hears an XOR-ed packet, it checks the list of `Nexthops` to determine whether it is an intended recipient for any of the native packets XOR-ed together, in which case it decodes the packet, and processes it further.

(b) Reception reports: Reception reports constitute the second block in the XOR header, as shown in Fig. 3-3. The block starts with the number of the reports in the packet, `REPORT_NUM`. Each report specifies the source of the reported packets `SRC_IP`. This is followed by the IP sequence number of the last packet heard from that source `Last_PKT`, and a bit-map of recently heard packets. For example, a report of the form {`128.0.1.9, 50, 10000001`} means that the last packet this node has heard from source `128.0.1.9` is packet 50, and it has also heard packets 42 and 49 from that source but none in between. The above representation for reception reports has two advantages: compactness and effectiveness. In particular, the bit-map allows the nodes to report each packet multiple times with minimal overhead. This guards against reception reports being dropped at high congestion.

(c) Expressing asynchronous Acks compactly and robustly: To ensure Ack delivery with minimum overhead, we use cumulative Acks. Since they implicitly repeat Ack information, cumulative Acks are robust against packet drops. Each node maintains a per-neighbor 16-bit counter, called `Neighbor_Seqno_Counter`. Whenever the node sends a packet to that neighbor, the counter is incremented and its value is assigned to the packet as a local sequence number, `Local_PKT_SEQ_NUM`. The two neighbors use this sequence number to identify the packet. Now, a node can use cumulative Acks on a per-neighbor basis. Each coded packet contains an Ack header as shown in Fig. 3-3. The Ack block starts with the number of Ack entries, followed by the packet local sequence number. Each Ack entry starts with a neighbor MAC address. This is followed by a pointer to tell the neighbor where the cumulative Acks stop, and a bit-map indicating previously received and missing packets. For example, an entry of {`A, 50, 01111111`} acks packet 50, as well as the sequence 43-49, from neighbor A. It also shows that packet 42 is still missing. Note that though we use cumulative Acks, we do not guarantee reliability at link layer. In particular, each node retransmits a lost packet a few times (default is 2), and then gives up.

3.3.2 Control Flow

Fig. 3-4 abstracts the architecture of COPE. On the sending side, (shown in Fig. 3-4(a)), whenever the MAC signals an opportunity to send, the node takes the packet at the head of its output queue and hands it to the coding module (Section 3.2.2). If the node can encode multiple native packets in a single XOR-ed version, it has to schedule asynchronous retransmissions. Either way, before the packet can leave the node, pending reception reports and Acks are added.

On the receiving side, (shown in Fig. 3-4(b)), when a packet arrives, the node extracts any Acks sent by this neighbor to the node. It also extracts all reception reports and updates its view of what packets its neighbor stores. Further processing depends on whether the packet is intended for the node. If the node is not a nexthop for the packet, the packet is stored in the Packet Pool. If the node is a nexthop, it then checks if the packet is encoded. If it is, the node tries to decode by XOR-ing the encoded packet with the native packets it stores in its Packet Pool. After decoding, it acks this reception to the previous hop and stores the decoded packet in the Packet Pool. The node now checks if it is the ultimate destination of the packet, if so it hands the packet off to the higher layers of the network stack. If the node is an intermediate hop, it pushes the packet to the output queue. If the received packet is not encoded, the packet is simply stored in the Packet Pool and processed in the same fashion as a decoded packet.

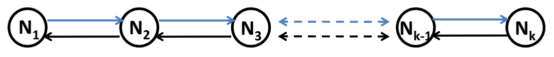
3.4 Understanding COPE's Gains

How beneficial is COPE? Its throughput improvement depends on the existence of coding opportunities, which themselves depend on the traffic patterns. This section provides some insight into the expected throughput increase and the factors affecting it.

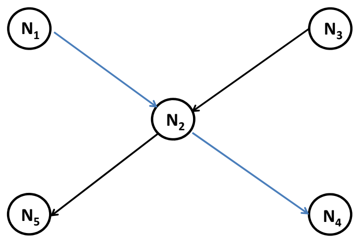
3.4.1 Coding Gain

We define the *coding gain* as the ratio of the number of transmissions required by the current non-coding approach, to the minimum number of transmissions used by COPE to deliver the same set of packets. By definition, this number is greater than or equal to 1.

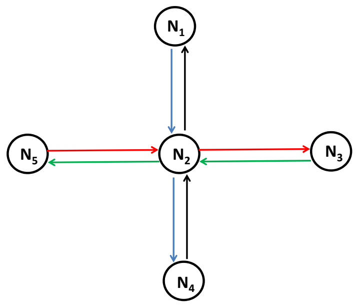
In the Alice-and-Bob experiment, as described in Figure 3-1, COPE reduces the number of transmissions from 4 to 3, thus producing a coding gain of $\frac{4}{3} = 1.33$.



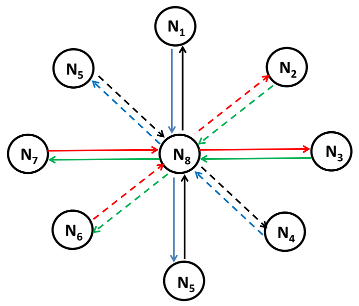
(a) Chain topology; 2 flows in reverse directions.



(b) "X" topology



(c) Cross topology



(d) Wheel topology

Figure 3-5: Simple topologies to understand COPE's Coding and Coding+MAC Gains.

But what is the maximum achievable coding gain, i.e., what is the theoretical capacity of a wireless network that employs COPE? The capacity of general network coding for unicast traffic is still an open question for arbitrary graphs [86, 122]. However, we analyze certain basic topologies that reveal some of the factors affecting COPE's coding gain. Our analysis assumes identical nodes, omni-directional radios, perfect hearing within some radius, and the signal is not heard at all outside this radius, and if a pair of nodes can hear each other the routing will pick the direct link. Additionally, we assume that the flows are infinite and we only consider the steady state.

Theorem 3.1 *In the absence of opportunistic listening, COPE's maximum coding gain is 2, and it is achievable.*

Proof. We first prove the upper bound of 2. Note that if the intermediate node codes N native packets together, these packets have to be to N different next-hops, by the coding rule of Section 3.1. In the absence of opportunistic listening, the only neighbor that has a packet is the previous hop of that packet. Suppose the intermediate hop codes ≥ 2 packets from the same neighbor. All other neighbors must have $\leq N-2$ packets in the encoded packet, which violates the coding rule. As a result, the intermediate hop can code at most one packet from a neighbor. Without opportunistic listening, this is the only native packet in the encoded packet that this neighbor has. Invoking the coding rule, this implies that the intermediate hop can code at most 2 packets together. This implies that the total number of transmissions in the network can at most be halved with coding, for a coding gain of 2.

Indeed, this gain is achievable in the chain of N links in Fig. 3-5(a). This topology is an extension of the Alice-and-Bob example where $N = 2$. The no-coding case requires a total of $2N$ transmissions to deliver a packet from Alice to Bob, and vice-versa. On the other hand, in the presence of coding, each of the $N - 1$ intermediate nodes on the path can transmit information simultaneously to neighbors on either side by coding the two packets traversing in opposite directions, for a total of $N + 1$ transmissions. The coding gain in this case is $\frac{2N}{N+1}$, which tends to 2 as the chain length grows. ■

While we do not know the maximum gain for COPE with opportunistic listening, there do exist topologies where opportunistic listening adds to the power of COPE. For example, consider the "X"-topology shown in Fig. 3-5(b). This is the analogy of the Alice-and-Bob topology, but the two flows travel along link-disjoint paths. COPE without opportunistic listening cannot achieve any gains on this topology. But with opportunistic listening and guessing, the middle node can combine packets traversing in opposite directions, for a coding gain of $\frac{4}{3} = 1.33$. This result is important, because in a real wireless network, there might be only a small number of flows traversing the reverse path of each other à la Alice-and-Bob, but one would expect many flows to intersect at a relay, and thus can be coded together using opportunistic listening and guessing.

The "X" and Alice-and-Bob examples can be combined to further improve the benefits of coding, as in the cross topology of Fig. 3-5(c). Without coding, 8 transmissions are necessary for each flow to send one packet to its destination. However, assuming perfect overhearing (N_1 and n_4 can overhear n_3 and n_5, and vice versa), N_2 can XOR 4 packets in each transmission, thus reducing the number of transmissions from 8 to 5, producing a coding gain of $\frac{8}{5} = 1.6$.

We observe that while this section has focused on theoretical bounds, the gains in practice tend to be lower due to the availability of coding opportunities, packet header overheads, medium losses, etc. However, it is important to note that COPE increases the actual information rate of the medium far above the bit rate, and hence its benefits are sustained even when the medium is fully utilized. This contrasts with other approaches to improving wireless throughput, such as opportunistic routing [12], which utilize the medium better when it is not fully congested, but do not increase its capacity.

■ 3.4.2 Coding+MAC Gain

When we ran experiments with COPE, we were surprised to see that the throughput improvement sometimes greatly exceeded the coding gain for the corresponding topology. It turns out that the interaction between coding and the MAC produces a beneficial side effect that we call the Coding+MAC gain.

The Coding+MAC gain is best explained using the Alice-and-Bob scenario. Because it tries to be fair, the MAC divides the bandwidth equally between the 3 contending nodes: Alice, Bob, and the router. Without coding, however, the router needs to transmit twice as many packets as Alice or Bob. The mismatch between the traffic the router receives from the edge nodes and its MAC-allocated draining rate makes the router a bottleneck; half the packets transmitted by the edge nodes are dropped at the router's queue. COPE allows the bottleneck router to XOR pairs of packets and drain them twice as fast, doubling the throughput of this network. Thus, the Coding+MAC gain of the Alice-and-Bob topology is 2.

The Coding+MAC gain assumes all nodes continuously have some traffic to send (i.e., backlogged), but are limited by their MAC-allocated bandwidth. It computes the throughput gain with COPE under such conditions. For topologies with a single bottleneck, like the Alice-and-Bob's, the Coding+MAC gain is the ratio of the bottleneck's draining rate with COPE to its draining rate without COPE.

Similarly, for the "X" and cross topologies, the Coding+MAC gain is higher than the coding gain. For the "X", the Coding+MAC gain is 2 since the bottleneck node is able to drain twice as many packets, given its MAC allocated rate. For the cross topology, the Coding+MAC gain is even higher at 4. The bottleneck is able to send 4 packets out in each transmission, hence it is able to drain four times as many packets compared to no coding. This begs the question: what is the maximum Coding+MAC gain? The maximum possible Coding+MAC gains with and without opportunistic listening are properties of the topology and the flows that exist in a network. Here we prove some upper bounds on Coding+MAC gains.

Theorem 3.2 *In the absence of opportunistic listening, COPE's maximum Coding+MAC gain is*

2, and it is achievable.

Proof. As proved above, in the absence of opportunistic listening, a node can code atmost 2 packets together. Hence, a bottleneck node can drain its packets atmost twice as fast, bounding the Coding+MAC gain at 2. This gain is achieved even in the simple Alice-and-Bob experiment as explained above (longer chains result in the same Coding+MAC gain). ■

Theorem 3.3 *In the presence of opportunistic listening, COPE's maximum Coding+MAC gain is unbounded.*

Proof. Consider the wheel topology with radius r in Fig. 3-5(d) with N nodes uniformly placed on the circumference, and one node at the center of the circle. Assume that when a node transmits, all other nodes in the circle overhear this transmission, except for the diametrically opposed node (i.e., the radio range is $2r - \varepsilon$, where $\varepsilon \approx 0$). Suppose now that there are flows between every pair of diametrically opposed nodes. Note that nodes on either end of a diameter cannot communicate directly, but can communicate using a two-hop route through the middle node. In fact, this route is the geographically shortest route between these nodes. In the absence of coding, a single flow requires 1 transmission from an edge node, and 1 transmission from the middle node. This adds to a total of 1 transmission per edge node, and N transmissions for the middle node, across all packets. Since the MAC gives each node only a $\frac{1}{N+1}$ share of the medium, the middle node is the bottleneck in the absence of coding. However, COPE with opportunistic listening allows the middle node to code all the N incoming packets and fulfill the needs of all flows with just one transmission, thereby matching its input and output rates. Hence, the Coding+MAC gain is N, which grows without bound with the number of nodes. ■

While the previous example is clearly artificial, it does illustrate the potential of COPE with opportunistic listening to produce a several-fold improvement in throughput, as in Section 3.5. Table 3-2 lists the gains for a few basic topologies.

Topology	Coding Gain	Coding+MAC Gain
Alice-and-Bob	1.33	2
"X"	1.33	2
Cross	1.6	4
Infinite Chain	2	2
Infinite Wheel	2	∞

Table 3-2: Theoretical gains for a few basic topologies.

3.5 Experimental Results

This section uses measurements from a 20-node wireless testbed to study both the performance of COPE and the interaction of network coding with the wireless channel and higher-layer protocols. Our experiments reveal the following findings:

- When the wireless medium is congested and the traffic consists of many random UDP flows, COPE delivers a 3-4x increase in the throughput of our wireless testbed.
- When the traffic does not exercise congestion control (e.g., UDP), COPE's throughput improvement substantially exceeds the expected coding gain and agrees with the Coding+MAC gain.
- For a mesh network connected to the Internet via a gateway, the throughput improvement observed with COPE varies depending on the ratio of download traffic to upload traffic at the the gateway, and ranges from 5% to 70%.
- Hidden terminals create a high loss rate that cannot be masked even with the maximum number of 802.11 retransmissions. In these environments, TCP does not send enough to utilize the medium and does not create coding opportunities. In environments with no hidden terminals, TCP's throughput improvement with COPE agrees with the expected coding gain.

The evaluation section demonstrates the three factors that the gains of COPE depend on

- **Topology** Topologies that have multiple flows going through a router show higher gains.
- **Traffic** Networks where the traffic is not uni-directional can achieve significantly higher gains.

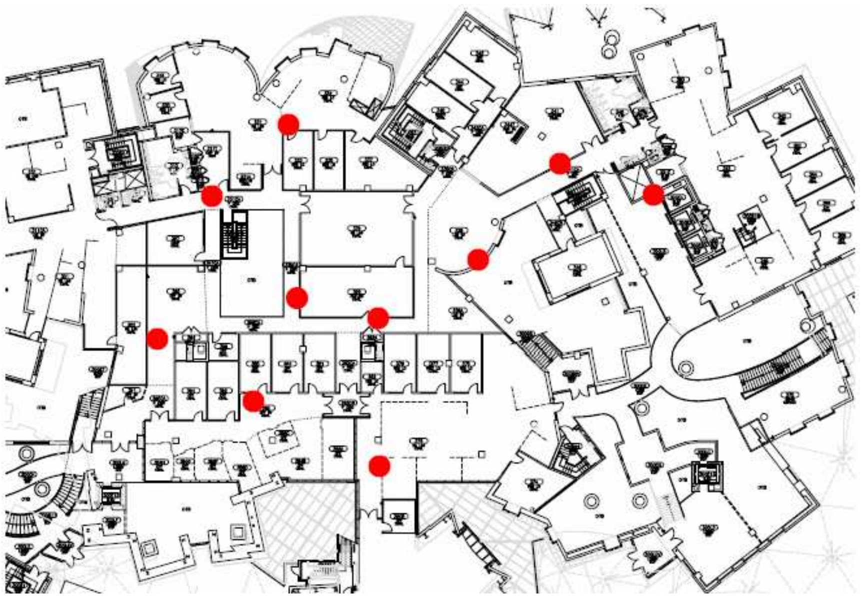


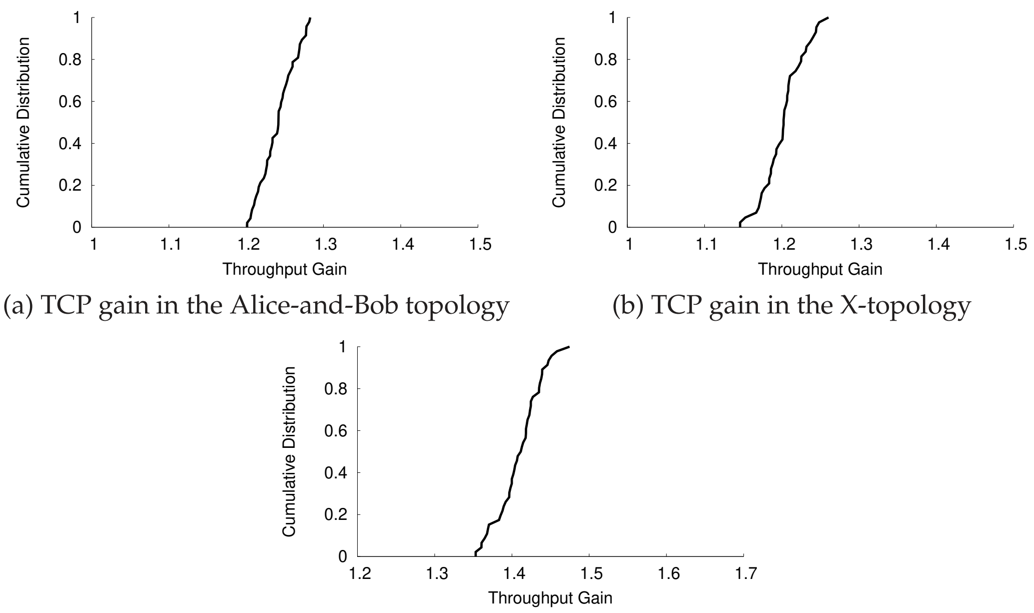
Figure 3-6: Node locations for one floor of the testbed.

- **Transport Protocol** The gains also depend on the interaction between the congestion control protocol employed by the flows and the underlying MAC protocol. When the flows employ TCP, the gains are consistent with the coding gain for the corresponding topology defined in Section 3.4. With UDP that doesn't employ congestion control, the gains are higher and correspond to the Coding+MAC gain.

■ 3.5.1 Testbed

(a) Characteristics: We have a 20-node wireless testbed that spans two floors in our building connected via an open lounge. The nodes of the testbed are distributed in several offices, passages, and lounges. Fig. 3-6 shows the locations of the nodes on one of the floors. Paths between nodes are between 1 and 6 hops in length, and the loss rates of links on these paths range between 0 and 30%. The experiments described in this chapter run on 802.11a with a bit-rate of 6Mb/s. Running the testbed on 802.11b is impractical because of a high level of interference from the local wireless networks.

(b) Software: Nodes in the testbed run Linux. COPE is implemented using the Click toolkit [78]. Our implementation runs as a user space daemon, and sends and receives raw 802.11 frames from the wireless device using a libpcap-like interface. The implementation exports a network interface to the user that can be treated like any other network



(a) TCP gain in the Alice-and-Bob topology (b) TCP gain in the X-topology

(c) TCP gain in the cross topology

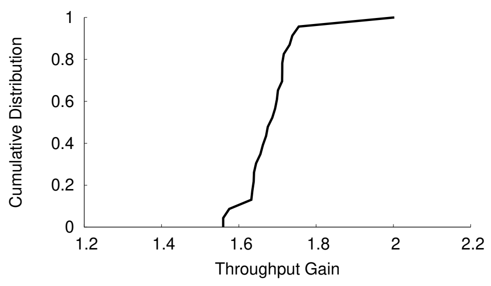
Figure 3-7: CDF of throughput gains obtained with COPE, for long-lived TCP flows.

device (e.g., `eth0`). Applications interact with the daemon as they would with a standard network device provided by the Linux kernel. No modifications to the applications are therefore necessary. The implementation is agnostic to upper and lower layer protocols, and can be used by various protocols including UDP and TCP.

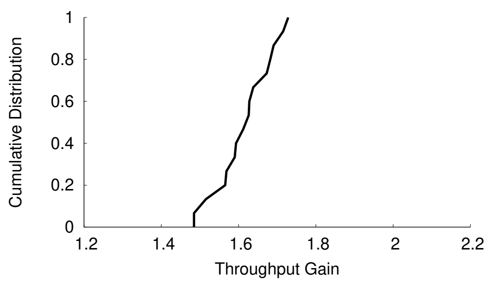
(c) Routing: Our testbed nodes run the Srcr implementation [27], a state-of-the-art routing protocol for wireless mesh networks. The protocol uses Djikstra's shortest path algorithm on a database of link weights based on the ETT metric [27]. Router output queue is bounded at 100 packets.

(d) Hardware: Each node in the testbed is a PC equipped with an 802.11 wireless card attached to an omni-directional antenna. The cards are based on the NETGEAR 2.4 & 5 GHz 802.11a/g chipset. They transmit at a power level of 15 dBm, and operate in the 802.11 ad hoc mode, with RTS/CTS disabled.

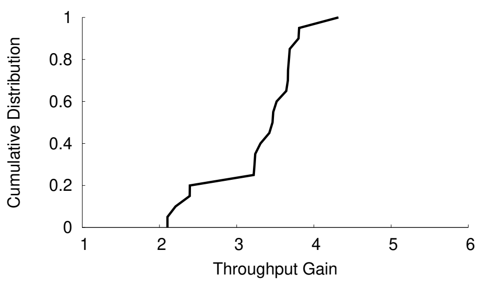
(e) Traffic Model: We use a utility program called `udpgen` from the Click [78] software package to generate UDP traffic, and `ttcp` [98] to generate TCP traffic. We either use long-lived flows, or many shorter flows that match empirical studies of Internet traffic [103, 101], i.e., they have Poisson arrivals, and a Pareto file size with the shape parameter set to 1.17.



(a) UDP gain in the Alice-and-Bob topology



(b) UDP gain in the X-topology



(c) UDP gain in the cross topology

Figure 3-8: CDF of throughput gains obtained with COPE, for UDP flows.

■ 3.5.2 Metrics

Our evaluation uses the following metrics.

- *Network Throughput:* The measured total end-to-end throughput, i.e., the sum of the throughput of all flows in the network as seen by their corresponding applications.

- *Throughput Gain:* The ratio of the measured network throughputs with and without COPE. We compute the throughput gain from two consecutive experiments, with coding turned on, then off.

■ 3.5.3 COPE in gadget topologies

We would like to compare COPE's actual throughput gain with the theoretical gains described in Section 3.4, and study whether it is affected by higher layer protocols. We start by looking at a few toy topologies with good link quality (medium loss rate after MAC retries $< 1\%$), and no hidden terminals.

Long-Lived TCP Flows

We run long-lived TCP flows over 3 toy topologies: Alice-and-Bob, the "X", and the cross topologies depicted in Figs. 3-1 and 3-5. Fig. 3-7 plots the CDFs of the TCP throughput gain measured over 40 different runs. For the Alice-and-Bob topology the gain, shown in Fig. 3-7(a), is close to the theoretical coding gain of 1.33. The difference of $5-8\%$ is due to the overhead of COPE's headers, as well as asymmetry in the throughput of the two flows, which prevents the router from finding a codemate for every packet. Similarly, for the "X"-topology, the gain in Fig. 3-7(b) is comparable to the optimal coding gain of 1.33. Finally, Fig. 3-7(c) shows the throughput gain for the cross topology with TCP. The gains are slightly lower than the expected coding gain of 1.6 because of header overhead, imperfect overhearing, and a slight asymmetry in the throughputs of the four flows.

The above experimental results reveal that when the traffic exercises congestion control, the throughput gain corresponds to the coding gain, rather than the Coding+MAC gain. The congestion control protocol, built into TCP, naturally matches the input rate at the bottleneck to its draining rate. When multiple long-lived TCP flows get bottlenecked at the same router, the senders back off and prevent excessive drops, leaving only pure coding gains.

UDP Flows

We repeat the above experiments with UDP and evaluate the throughput gains. Fig. 3-8 plots a CDF of the UDP gain with COPE for the Alice-and-Bob, the "X", and the cross topologies. The figure shows that the median UDP throughput gains for the three topologies are 1.7, 1.65, and 3.5 respectively.

Interestingly, the UDP gains are much higher than the TCP gains; they reflect the Coding+MAC gains for these toy topologies. Recall from Section 3.4 that the coding gain arises purely from the reduction in the number of transmissions achieved with COPE. Additionally, coding compresses the bottleneck queues, preventing downstream congested routers from dropping packets that have already consumed bandwidth, and producing a Coding+MAC gain. In Section 3.4, we have shown that the theoretical Coding+MAC gains for the above toy topologies are 2, 2, and 4 respectively. These numbers are fairly close to the numbers we observe in actual measurements.

One may wonder why the measured throughput gains are smaller than the theoretical Coding+MAC gain bounds. The XOR headers add a small overhead of 5-8%. However, the difference is mainly due to imperfect overhearing and flow asymmetry. Specifically, the nodes do not overhear all transmitted packets. Further, some senders capture the wireless channel sending more traffic in a particular direction, which reduces coding opportunities and overall gain.

In practice, traffic is a combination of congestion-controlled and uncontrolled flows. Further, most TCP flows are short-lived and do not fully exercise congestion control during slow-start.Thus, one would expect COPE's gains to be higher than those observed with long-lived TCP and lower than those observed with UDP. Indeed, we have run experiments for the Alice-and-Bob scenario with short-lived TCP flows with Poisson arrivals and Pareto transfer size. Depending on the flow inter-arrival times, the measured throughput gains vary between the coding gain and the Coding+MAC gain.

■ 3.5.4 COPE in an Ad Hoc Network

How does COPE perform in a wireless mesh network? We have advocated a simple approach to wireless network coding where each node relies on its local information to detect coding opportunities, and when possible XORs the appropriate packets. However, it is unclear how often such opportunities arise in practice, and whether they can be detected using only local information. Thus, in this section, we run experiments on our 20-node testbed to gauge the throughput increase provided by COPE in an ad hoc network.

TCP

We start with TCP flows that arrive according to a Poisson process, pick sender and receiver randomly, and transfer files whose sizes follow the distribution measured on the Internet [101].

Surprisingly, in our testbed, TCP does not show any significant improvement with coding (the average gain is 2-3%). The culprit is TCP's reaction to collision-related losses. There are a number of nodes sending packets to the bottleneck nodes, but they are not within carrier sense range of each other, resulting in the classic hidden terminals problem. This creates many collision-related losses that cannot be masked even with the maximum number of MAC retries. To demonstrate this point, we repeat the TCP experiments with varying number of MAC retransmissions with RTS/CTS enabled. Note that disabling RTS/CTS exacerbates the problem further. Fig. 3-9 plots the end-to-end loss rates for TCP flows as a function of the number of MAC retransmissions. *These experiments have COPE turned off.* Even after 15 MAC retries (the maximum possible) the TCP flows experience

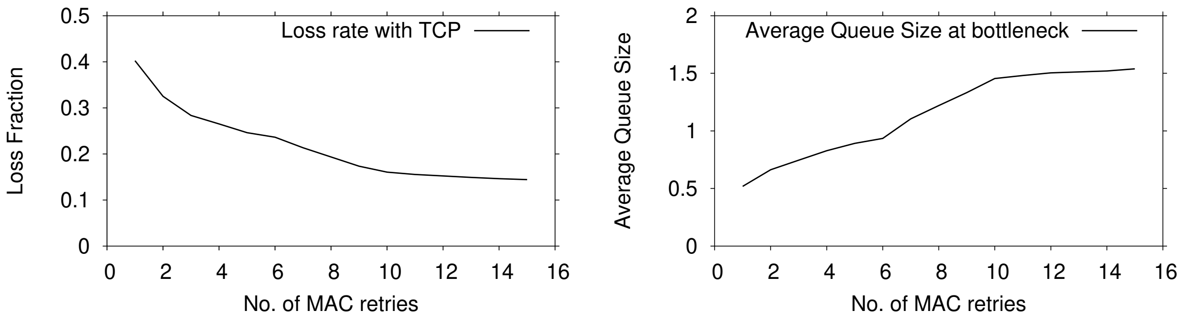


Figure 3-9: End-to-end loss rate and average queue size at the bottlenecks for the TCP flows in the testbed. Loss rates are as high as 14% even after 15 MAC retries; TCP therefore performs poorly. The queues at the bottlenecks almost never build up resulting in very few coding opportunities and virtually no gains.

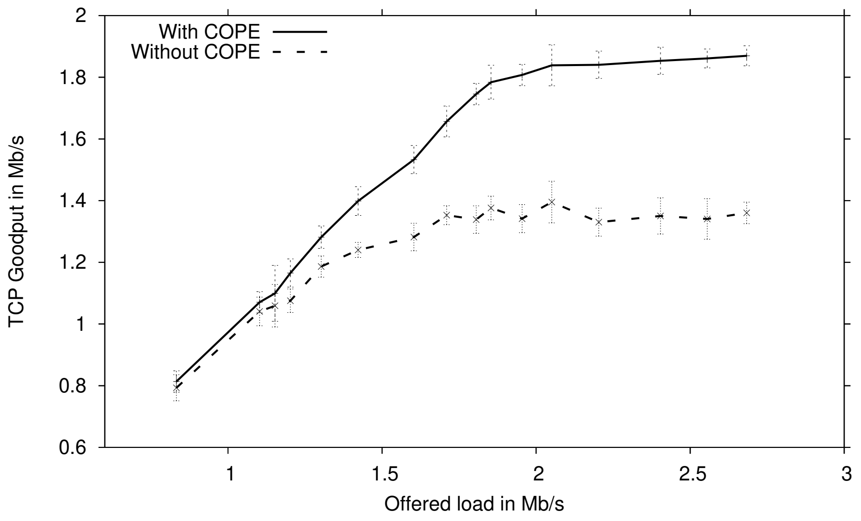


Figure 3-10: COPE provides 38% increase in TCP goodput when the testbed topology does not contain hidden terminals.

14% loss. As a result, the TCP flows suffer timeouts and excessive back-off, and are unable to ramp up and utilize the medium efficiently. Fig. 3-9 plots the average queue sizes at the bottleneck nodes[1]. The bottleneck nodes never see enough traffic to make use of coding; most of their time is spent without any packets in their queues or just a single packet. Few coding opportunities arise, and hence the performance is the same with and without coding.

Collision-related losses are common in wireless networks and recent work has studied their debilitating effect on TCP [42, 21]. Making TCP work in such a setting would imply solving the collision problem; such a solution is beyond the scope of this thesis.

[1] The few nodes connecting the two floors are where the flows intersect; they are the main bottlenecks in our testbed.

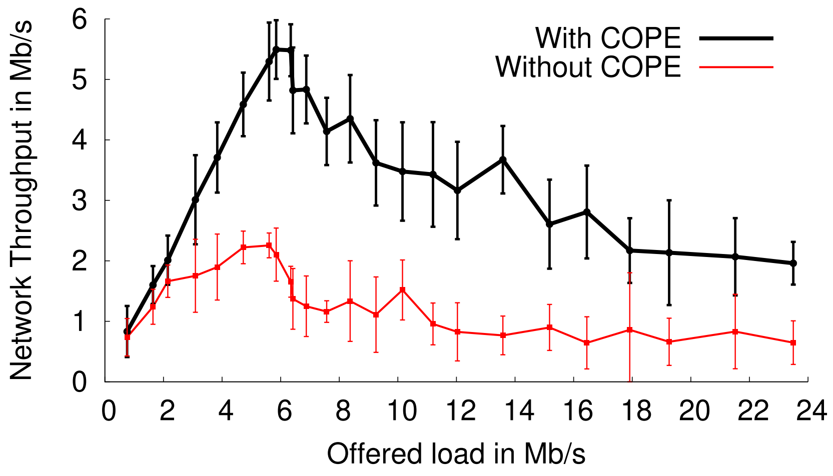


Figure 3-11: COPE can provide a several-fold ($3\times-4\times$) increase in the throughput of wireless mesh networks. Results are for UDP flows with randomly picked source-destination pairs, Poisson arrivals, and heavy-tail size distribution.

Would TCP be able to do better with COPE if we eliminated collision-related losses? We test the above hypothesis by performing the following experiment. We compress the topology of the testbed by bringing the nodes closer together, so that they are within carrier sense range. We artificially impose the routing graph and inter-node loss rates of the original testbed. The intuition is that the nodes are now within carrier sense range and hence can avoid collisions. This will reduce the loss rates and enable TCP to make better use of the medium. We repeat the above experiment with increasing levels of congestion obtained by decreasing the inter-arrival times of the TCP flows. Fig. 3-10 plots the network TCP goodput with and without COPE as a function of the demand. For small demands, COPE offers a slight improvement since coding opportunities are scarce. As the demands increase, network congestion and coding opportunities increase, leading to higher goodput gains. As congestion increases beyond a certain level, the throughput levels off, reflecting the fact that the network has reached its capacity and cannot sustain additional load. At its peak, COPE provides 38% improvement over no coding. The medium loss rates after retransmissions are negligible. The TCP flows are therefore able to use the medium efficiently, providing coding opportunities which result in throughput gains.

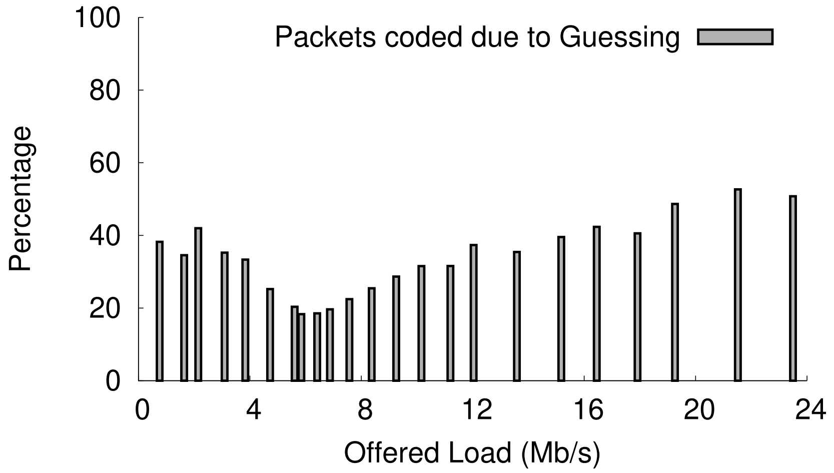


Figure 3-12: Percentage of packets coded in the testbed due to guessing, as a function of offered load, for the set of experiments in Fig. 3-11.

UDP

We repeat the large scale testbed experiments with UDP. The flows again arrive according to a Poisson process, pick sender and receiver randomly, and transfer files whose sizes follow the distribution measured on the Internet [101]. We vary the arrival rates of the Poisson process to control the offered load. For each arrival rate, we run 10 trials, with coding on and then off (for a total of 500 experiments), and compute the network throughput in each case.

Fig. 3-11 shows that COPE greatly improves the throughput of these wireless networks, by a factor of 3-4x on average. The figure plots the aggregate end-to-end throughput as a function of the demands, both with COPE and without. At low demands (below 2Mb/s), coding opportunities are scarce, and COPE performs similar to no coding. As demands increase, both network congestion and the number of coding opportunities increase. In such dense networks, the performance without coding deteriorates because of the high level of contention and consequent packet loss due to collisions. In contrast, coding reduces the number of transmissions, alleviates congestion, and consequently yields higher throughput.

It is interesting to examine how much of the coding is due to guessing, as opposed to reception reports. Fig. 3-12 plots the percentage of packets that have been coded because of guessing for the experiments in Fig.3-11. It is calculated as follows: If n packets are coded together, and at most k packets could be coded using reception reports alone, then $n - k$

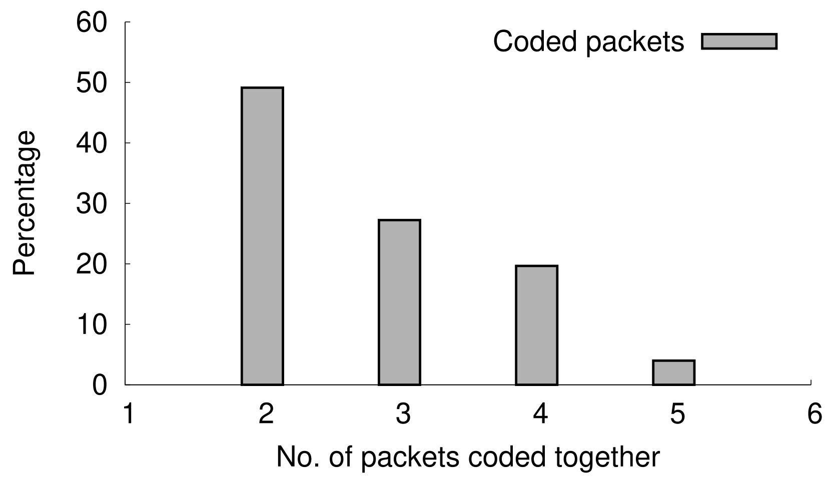


Figure 3-13: Distribution of number of packets coded together in the test bed at the peak point of Fig. 3-11.

packets are considered to be coded due to guessing. The figure shows that the benefit of guessing varies with demands. At low demands, the bottleneck nodes have small queues, leading to a short packet wait time. This increases dependence on guessing because reception reports could arrive too late, after the packets have been forwarded. As demands increase, the queues at the bottlenecks increase, resulting in longer wait times, and consequently allowing more time for reception reports to arrive. Hence, the importance of guessing decreases. As demands surge even higher, the network becomes significantly congested, leading to high loss rates for reception reports. Hence, a higher percentage of the coding decisions are again made based on guessing.

Let us now examine in greater detail the peak point in Fig. 3-11, which occurs when demands reach 5.6 Mb/s. Fig. 3-13 shows the PDF of the number of native packets XOR-ed at the bottleneck nodes (i.e., the nodes that drop packets). The figure shows that, on average, nearly 3 packets are getting coded together. Due to the high coding gain, packets are drained much faster from the queues of the bottleneck nodes.

■ 3.5.5 COPE in a Mesh Access Network

There is growing interest in providing cheap Internet access using multi-hop wireless networks that connect to the rest of the Internet via one or more gateways/access points [1, 8, 129]. We evaluate COPE in such a setting, where traffic is flowing to and from the closest gateway. We divide the nodes in the testbed into 4 sets. Each set communicates

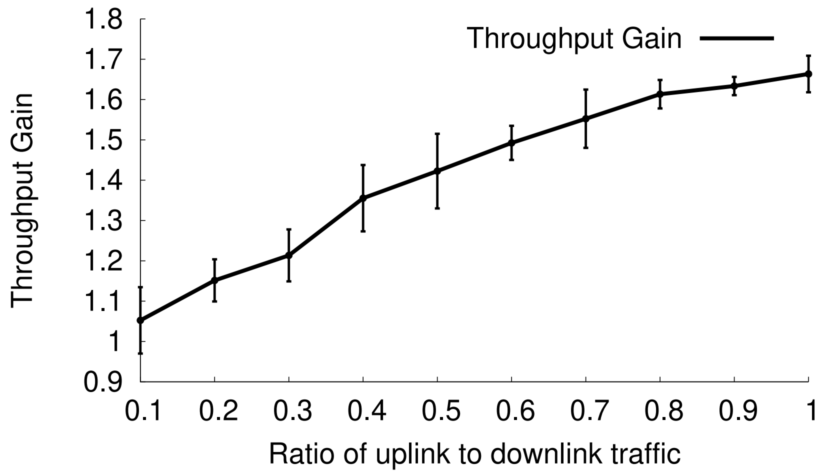


Figure 3-14: COPE's throughput gains as a function of the ratio of uplink to downlink traffic in a congested mesh access network.

with the Internet via a specific node that plays the role of a gateway. We use UDP flows[2], and control the experiments by changing the ratio of the upload traffic to download traffic. Fig. 3-14 plots the throughput gains as a function of this ratio.

The throughput gain increases as the fraction of uplink traffic increases. When the amount of uplink traffic is small, gains are correspondingly modest; around $5 - 15\%$. As uplink traffic increases, gains increase to 70%. COPE's throughput gain relies on coding opportunities, which depend on the diversity of the packets in the queue of the bottleneck node. For example, in the Alice-and-Bob topology, if only 10% of the packets in the bottleneck queue are from Alice and 90% from Bob, then coding can at best sneak 10% of Alice's packets out on Bob's packets. Hence, as the ratio of uplink traffic increases, the diversity of the queues at bottlenecks increases, more coding opportunities arise, and consequently higher throughput gains are obtained.

Fairness

The access network experiment above illuminates the effect fairness has on coding opportunities. An important source of unfairness in wireless networks is the comparative quality of the channels from the sources to the bottleneck, usually referred to as the *capture effect*. For example, in the Alice and Bob experiment, if the channel between Alice and the

[2]As mentioned earlier, in the uncompressed testbed, TCP backs off excessively because of collision-based losses from hidden terminals, and does not send enough to fully utilize the medium.

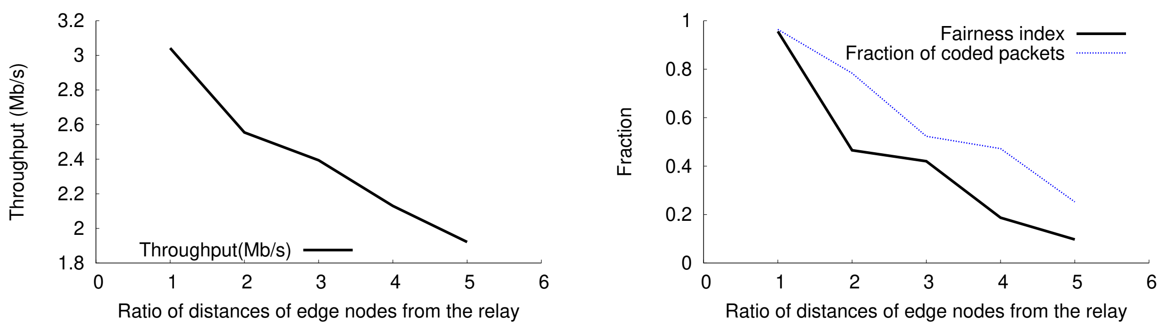


Figure 3-15: Effect of unequal channel qualities on coding opportunities and throughput gain in the Alice-and-Bob topology. COPE aligns the fairness and efficiency objectives. Increased fairness increases coding opportunities and hence improves the aggregate throughput.

router is worse than that between Bob and the router, Alice might be unable to push the same amount of traffic as Bob. Although the 802.11 MAC should give a fair allocation to all contenders, the sender with the better channel (here Bob) usually captures the medium for long intervals. The routing protocol tries to discount the capture effect by always selecting the stronger links; but in practice, capture always happens to some degree.

We study the effect of capture on COPE by intentionally stressing the links in the Alice and Bob topology. We set it up such that both Alice and Bob are equidistant from the router, and compute the total network throughput. We then gradually move Alice's node away from the router, and repeat the experiment and the measurements.

Fig. 3-15 shows the network throughput as a function of the ratio of Alice's and Bob's distance to the router. It also shows the percentage of coded packets and the *fairness index*, computed as the ratio of Alice's throughput to Bob's. As Alice moves further away, Bob increasingly captures the channel, reducing fairness, coding opportunities, and the aggregate network throughput. Interestingly, without coding, fairness and efficiency are conflicting goals; throughput increases if the node with the better channel captures the medium and sends at full blast. Coding, however, aligns these two objectives; increasing fairness increases the overall throughput of the network.

■ 3.6 Related Work

In this section we discuss the related work for COPE. Related work can be divided into three parts: first, how traditional network coding compares to COPE; second, theoretical work on multiple unicasts and network coding; and finally, further work by researchers that built on COPE. We review each of them in separate paragraphs below.

As discussed before, traditional work on network coding started with a pioneering paper by Ahlswede et al. [2], who showed that having the routers mix information in different messages allows the communication to achieve multicast capacity. This was soon followed by the work of Li et al., who showed that, for multicast traffic (e.g., the butterfly scenario), linear codes are sufficient to achieve the maximum capacity bounds [85]. Koetter and Médard [77] presented polynomial time algorithms for encoding and decoding, and Ho et al. extended these results to random codes [58]. Some recent work studied wireless network coding [40]. In particular, Lun et al. studied network coding in the presence of omni-directional antennae and showed that the problem of minimizing the communication cost can be formulated as a linear program and solved in a distributed manner [91]. All of this work is primarily theoretical and assumes multicast traffic. Further, it assumed fixed rate traffic, known topology and idealized congestion control algorithms.

Recent work has also looked at the problem of improving the throughput of multiple simultaneous unicast sessions with network coding. The problem is similar in spirit to the one addressed by COPE (since it too addresses multiple unicast sessions), but the techniques are quite different. However, performing inter-session coding is difficult. To perform inter-session coding optimally, linear coding operations are not sufficient [86], and, even if we limit ourselves to particular class of linear coding operations, deciding what operations to perform is an NP-hard problem [86]. A number of papers have examined sub-optimal, yet improved, methods for inter-session coding [122], most of which are based on remedies for decoding packets which got encoded with packets from other flows. Most of these algorithms are variations built on top of the simple design presented in this chapter. COPE, and in general this thesis, has a mix of conceptual and practical focus, designing simple algorithms and demonstrating that they are practical by building a real implementation and testing them in a real setting.

Subsequent work after the first publication of COPE has focussed on two aspects: modifying the different layers to suit COPE's coding capabilities and analyzing the capacity gains of COPE. Rayanchu et al. [116] have proposed to modify the routing layer to increase coding opportunities and consequently throughput. Further work by the same group has looked at making the COPE layer loss-aware and how that can be used to improve throughput. Both these approaches involve modifying network layers to work in concert with COPE. Our work consciously chose to avoid changes in any other layer and it was a first-class goal to make the COPE layer a plug-and-play module. A large body [89, 116] of theoretical work has also analyzed the capacity bounds of COPE. Jiu et al. [89] show that COPE can asymptotically provide a factor of two gain over traditional routing.

Finally, a rich body of systems research has tackled the problem of improving the throughput of wireless networks. The proposed solutions range from designing better routing metrics [27, 105] to tweaking the TCP protocol [110], and include improved routing and MAC protocols [12, 75, 55]. Our work builds on these foundations but adopts a fundamentally different approach; it explores the utility of network coding in improving the throughput of wireless networks.

■ 3.7 Discussion

Finally, we would like to comment on the scope of COPE. The present design targets stationary wireless mesh networks where the nodes are not resource-constrained. More generally, COPE can be used in multi-hop wireless networks that satisfy the following:

- *Memory:* COPE's nodes need to store recently heard packets for future decoding. Only packets in flight are used in coding; there is no need to store packets that have already reached their destination. Consequently, the storage requirement should be slightly higher than a delay-bandwidth product. (For e.g., an 11 Mb/s network with a 50ms RTT has a delay-bandwidth product of 70 KB.)

- *Omni-directional antenna:* Opportunistic listening requires omni-directional antennas to exploit the wireless broadcast property.

- *Power requirements:* Our current design of COPE does not optimize power usage and assumes that the nodes are not energy limited.

The ideas in COPE may be applicable beyond WiFi mesh networks. Note that COPE can conceptually work with a variety of MAC protocols including WiMax and TDMA. One may envision modifying COPE to address the needs of sensor networks. Such a modification would take into account that only a subset of the sensor nodes is awake at any point of time and can participate in opportunistic listening. Sensor nodes may also trade-off saved transmissions for reduced battery usage, rather than increased throughput. Additionally, COPE may be useful for cellular relays. Deploying cellular base stations is usually expensive. A cheap way to increase coverage is to deploy relay nodes that intervene between the mobile device and the base station [33], creating a multi-hop cellular backbone. COPE would allow cellular relays to use the bandwidth more efficiently. Indeed, after the publication of COPE, we have learned that Ericsson has independently proposed a design for cellular relays with a subset of COPE's functionality, where the cellular relay XORs only duplex flows, as in the Alice-and-Bob scenario [33]. This scheme can be extended to make full usage of the ideas embedded in COPE.

Our community knows a few fundamental approaches that can improve wireless throughput, including more accurate congestion control, better routing, and efficient MAC protocols. We believe that COPE is an important step forward in our understanding of the potential of wireless networks because it presents a new orthogonal axis that can be manipulated to extract more throughput; namely, how to maximize the amount of data delivered in a single transmission. This is coding, which is an old theme, traditionally used at the physical and application layers. But COPE and a few other recent projects [18, 67] introduce coding to the networking community as a practical tool that can be integrated with forwarding, routing, and reliable delivery.

CHAPTER 4

ANC: Analog Network Coding

Wireless interference is considered harmful. Interference creates collisions, prevents reception, and wastes scarce bandwidth. Wireless networks strive to prevent senders from interfering. They may reserve the medium for a specific node using TDMA or probe for idleness as in 802.11. This fear of interference is inherited from single-channel design and may not be the best approach for a wireless network [26, 117]. With bandwidth being scarce in the frequencies allocated to wireless networks, enabling concurrent receptions despite interference is essential.

This chapter introduces Analog Network Coding (ANC). Instead of avoiding interference, we exploit the interference of strategically picked senders to increase network throughput. When multiple senders transmit simultaneously, the packets collide. But looking deeper at the signal level, collision of two packets means that the channel adds their physical signals after applying attenuations and time shifts. Thus, if the receiver knows the content of the packet that interfered with the packet it wants, it can cancel the signal corresponding to the known packet after correcting for channel effects. The receiver is left with the signal of the packet it wants, which it decodes using standard methods. In a wireless network, packets traverse multiple hops. When packets collide, nodes often know one of the colliding packets by virtue of having forwarded it earlier or having overheard it. Thus, our approach encourages two senders to transmit simultaneously if their receivers can leverage network-layer information to reconstruct the interfering signal, and disentangle it from the packet they want.

Note the analogy between analog network coding and its digital counterpart, COPE. In COPE, senders transmit sequentially, the routers mix the content of the packets and broadcast the mixed version. In analog network coding, senders transmit simultaneously. The wireless channel naturally mixes these signals. Instead of forwarding mixed packets, routers forward mixed signals.

We build on prior work on the capacity of the 2-way relay channel [117, 95] and a recent paper on physical network coding [137]. Prior work, however, is focused on theoretical bounds and thus allows impractical assumptions. First, it assumes that the interfering signals are synchronized at symbol boundaries. Second, it assumes the channel functions are known a priori and do not change. We make no such impractical assumptions. We further implement our design in a software-radio testbed, showing the practicality of our approach.

ANC was the first work to present a full-fledged implementable design that exploits analog network coding to increase network capacity. Our contributions can be summarized as follows:

- We present a practical approach to perform analog network coding within a flow, and across different flows that intersect at a router.
- In contrast to prior work which assumes synchronized signals, our design exploits the lack of synchronization between the interfering signals to facilitate decoding.
- We implement our approach in software radios, proving its practicality.
- We evaluate analog network coding in a testbed of software radios. Empirical results show that our technique decodes interfered packets with an average bit error rate as low as 2-4%. As for the throughput, it increases by 70% in comparison with no-coding, and by 30% in comparison with traditional network coding.

4.1 Illustrative Examples

We explain the benefits of analog network coding using two canonical topologies, common in a mesh network. These two examples constitute building blocks for larger networks.

(a) Flows Intersecting at a Router: Consider the canonical example for wireless network coding which we described in Chapter 3. Alice and Bob want to send a message to each

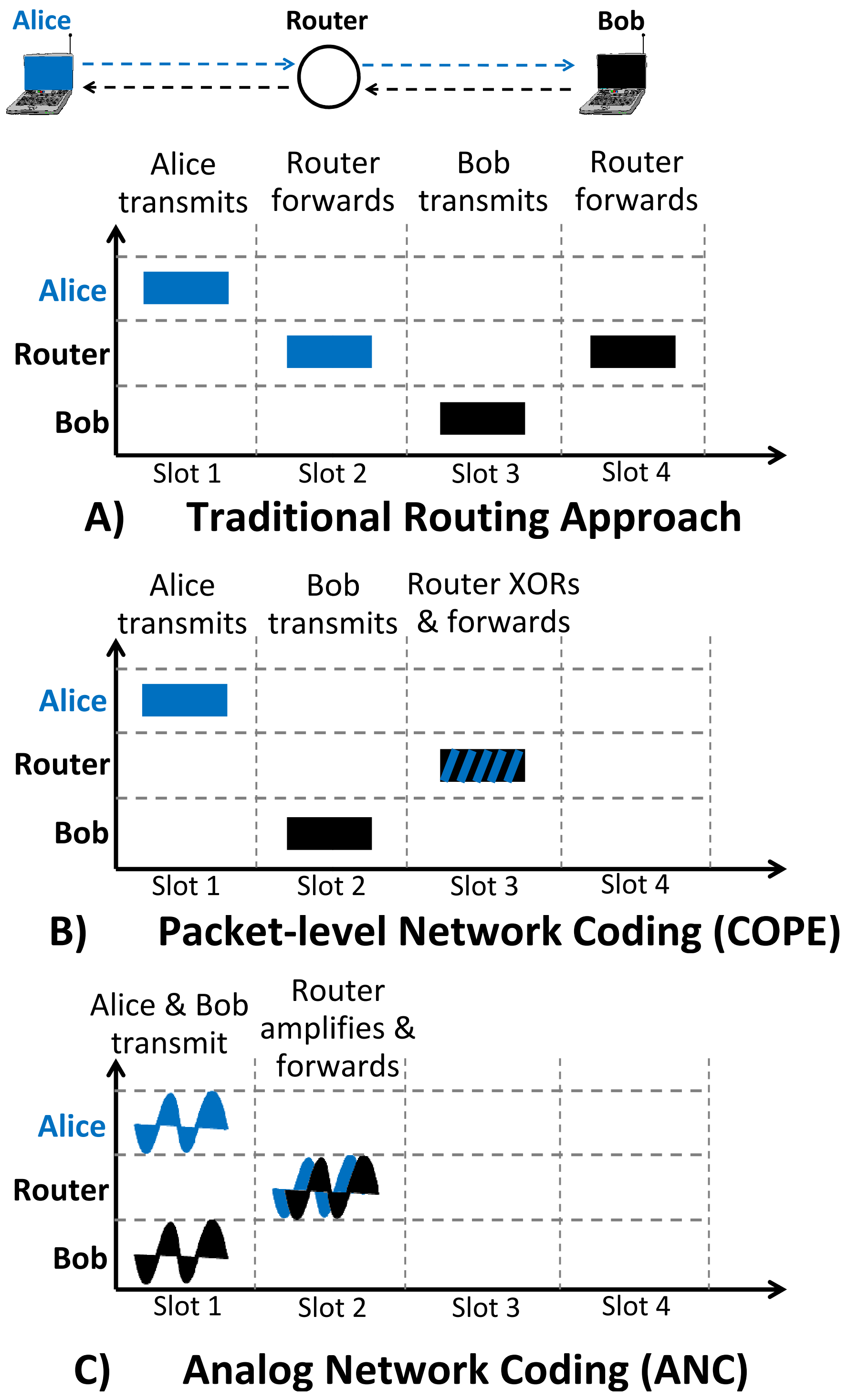


Figure 4-1: Alice-Bob Topology: Flows Intersecting at a Router. With analog network coding, Alice and Bob transmit simultaneously to the router, the router relays the interfered signal to Alice and Bob, who decode each others packets. This reduces the number of time slots from 4 to 2, doubling the throughput.

other. The radio range does not allow them to communicate without a router, as shown in Fig. 4-1(a). In the traditional approach, Alice sends her packet to the router, which forwards it to Bob, and Bob sends his packet to the router, which forwards it to Alice. Thus, to exchange two packets, the current approach needs 4 time slots. COPE achieves the same goal, but with fewer transmissions. In particular, Alice and Bob send their packets to the router one after the other; the router then XORs the two packets and broadcasts the XOR-ed version. Alice recovers Bob's packet by XOR-ing again with her own, and Bob recovers Alice's packet in the same way. Thus, COPE reduces the number of time slots from 4 to 3. The freed slot can be used to send new data, improving wireless throughput. But, can we reduce the time slots further? Can we deliver both packets in 2 time slots?

The answer is "yes". Alice and Bob could transmit their packets simultaneously, allowing their transmissions to interfere at the router. This consumes a single time slot. Due to interference, the router receives the sum of Alice's and Bob's signals, $s_A(t) + s_B(t)$. This is a collision and the router cannot decode the bits. The router, however, can simply amplify and forward the received interfered signal at the physical layer itself without decoding it. This consumes a second time slot. Since Alice knows the packet she transmitted, she also knows the signal $s_A(t)$ corresponding to her packet. She can therefore subtract $s_A(t)$ from the received interfered signal to get $s_B(t)$, from which she can decode Bob's packet. Bob can similarly recover Alice's packet. We call such an approach analog network coding. It is analogous to the network coding employedin COPE, but is done over physical signals in the wireless channel itself. As a result, we reduce the required time slots from 4 to 2, doubling the wireless throughput.

(b) Flows in a Single Direction: Analog network coding also applies to new scenarios to which traditional network coding (like COPE's codingtechnique) did not apply. Consider the chain topology in Fig. 4-2(a), where a single flow traverses 3 hops. The traditional routing approach needs 3 time slots to deliver every packet from source to destination. Digital network coding cannot reduce the number of time slots in this scenario, but analog network coding can.

Analog network coding improves the throughput of the chain topology in Fig. 4-2(a) because it allows nodes S and R2 to transmit simultaneously and have their packets received correctly despite collisions. In particular, let node R1 transmit packet P_i to R2. Then, S transmits the next packet P_{i+1}, whereas R2 forwards P_i to D. These two transmissions happen concurrently. The destination, D, receives only P_i because it is outside the radio range of node S. But, the two packets collide at node R1. With the traditional approach, R1 loses the packet sent to it by S. In contrast, in our approach, R1 exploits the fact that it knows the data in R2's transmission because it forwarded that packet to R2 earlier. Node R1 can recreate the signal that R2 sent and subtract that signal from the received signal. After subtraction, R1 is left with the signal transmitted by S, which it can decode to obtain packet P_{i+1}. Thus, instead of requiring a time slot for transmission on each hop, we can transmit on the first and third hops simultaneously, reducing the time slots from 3 to 2. This creates a throughput gain of $3/2 = 1.5$.

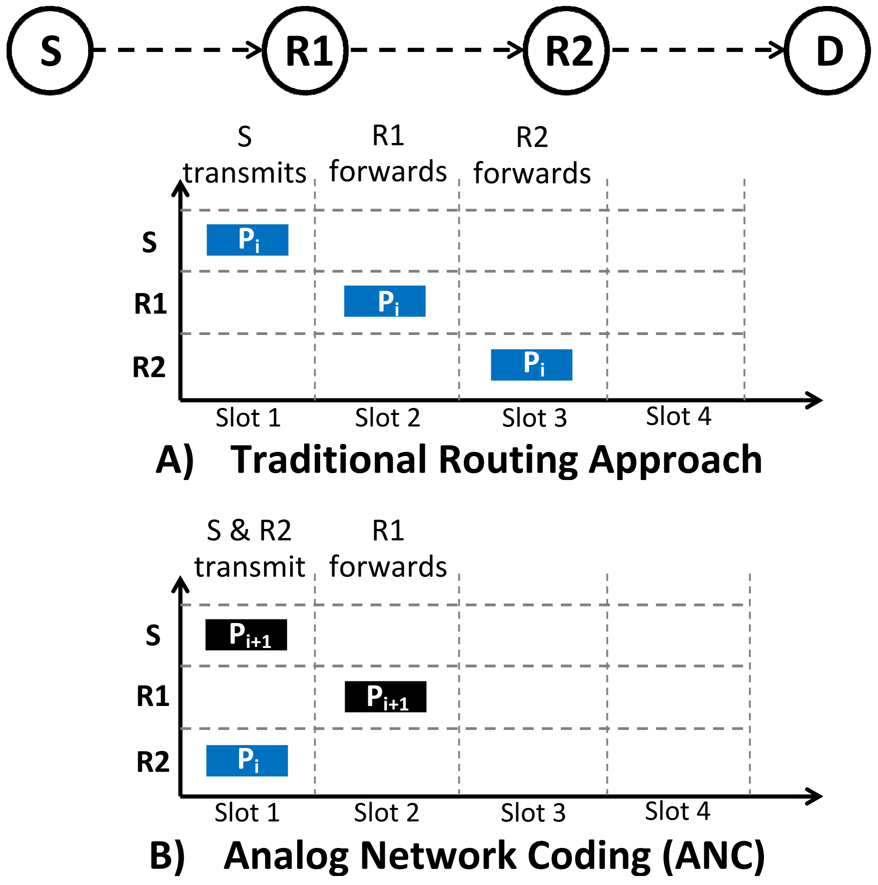


Figure 4-2: Chain Topology: Flows in one Direction. Nodes S and R2 can transmit at the same time. R1 gets an interfered signal, but can recover S's packet because it already knows the content of the interfering signal sent by R2. This reduces the time slots to deliver a packet from 3 to 2, producing a throughput gain of $3/2 = 1.5$

In practice, the throughput gain of the chain topology may be even higher. Without analog network coding, the nodes need an added mechanism to handle the hidden terminal problem in Fig. 4-2(a). They use either RTS-CTS or a statistical method like the exponential backoff built into the 802.11 MAC. Both methods incur a cost and reduce the achievable throughput [1]. With our approach, hidden terminals are harmless, and there is no need for an additional synchronization mechanism beyond carrier sense. Analog network coding therefore solves the hidden terminal problem for chain topologies with both uni-directional as well as bi-directional traffic. The hidden terminal problem persists in networks with multiple interacting chains and general ad-hoc networks. Addressing the hidden terminal problem in these latter cases is beyond the scope of this thesis.

The description above has intentionally ignored important details. For analog network coding to become practical, we need to address important challenges.

- The wireless channel distorts the signals, and hence Alice and Bob cannot simply subtract the signal they sent from the one they received to obtain each other's packet. They need to compensate for channel effects before they can cancel the interfering signal.

- Also, it is impossible for Alice's and Bob's transmissions to be fully synchronized. Thus, there will be a time shift between the two signals. A practical design has to work despite lack of synchrony between the interfering signals.

Thus, to implement a proof of concept of analog network coding, we have to dive into the physical layer and adapt channel acquisition, modulation, clock recovery, and other signal processing techniques to this new environment, namely, we need to design a new communication system from the ground up.

■ 4.2 Scope

Analog network coding is a general technique, independent of the underlying wireless technology. It is applicable in a wide variety of scenarios, with 802.11 mesh networks being an obvious example. Cellular networks and underwater acoustic wireless networks [121] are also possible examples. In particular, cellular networks deploy inexpensive bi-directional relays to expand their coverage area. These nodes intervene between the mobile device and the base station. They simply amplify and retransmit the signal they receive, which is exactly the functionality they need to implement analog network coding [33].

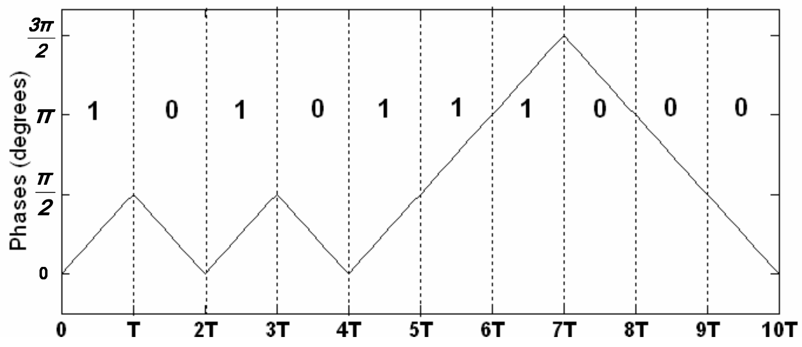


Figure 4-3: Example MSK modulation. MSK represents a bit of 1 as a phase difference of $\pi/2$ over an interval T. It represents a a bit of 0 as a phase difference of $-\pi/2$ over T.

Our goal is to design and implement a proof of concept of ANC. Since ANC works at the signal level, this implies designing an entire communication system from the ground up. Hence, we have to make a number of design choices at the physical layer. Most importantly, we have to choose a modulation/demodulation scheme. We want a modulation scheme that is widely used in many wireless technologies because it is infeasible to try the myriad of possible modulation schemes.

To this end, we choose phase shift keying (PSK), which is widely used in modern communication systems. For example, 802.11 uses Binary and Quadrature Phase Shift Keying (BPSK/QPSK) and GSM, a widely used cell-phone standard, uses a variant of Minimum Shift Keying (MSK), which is another form of phase shift keying. Our implementation uses MSK. MSK has very good bit-error properties, has a simple demodulation algorithm and excellent spectral efficiency. But the ideas we develop in this chapter, especially Section 4.4.1, are applicable to any differential phase shift keying modulation scheme.

■ 4.3 Background: Single Signal Case

Before talking about disentangling interfering signals, we need to explain how a single signal is transmitted and received over the wireless channel. For the sake of simplicity, we will intentionally gloss over some details that are unnecessary for understanding the technical ideas proposed in this chapter (e.g., pass-band vs. base-band, error correction, upconversion, and downconversion). We will describe how MSK transmits and receives a packet of bits.

4.3.1 Wireless Communication Systems

A wireless signal is usually represented as a complex function of time. Thus, the signal transmitted by the sender, which we annotate with the subscript s, can be represented as:

$$s(t) = A_s(t)e^{i\theta_s(t)},$$

where $A_s(t)$ is the amplitude of the waveform and $\theta_s(t)$ is its phase. To transmit a stream of bits, we need to map "0" and "1" to two different complex functions. Then, we divide time into consecutive slots of duration T. During each slot, we transmit the complex function corresponding to "1" or "0", depending on the bit value we want to transmit.

Although the transmitted signal is a continuous function, modern communication systems are digital. They produce discrete samples of the continuous signal. The wireless transmitter interpolates the samples to generate a continuous signal, which it transmits over the wireless channel. Thus, for the rest of this chapter, we will talk about complex samples, of the form $A_s[n]e^{i\theta_s[n]}$.

4.3.2 The Sender Side

Say that we have a packet to transmit over the wireless channel. As said above, we need to map "0" and "1" into two different complex representations. This is called modulation. In particular, the MSK modulation represents bits by varying the phase difference between consecutive complex samples. A phase difference of $\pi/2$ represents "1", whereas a phase difference of $-\pi/2$ represents a "0".

To see how MSK works, let us go through an example. Assume the data being sent is 1010111000, then the phase of the signal would vary as seen in Fig. 4-3. The signal itself is the complex function whose phase changes as shown in the figure. Initially, at time $t = 0$, the signal is A_se^{i0}. Since the first bit is a "1", the signal sample at time $t = T$ should be $A_se^{i(\pi/2)}$. The second bit is a "0", hence the signal sample at time $t = 2T$ should be $A_se^{i(\pi/2-\pi/2)} = A_se^{i0}$. This is repeated for all the bits. Note that in MSK, the amplitude of the transmitted signal, A_s, is a constant. The phase embeds all information about the bits.

4.3.3 The Receiver Side

How does the signal look like at the receiver, after traversing the wireless channel? The received signal is also a stream of complex samples spaced by T. But these samples differ from the transmitted samples, both in amplitude and phase. In particular, if the transmitted sample is $A_s[n]e^{i\theta_s[n]}$ the received signal can be approximated as:

$$y[n] = h\ A_s[n]e^{i(\theta_s[n]+\gamma)},$$

where h is channel attenuation and γ is a phase shift that depends on the distance between the sender and the receiver. The receiver needs to map the received complex samples back into a bit stream.

Demodulation is the process of mapping the received signal to the transmitted bits. For MSK, this amounts to discovering the phase differences between consecutive complex samples separated by T, and then mapping that phase difference back to a bit value.

Calculating phase differences of the complex samples is simple. Recall that in MSK, the amplitude of the samples is fixed and does not change from one signal sample to the next. Consider the following consecutive complex samples $h\ A_s\ e^{i(\theta_s[n+1]+\gamma)}$ and $h\ A_s\ e^{i(\theta_s[n]+\gamma)}$. First, we calculate the ratio of these complex numbers,

$$r = \frac{h\ A_s\ e^{i(\theta_s[n+1]+\gamma)}}{h\ A_s\ e^{i(\theta_s[n]+\gamma)}} = e^{i(\theta_s[n+1]-\theta_s[n])}. \tag{4.1}$$

To demodulate, we simply compute the angle of the complex number r, which gives us the phase difference, i.e., $\arg(r) = \theta_s[n+1] - \theta_s[n]$, where $\arg(x)$ is the angle of the complex number x. We map these phase differences to "0" and "1" bits using a simple rule. A positive phase difference is a "1" whereas a negative phase difference is a "0".

The most important fact about the computation in Eq. 4.1 is its invariance to both the channel attenuation h and the channel phase shift γ. This makes MSK demodulation very robust because the receiver does not need to accurately estimate the channel. Phase modulation schemes like MSK are therefore very attractive and are widely used in cellular communications and other networks.

4.4 Decoding Interfered MSK signals

So, how does Alice (or Bob) decode the interfered signals? The first step in answering this question is to understand what Alice receives. As described earlier, when Alice and Bob transmit simultaneously, the router receives the sum of their signals, amplifies this composite signal, and broadcasts it to Alice and Bob. Thus, Alice receives an interfered signal, $y_A(t) + y_B(t)$. However, $y_A(t)$ and $y_B(t)$ are not the two signals Alice and Bob have sent. Rather, they are the two transmitted signals after they traversed the channels from their corresponding senders to the router and the channel from the router to their corresponding receivers. The effect of the wireless channels can be approximated by an attenuation and phase shift [124]. Thus, the signal that Alice receives is:

$$\begin{aligned} y[n] &= y_A[n] + y_B[n] \\ y[n] &= h' A_s e^{i(\theta_s[n]+\gamma')} + h'' B_s e^{i(\phi_s[n]+\gamma'')}, \end{aligned}$$

where θ_s refers to the phase of the signal transmitted by Alice and ϕ_s refers to the phase of the signal transmitted by Bob, whereas A_s and B_s are the amplitudes at the transmitter.

Note that we use the subscript s to refer to the transmitted signal as opposed to the received signal, for which we use no subscripts. Note also that n refers to the index of the received sample; it is not the index of the bit transmitted by Alice or Bob[1].

At first, it seems that to decode the interfered signals, Alice should estimate the channel parameters h' and γ'. Once she knows these parameters, Alice recreates the version of her signal that interfered with Bob's signal, and subtracts it from the received signal. The result is $y_B[n]$, a sampled version of Bob's signal that Alice can decode using the standard method described in 4.3.

In practice, however, this subtraction method is not robust since it depends on the errors in Alice's estimate of the channel parameters. Though we tend to think of those parameters as constant, they do vary with time. Further, the channel model is approximate. There are other sources of noise that add up to the estimation errors.

We need a more robust method. Indeed, the main reason for the robustness of MSK is that demodulation does not require estimating the channel. Specifically, Eq. 4.1 computes

[1]Our design does not assume synchronization of Alice's and Bob's signals. We will talk about that issue in detail in 4.5.2

the phase difference without worrying about the exact values of γ and h. This gives us a hint of how to design a more robust demodulation scheme for interfered signals. In particular, one should focus on discovering the phase differences for the two signals, namely $\Delta\theta$ and $\Delta\phi$. It is phase differences that carry all the information about Alice's and Bob's bits, not the values of the phases themselves.

Thus, in the rest of this section, we will develop an algorithm that allows Alice to decode the phase differences between the consecutive samples of Bob's signal. For simplicity of notation, we will represent the received signal at Alice as:

$$y[n] = Ae^{i\theta[n]} + Be^{i\phi[n]}, \tag{4.2}$$

where $A = h'A_s$, $B = h''B_s$, $\theta[n] = \theta_s[n] + \gamma'$, and $\phi[n] = \phi_s[n] + \gamma''$.

How do you calculate phase differences when two signals interfere and you know the phase differences of one of the signals? We will use a two-step process. First, Alice uses her received signal to calculate pairs $(\Delta\theta, \Delta\phi)$ that could have produced the observed signal. Next, Alice uses her knowledge of her phase difference $\Delta\theta_s$ to pick the most likely pair. This gives Alice an estimate of $\Delta\phi$, Bob's phase difference. Based on this estimate Alice decides whether Bob sent a "0" or a "1".

■ 4.4.1 Possible Phases of Both Signals

Say that Alice receives the interfered signal in Eq. 4.2, can she tell the values of $\theta[n]$ and $\phi[n]$ just by analyzing the received signal? The answer is "No"; without extra information, Alice cannot tell the exact phases. She can, however, calculate possible values for those phases. First, we prove the following lemma.

Lemma 4.1 *If $y[n]$ is a complex number satisfying Eq. 4.2, then the pair $(\theta[n], \phi[n])$ takes one of the following two values.*

$$\begin{aligned} \theta[n] &= \arg(y[n](A + BD \pm iB\sqrt{1-D^2})) && (4.3) \\ \phi[n] &= \arg(y[n](B + AD \mp iA\sqrt{1-D^2})) && (4.4) \end{aligned}$$

where, $D = \frac{|y[n]|^2 - A^2 - B^2}{2AB}$, $|y[n]|$ is the norm, and $\arg$ is the angle of the complex number.

Proof. Since,

$$y[n] = Ae^{i\theta[n]} + Be^{i\phi[n]} \tag{4.5}$$

the square of the magnitude of $y[n]$ is given by,

$$|y[n]|^2 = A^2 + B^2 + 2AB\cos(\theta[n] - \phi[n]) \tag{4.6}$$

Let us denote $\cos(\theta[n] - \phi[n])$ by D. Hence, D can be computed as,

$$D = \cos(\theta[n] - \phi[n]) = \frac{|y[n]|^2 - A^2 - B^2}{2AB} \tag{4.7}$$

We use D, to separate out the two phases $\theta[n]$ and $\phi[n]$ into separate independent expressions. For brevity, we only show the computation for $\theta[n]$, the analysis for $\phi[n]$ is similar.

From Eq. 4.5, we can rewrite $e^{i\theta[n]}$ as

$$e^{i\theta[n]} \quad = \quad \frac{y[n](A + Be^{-i(\phi[n]-\theta[n])})}{A^2 + B^2 + 2AB\cos(\phi[n] - \theta[n])}$$

The complex number's phase is $\theta[n]$, which is the quantity of interest. Hence all we have to do is compute the phase of this complex number,

$$\begin{aligned} \theta[n] &= \arg(y[n](A + Be^{-i(\phi[n]-\theta[n])})) \\ \theta[n] &= \arg(y[n](A + B\cos(\phi[n] - \theta[n]) \\ &\quad -iBsin(\phi[n] - \theta[n]))) \end{aligned} \tag{4.8}$$

where $\arg(x)$ represents the phase of the complex number x.

Notice that we already know $\cos(\phi[n] - \theta[n]) = \cos(\theta[n] - \phi[n]) = D$. We can use this to compute $\sin(\phi[n] - \theta[n])$.

$$\sin(\phi[n] - \theta[n]) = \mp\sqrt{1 - D^2} \tag{4.9}$$

Substituting in Eqn. 4.8 we get two solutions for $\theta[n]$,

$$\theta[n] = \arg(y[n](A + BD \pm iB\sqrt{1 - D^2})) \tag{4.10}$$

Similarly we can compute $\phi[n]$ as,

$$\phi[n] = \arg(y[n](B + A\cos(\phi[n] - \theta[n]) + iAsin(\phi[n] - \theta[n])) \tag{4.11}$$

Thus, we get two corresponding solutions for $\phi[n]$ as well,

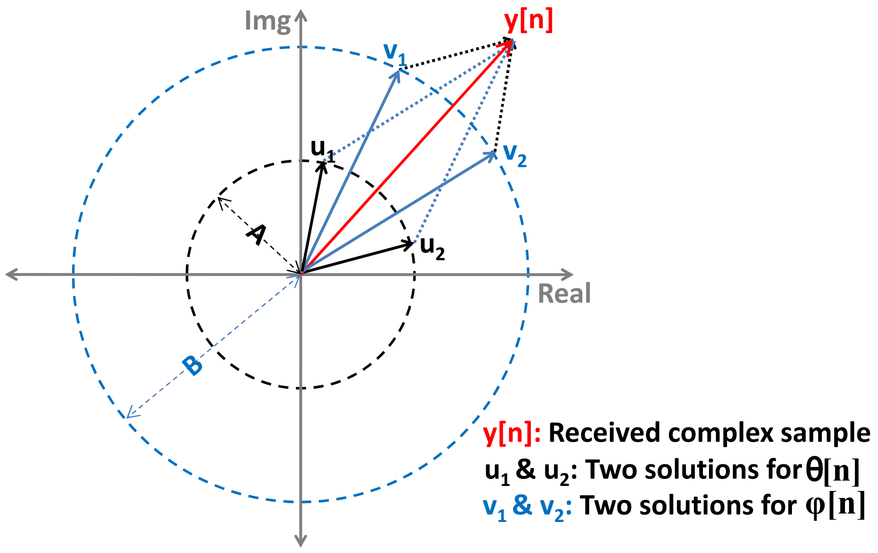

Img
y[n]
v1
v2
u1
u2
A
B
Real
y[n]: Received complex sample
u1 & u2: Two solutions for θ[n]
v1 & v2: Two solutions for φ[n]

Figure 4-4: Geometric representation of the phase computation. The received complex sample $y[n]$ is the sum of two complex numbers u and v. The length of the first complex number is A and the length of second is B. There are exactly two pairs of such complex numbers (u, v) that sum up to $y[n]$. Thus, two solutions exist for the pair $(\theta[n], \phi[n])$.

$$\phi[n] = \arg(y[n](B + AD \mp iA\sqrt{1 - D^2})) \tag{4.12}$$

■

Note that for each solution to $\theta[n]$, there is a unique solution for $\phi[n]$. Thus, when $\theta[n] = \arg(y[n](A + BD + iB\sqrt{1 - D^2}))$, the corresponding solution is $\phi[n] = \arg(y[n](B + AD - iA\sqrt{1 - D^2}))$. The solutions come in two pairs. The intuition underlying the proof can be explained geometrically. As a complex number, $y[n]$ can be represented with a vector, as in Fig. 4-4. According to Eq. 4.2, $y[n]$ is the sum of two vectors, which have lengths A and B respectively. Thus, we want to find a pair of vectors, (u, v), that sum up to the received complex sample, $y[n]$. The constraint is that the first vector is of length A and the second of length B, i.e.,the two vectors lie on two circles with radius A and B. From the figure, there are only two such pairs of vectors. Therefore, there are two solutions for the pair ($\theta[n]$, $\phi[n]$).

■ 4.4.2 Estimating the Amplitudes A and B

If Alice knows the amplitude of the two signals, i.e., A and B, she can substitute those values and the received complex sample $|y[n]|$ into the equations in Lemma 4.1 to calculate the phases. In fact, Alice can estimate A and B from the received signal. Since she has two unknowns (A and B), she needs two equations.

The first equation for computing A and B comes from the energy of the received signal. When two signals interfere, their energies add up. In particular, the energy is:

$$E[|y[n]|^2] \quad = \quad E[A^2 + B^2 + 2AB\cos(\theta[n] - \phi[n])],$$

where $E[.]$ is the expectation. The value of $E[\cos(\theta[n] - \phi[n])] \approx 0$ for a random bit sequence. To ensure the bits are random, we XOR them with a pseudo-random sequence at the sender, and XOR them again with the same sequence at the receiver to get the original bits. Hence,

$$E[|y[n]|^2] \quad = \quad A^2 + B^2.$$

Alice estimates the expectation by averaging the energy of the complex samples over a window of size N.

$$\mu = \frac{1}{N}\sum_{n=1}^{N} |y[n]|^2 = A^2 + B^2. \tag{4.13}$$

Alice still needs a second equation to estimate A and B. She computes the following quantity.

$$\sigma = \frac{2}{N}\sum_{|y[n]|^2 > \mu} |y[n]|^2.$$

Said differently, Alice computes the average energy in samples whose squared norm is greater than the mean energy μ. We can show that σ can be reduced to,

$$\sigma = A^2 + B^2 + 4AB/\pi. \tag{4.14}$$

Proof. From our definition,

$$\sigma = \frac{2}{N}\sum_{|y[n]|^2 > \mu} |y[n]|^2. \tag{4.15}$$

where μ is $A^2 + B^2$. Essentially we are calculating the expectation of those $|y[n]|^2$s which are greater than $A^2 + B^2$. From Eq. 4.6 we have,

$$|y[n]|^2 = A^2 + B^2 + 2ABcos(\theta[n] - \phi[n]) \tag{4.16}$$

Thus, in the computation of σ we are using only those $|y[n]|^2$ which have $cos(\theta[n] - \phi[n])$ greater than zero. Hence, we can rewrite σ as

$$\begin{aligned} &E(|y[n]|^2 \;|cos(\theta[n] - \phi[n]) > 0) \\ &= E(A^2 + B^2 + 2ABcos(\theta[n] - \phi[n]) \;|cos(\theta[n] - \phi[n]) > 0) \\ &= \; A^2 + B^2 + 2AB \; E(cos(\theta[n] - \phi[n]) \;|cos(\theta[n] - \phi[n]) > 0) \end{aligned}$$

Assuming that we are sending random bit patterns, we can derive $E(cos(\theta[n] - \phi[n])|cos(\theta[n] - \phi[n]) > 0)$ as $2/\pi$ by taking the average of a cosine over its positive lobes. Thus, finally we get

$$\sigma = A^2 + B^2 + 4AB/\pi \tag{4.17}$$

■

Given Eqs. 4.13 and 4.14, Alice has two equations with two unknowns and can solve for A and B.

■ 4.4.3 Estimating Phase Differences for Bob's Signal

Alice's next step is to estimate the phase differences of Bob's signal, i.e., $\phi[n+1] - \phi[n]$. She uses the phases from Lemma 4.1 to calculate phase differences of both her signal, $\theta[n+1] - \theta[n]$, as well as Bob's signal $\phi[n+1] - \phi[n]$. There is, however, ambiguity in these calculations because this lemma gives two solutions for each phase, at any sample time n. Alice cannot tell which of the two solutions is the correct one.

Alice therefore computes all possible phase differences based on Lemma 4.1. Let us denote the two solutions pairs as $(\theta_1[n], \phi_1[n])$ and $(\theta_2[n], \phi_2[n])$. Then, Alice has the following four possible phase difference pairs:

$$\begin{aligned} (\Delta\theta_{xy}[n], \Delta\phi_{xy}[n]) \;\; &= \;\; (\theta_x[n+1] - \theta_y[n], \phi_x[n+1] - \phi_y[n]) \\ &\qquad\qquad \forall x, y \in \{1, 2\} \end{aligned} \tag{4.18}$$

Next, Alice has to pick the right phase difference pair from the four choices in Eq. 4.18. This is where she leverages *network layer information*. Alice knows the signal she transmitted earlier, and which interfered with Bob's signal. Thus, she knows the phase difference of her transmission $\Delta\theta_s[n]$. Phase differences are fairly robust to channel distortion (if you take the phase difference the γ term cancels out). Thus, she can use the known $\Delta\theta_s[n]$ to pick the correct $\Delta\theta_{xy}$.

Alice calculates the error for each of the four choices she got from Eq. 4.18.

$$err_{xy} = |\Delta\theta_{xy}[n] - \Delta\theta_s[n]| \quad ,\forall x, y \in \{1, 2\} \tag{4.19}$$

Alice picks the $\Delta\theta_{xy}[n]$ that produces the smallest error err_{xy}. She finds the matching $\Delta\phi_{xy}[n]$ phase difference for Bob's signal. Alice repeats this for all values of n, to estimate the sequence of Bob's phase differences. She uses these estimated phase differences to decode Bob's bits.

■ 4.4.4 Obtaining Bob's Bits

Recall that MSK modulation maps "1" to a phase difference of $\pi/2$ and "0" to a phase difference of $-\pi/2$. In the last step above, Alice has an estimate of the phase differences of Bob's signal, $\Delta\phi[n]$. She now maps them back to bits. Because of estimation errors and the distortion of the received signal, the phases that Alice estimates do not match exactly the phases sent by Bob. Thus, Alice follows a simple rule.

$$\text{if } \Delta\phi[n] \geq 0, \text{the } n^{th} \text{ bit is "1", else it is "0".}$$

■ 4.5 Practical Issues

Is the scheme described above feasible in practice? The short answer is "yes". Building an operational communication system, however, involves many practical challenges.

■ 4.5.1 How Does Alice Detect Interference?

We begin with the most basic question: How does Alice detect a packet transmission? This is a standard problem in communication systems. To detect a transmission, Alice looks at the energy in the received signal. During transmission the energy level is much higher than the noise energy.

Next, how can Alice tell whether a packet has been subjected to interference? If it is an interfered packet, Alice needs to run the interference decoding algorithm described in 4.4;

otherwise, Alice runs standard MSK decoding.

To answer this question, Alice uses the variance in the energy of the received signal. Recall that, in MSK, the transmitted signal amplitude is constant; MSK encodes the bits in the phase, not the magnitude of the complex sample. Hence, the energy of a non-interfered MSK signal is nearly constant [2]. Packet interference destroys this property of nearly constant signal energy. When two packets collide, the signals interfere with each other in a random fashion. The constant energy property of MSK no longer holds. We use this insight to detect interference. We quantify this variation in energy by measuring the variance in the energy of the received samples. If the variance is greater than a threshold, Alice detects interference and applies the decoding algorithm from 4.4.

We calculate energy and energy variance over moving windows of received samples. Our detection algorithm declares occurrence of a packet if the energy is greater than $20dB$, which is a typical threshold. It detects interference if the variance in the energy is greater than $20dB$. This threshold is picked because when two MSK signals interfere, the energy of the interfered signal varies from $(A+B)^2$ to $(A-B)^2$, depending on whether they interfere constructively or destructively. Thus, the variance is on the order of $((A+B)^2 - (A-B)^2)^2 = 16A^2B^2$, which is greater than the energy of either of the signals constituting the interfered signal (i.e., greater than A^2 and B^2).

4.5.2 How Does Alice Deal with Lack of Synchronization?

In an ideal world, Alice's and Bob's signals arrive at the router at the same instant, and interfere exactly at the beginning of the two packets. In reality, there is a time shift between the two signals. This time shift complicates our algorithm described in 4.4. In particular, the algorithm needs Alice to match the phase difference of the signal she sent against four possible solutions, in order to pick the right one. But without synchronization, Alice does not know the index of the first interfering sample.

Interestingly, our solution to the problem leverages the lack of complete synchronization. Since packets do not interfere perfectly, there are parts at the start and end of the received signal which do not have any interference. For example, assume Alice's signal arrived before Bob's. Then, the first few bits of Alice's packet are interference free. Assuming Alice and Bob have similar packet sizes, the last few bits of Bob's packet are also interfer-

[2]The energy of a complex sample $Ae^{i\theta}$ is A^2.

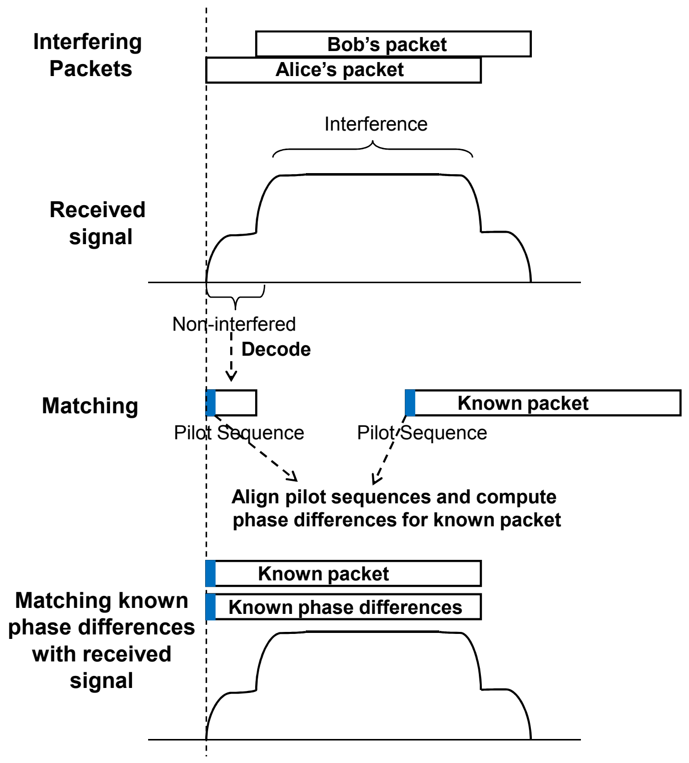


Figure 4-5: Aligning known phase differences with received signal. Alice finds where her packet starts using the pilot bits at the beginning of the packet, which are interference free. Bob, whose packet starts second, uses the pilot bits at the end of the packet and runs the alignment process backward.

ence free. Indeed, our approach enforces this incomplete overlap between the two packets to ensure that there are a few bits at the beginning and end of the interfered signal that are interference free, which can be used to synchronize. Specifically, we use a randomization scheme similar to 802.11 MAC. Nodes start their transmission after a *random delay*. They do this by picking a random number between 1 and 32, and starting their transmission in the corresponding time slot. The size of the slot is dependent on the transmission rate, packet size, modulation scheme used, etc.

Our solution attaches a known *pilot bit sequence* to the beginning of each packet. It also attaches a mirrored version of the pilot sequence to the end of the packet. The pilot is a 64-bit pseudo-random sequence. It is used to detect when exactly the known signal starts showing in the received signal.

We describe our solution assuming Alice's packet starts first. Bob's decoding algorithm is described in Sec. 4.5.4. Alice first detects the beginning of a packet using the *energy detector* from 4.5.1. She then looks for the known pilot sequence in the interference free part of the signal at the start of the packet. She decodes this part using standard MSK demodulation. Fig. 4-5 displays the matching process that Alice performs over the received signal. After decoding the interference free part, she tries to match the known pilot sequence with every sequence of 64 bits. Once a match is found, she aligns her known signal with the received signal starting at that point, i.e., starting at the end of the pilot. If Alice fails to find the pilot sequence, she drops the packet.

At the end of the pilot sequence, Alice starts applying the algorithm in 4.4 which detects the two interfering signals. By then Bob's signal might not have started yet. Despite this Alice can still apply our decoding algorithm from Section 4.4. The values for the initial estimated phase differences, $\Delta\phi[n]$ could be random and dependent on the noise since Bob's signal might not have started yet. Once Bob's signal starts, the estimated phases differences $\Delta\phi[n]$, will correspond to the pilot sequence at the start of Bob's packet. At that point, Alice detects the beginning of Bob's packet.

Thus, the pilot sequence helps Alice align her own sent signal with respect to the received signal. It also helps her detect the beginning of Bob's signal in the received signal.

■ 4.5.3 How does Alice know which packet to use to decode?

Alice keeps copies of the sent packets in a *Sent Packet Buffer*. When she receives a signal that contains interference, she has to figure out which packet from the buffer she should use to decode the interfered signal. Hence, we add a header after the pilot sequence that tells Alice the source, destination and the sequence number of the packet. Using the decoded header information, Alice can pick the right packet from her buffer to decode the interfered signal and get Bob's packet.

■ 4.5.4 How does Bob decode?

Bob's signal starts second in the interfered signal. Thus, he cannot blindly use the same decoding algorithm as Alice. Bob instead decodes the packet by running the decoding procedure backward. More precisely, he stores the received complex samples until the end of the packet, i.e., until the energy drops to the noise level. Then he runs the algorithm starting with the last sample and going backward in time. Our packets have the header and the pilot sequence both at the beginning and end, as shown in Fig. 4-6. Bob starts from the end of the packet, decodes the header and the pilot sequence there, discovers which

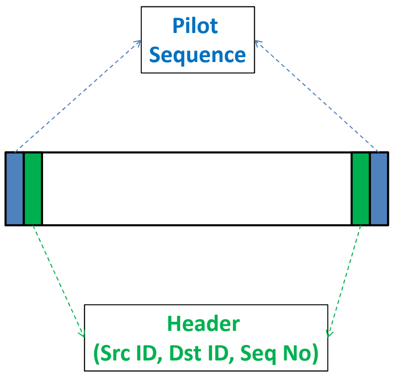


Figure 4-6: Frame Layout for Analog Network Coding

packet in his *sent packet buffer* to use to cancel the interference, and decodes Alice's packet backwards, using the interference decoding algorithm.

4.5.5 What does the router do?

In the Alice-Bob experiment, the router has to amplify the interfered signal it receives from Alice and Bob, and broadcast it. But in the chain topology, the router, N_2, has to decode the packet itself. Thus, the router needs to make a decision about what to do with an interfered signal. The router uses the headers in the interfered signal to discover which case applies. If either of the headers corresponds to a packet it already has, it will decodes the interfered signal. If none of the headers correspond to packets it knows, it checks if the two packets comprising the interfered signal are headed in opposite directions to its neighbors. If so, it amplifies the signal and broadcasts the interfered signal. If none of the above conditions is met, it simply drops the received signal.

Finally, Alg. 2 summarizes the interference decoding algorithm.

4.5.6 How to get the right packets to interfere?

We want to encourage interfering transmissions, from the right senders, i.e., those whose interfered signal can be correctly decoded at both destinations. To do so, we design a simple *trigger* protocol. To "trigger" simultaneous transmissions, a node adds a short trigger sequence at the end of a standard transmission. The trigger stimulates the right neighbors to try to transmit immediately after the reception of the trigger.[3] For example, in the Alice-Bob topology, the router adds the trigger sequence to the end of its transmission, triggering

[3]The nodes still insert the short random delay mentioned in 4.5.2.

2 Pseudocode for the Interference Decoding Algorithm

```
Use energy detector from  4.5.1 to detect signal reception
if Signal detected then
  Use variance detector from  4.5.1 to detect interference
  if Interfered Signal then
    Decode start and end of received signal to get both headers
    Discover whether my known signal starts first or second using the headers
    Lookup known packet from the headers
    Match phase differences of known signal with received signal using algorithm from
    4.5.2
    Decode packet using algorithm from  4.4
    Collect the decoded bits and frame it into a packet and pass it to the upper layers
  else
    Decode signal using normal MSK demodulation
    Collect the decoded bits and frame it into a packet and pass it to the upper layers
  end if
end if
```

both Alice and Bob to transmit. Alice and Bob respond by transmitting as soon as the transmission from the router ends. In the chain topology in Fig. 4-2, node N_2 triggers nodes N_1 and N_3 to transmit simultaneously by adding the appropriate trigger sequence to the end of its transmission. Thus, the triggering mechanism encourages positive interference that we can exploit to increase network capacity.

Clearly, for a node to trigger its neighbors to interfere, it needs to known the traffic flow in its local neighborhood. We assume that this information is provided via control packets that the nodes exchange.

In our context, the "trigger" protocol provides a simplified MAC for ANC. Designing a general MAC protocol for ANC depends on the environment in which it is used. For example, cellular networks already have strict scheduling-based MAC protocols (TDMA, CDMA etc). The trigger protocol for ANC in these networks can be easily integrated into the scheduling mechanism. In contrast, 802.11 wireless mesh networks use random access. In this case, short control sequences may be used as triggers. However, customizing the MAC protocol for ANC in 802.11 or other networks is beyond the scope of this thesis.

4.6 Capacity Analysis

To estimate the throughput gains expected from analog network coding, we analyze the capacity of the Alice-Bob network, shown in Fig. 4-1. We focus on the practical case, where

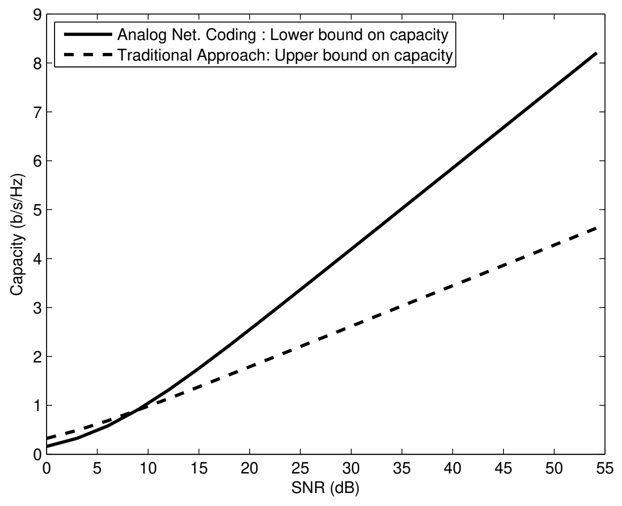


Figure 4-7: Capacity bounds as functions of SNR, for half-duplex nodes. At high SNRs, analog network coding doubles the throughput compared to traditional routing.

radios are half-duplex. The information theory literature refers to this network as a 2-way half-duplex relay channel.

Note that the capacity of a general wireless network is an open problem in information theory. In fact, the exact capacity of a 3-node relay network, that is, a source-destination pair with a router in the middle, is itself an open problem. The standard approach is to compute upper and lower bounds on the capacity of these networks, which is what we do in this section.

We compare the capacity of the Alice-Bob network, under analog network coding and the traditional routing approach. To do so, we compute an upper bound on the capacity under the traditional routing approach, and a lower bound on the capacity with our approach. Channel capacity depends on the received signal strength in comparison to the noise power at the receiver–i.e., a function of Signal-to-Noise Ratio (SNR). We compute our bounds for a wireless channel with additive white Gaussian noise. For simplicity, we assume the channel between Alice and the router is similar to the channel between Bob and the router, and all nodes transmit at the same power. When the router receives the interfered signal, it simply re-amplifies the signal and broadcasts it to Alice and Bob. We prove the following theorem:

Theorem 4.1 *An upper bound on the capacity of the traditional routing approach is given by:*

$$C_{traditional} = \alpha(log(1+2SNR)+log(1+SNR)),$$

and a lower bound on the capacity of analog network coding is given by:

$$C_{analog\ netcode} = 4\alpha log(1+\frac{SNR^2}{3SNR+1}),$$

where α is a constant. Thus, the capacity gain of analog network coding over the traditional approach asymptotically approaches 2 as the SNR increases.

Proof. Let $X_A(1),\dots,X_A(i),\dots,X_A(m)$ and $X_B(1),\dots,X_B(i),\dots,X_B(m)$ be Alice and Bob's complex symbols. We assume a line of sight channel which attenuates the signal and the presence of white noise at the receiver. We also assume that all nodes transmit with the same power P.

(1.) Routing: The Outer Bound

We first compute the outer bound for traditional routing. We assume that the network is time-shared fairly between Alice and Bob's flows. We analyze the capacity of the relay network using the cutset bound [26]. We assume that transmissions are asynchronous but the channel gains are known. Then, the capacity from Alice to Bob is upper-bounded by $min\{C_1, C_2\}$, where C_1 and C_2 are given by:

$$\begin{aligned} C_1 &= \arg \max_{\rho, 0\le\rho<1} \tfrac{1}{4}\log(1+(h_{AB}^2+h_{RA}^2)P)+\tfrac{1}{4}\log(1+(1-\rho^2)h_{AB}^2P) \\ C_2 &= \arg \max_{\rho, 0\le\rho<1} \tfrac{1}{4}\log(1+(h_{AB}^2+h_{RB}^2)P+2\rho P\sqrt{h_{AB}^2h_{RB}^2})+\tfrac{1}{4}\log(1+h_{AB}^2P) \end{aligned} \tag{4.20}$$

Similar equations exist for Bob, with h_{RA} and h_{RB} interchanged.

(2.) Analog Network Coding: The Inner Bound

We compare the outer bound for traditional routing with an achievable inner bound for ANC. In this case the signal received at the relay can be written as,

$$Y_R[n] = \sqrt{2P}h_{AR}X_A[n] + \sqrt{2P}h_{BR}X_B[n] + Z_R[n] \tag{4.21}$$

where h_{AR} and h_{BR} are the attenuations on the links Alice-Relay and Bob-Relay respectively. P is the transmission power and Z_R is the noise at the relay.

The relay amplifies the signal and broadcasts it to Alice and Bob. Let the amplification factor be A. The signal received at Alice is:

$$\begin{aligned} Y_A[n] &= Ah_{RA}Y_R[n] + Z_A[n] \\ &= A(\sqrt{2P}h_{RA}(h_{AR}X_A[n] + h_{BR}X_B[n]) + h_{RA}Z_R[n]) + Z_A[n] \end{aligned} \tag{4.22}$$

where Z_A is the noise at Alice, and h_{RA} is the attenuation on the link from the relay to Alice. The amplification factor A is set such that the power is still equal to P. Therefore $A = \sqrt{P/(Ph_{AR}^2 + Ph_{BR}^2 + 1)}$

Assuming Alice perfectly knows the attenuations h_{RA} and h_{AR}, she can cancel her signal out and get:

$$Y_A^{'}[n] = A\sqrt{2P}h_{RA}h_{BR}X_B[n] + Ah_{RA}Z_R[n] + Z_A[n] \tag{4.23}$$

For simplicity, we assume that all noise powers are the same and equal to 1. Thus, the SNRs of the received signal at Alice and Bob can be computed as:

$$SNR_{Alice} = \frac{A^2Ph_{RA}^2h_{BR}^2}{(A^2h_{RA}^2+1)}, \quad SNR_{Bob} = \frac{A^2Ph_{RB}^2h_{AR}^2}{(A^2h_{RB}^2+1)} \tag{4.24}$$

Thus the total throughput of the system is given by,

$$C_{anc} = \frac{1}{2}(\log(1 + SNR_{Alice}) + \log(1 + SNR_{Bob})) \tag{4.25}$$

Assuming symmetric channels, it can be shown [71] that the ratio C_{anc}/C_r, where C_r is the routing throughput tends to 2 as $P \to \infty$ since the ratio $\frac{\log(1+x)}{\log(1+kx)} \to 1$ as $x \to \infty$. ■

Fig. 4-7 illustrates the capacity bounds for analog network coding and the traditional approach. The figure shows two SNR regions with different characteristics.

(a) Moderate to High SNR: At medium-to-high SNR, analog network coding almost doubles the throughput when compared to the traditional routing approach. At these SNRs, the gain is primarily dominated by the reduction in the number of time slots needed to send the packets (from 4 to 2).

(b) Low SNR: In contrast, at low SNRs around 0-8dB, the throughput of analog network coding is lower than the upper bound for the traditional approach. This is because when the router amplifies and broadcasts the interfered signal to Alice and Bob, it also amplifies

the noise that the channel adds to the interfered signal. At low SNR, this amplified noise has a deleterious effect at Alice and Bob, since the transmission power is quite low.

Note, however, that practical wireless systems typically operate around 20-40dB. The low SNR region is not used because it is hard to design practical receivers that decode at such low power. For example, indoor WLANs typically operate at SNR around 15-40dB. So, for most practical cases, analog network coding has a theoretical throughput gain of $2x$ for the Alice-Bob network.

■ 4.7 Implementation

We have implemented ANC using Software Defined Radios (SDR). SDRs implement all the signal processing components (source coding, modulation, clock recovery etc) of a wireless communication system entirely in software. The hardware is a simple radio frequency (RF) frontend, which acts as an interface to the wireless channel. The RF frontend passes the complex samples generated by the SDR to the Digital to Analog Converter (DAC), which produces the analog signal. The upconverter converts the output of the DAC to the carrier frequency and transmits it over the wireless channel. At the receiver side, the process is inverted. First, the downconverter converts the received signal to its baseband frequency and passes it to an Analog to Digital Converter (ADC). The discrete samples produced by the ADC are converted into complex numbers and passed to the SDR software.

We use the Universal Software Radio Peripheral (USRP) [59] as our RF frontend. The software for the signal processing blocks is from the open source GNURadio project [41]. USRP is a generic RF frontend developed specifically for the GNURadio SDR. The USRP connects to the PC via USB 2.0. Thus, its throughput is limited to 32MB/s. This also limits the bandwidth of the signal to at most 4MHz, which is enough for most narrowband data transmission.

■ 4.8 System Architecture

We export a network interface to the user, which can be treated like any other network device (e.g., eth0). Fig. 4-8 abstracts the system below the network interface.

On the sending side, the network interface pushes the packets to the *Framer*, which adds the pilot sequence and the header to the packet as described in 4.5.2 and 4.5.4, and

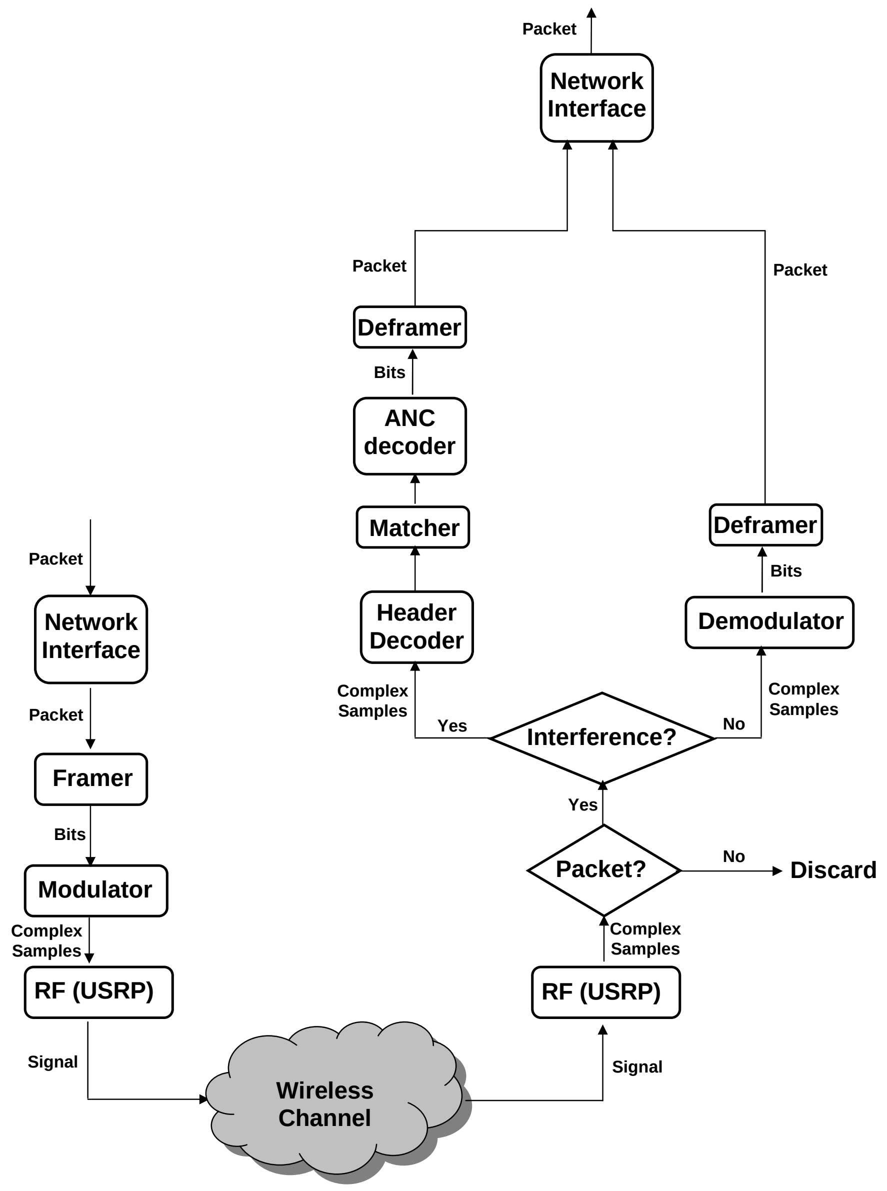


Figure 4-8: Flow chart of our implementation.

creates a frame. The framer also stores a copy of the frame, which could be used later for decoding interfered packets. Next, the modulator encodes the bit sequence to create complex samples of the signal. It pushes the samples to the USRP RF frontend, which transmits them on the channel.

On the receiving side, we continuously get complex samples from the USRP. The *Packet Detector* checks whether the received samples constitute a packet or just noise. If a packet is being received, the *Interference Detector* checks whether the packet has been subjected to interference using the interference detection algorithm described in 4.5.1. If no interference is detected, standard MSK demodulation is performed to decode the bits. The bits are then passed through the *Deframer*, which converts them into a packet and forwards the packet to the network interface.

If interference is detected, the received complex samples are passed to the *Header Decoder* which detects the pilot sequences and the headers at the start and end of the set of complex samples constituting the interfered packet. From the headers it discovers which two packets constitute the interfered packet and checks if it can be decoded or should be re-amplified and broadcast. If it can be decoded, the complex samples are passed on to the next module, the *Phase Difference Matcher* which looks up the known packet, and matches the phase differences of the known packet with the received interfered signal. Once the matching is done, the complex samples are passed through the *ANC Decoder*, which decodes the unknown bits out of the interfered signal. The bits are then passed through the *Deframer*, which converts them into a packet and pushes the recovered packet out on the network interface.

■ 4.9 Experimental Evaluation

This section uses results from a software radio testbed to study the performance of our approach. We run our experiments on three canonical topologies: the Alice-Bob topology in Fig. 4-1, the "X" topology in Fig. 4-11, and the chain topology in Fig. 4-2. These topologies form the basis for larger networks and provide examples of both 2-way and unidirectional traffic.

■ 4.9.1 Compared Approaches

We compare ANC against two other approaches.

(a) No Coding (Traditional Routing Approach): We implement traditional routing but with an optimal MAC, i.e., the MAC employs an optimal scheduler and benefits from knowing the traffic pattern and the topology. Thus, the MAC never encounters collisions or backoffs, and hence outperforms the conventional carrier sense based MAC.

(b) Digital Network Coding (COPE): We compare against packet-based network coding whenever applicable. We use the COPE protocol described in Chapter 3 as an example network coding protocol. Again we implement an optimal MAC that schedules transmissions knowing the traffic pattern and the topology.

Since the MAC is optimal for all three designs, the differences between them are due to their intrinsic characteristics rather than a sub-optimal MAC.

4.9.2 Metrics

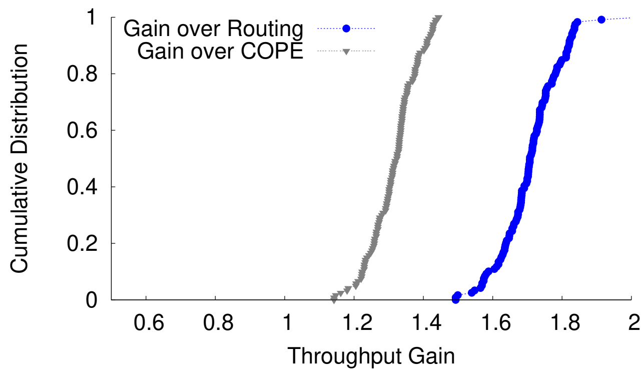
We use the following metrics.

- *Network Throughput:* This is the sum of the end-to-end throughput of all flows in the network. Note that ANC has a higher bit error rate than the other approaches and thus needs extra redundancy in its error-correction codes. We account for this overhead in our throughput computation.
- *Gain Over Routing:* This is the ratio of the network's throughput in ANC to the network's throughput in the traditional routing approach for two consecutive runs in the same topology and for the same traffic pattern.
- *Gain Over COPE:* This is the ratio of the network's throughput in ANC to the network's throughput in COPE for two consecutive runs in the same topology and for the same traffic pattern.
- *Bit Error Rate (BER):* The percentage of erroneous bits in an ANC packet, i.e., a packet decoded using our approach.

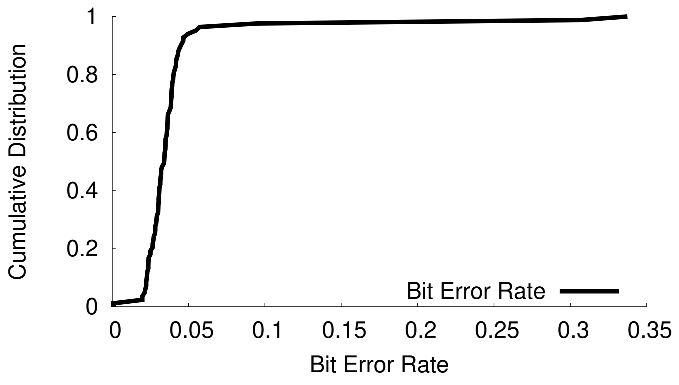
4.9.3 Summary of Results

Our experiments reveal the following findings:

- ANC provides significant throughput gains. For the Alice-Bob topology, ANC increases the network's throughput by 70% compared to the traditional routing approach. Compared to COPE the throughput increases by 30%.
- ANC improves the throughput for the "X" topology by 65% when compared to the traditional routing approach, and 28% when compared to COPE.
- For unidirectional flows in the chain topology, ANC improves throughput by 36% when compared to the traditional routing approach. (COPE does not apply to this scenario.)



(a) CDF of throughputs for Alice-Bob topology



(b) CDF of BERs for Alice-Bob topology

Figure 4-9: Results for the Alice-Bob topology: ANC has 70% average throughput gain over the traditional routing approach and 30% over COPE. The average BER is around 4%, which can be easily corrected by a small amount of error correcting codes.

- Differences between the theoretical gains of ANC and its practical gains are dominated by imperfect overlap between interfering packets, where only 80% of the two packets interfere on average.

- We evaluate ANC's sensitivity to the relative strength of the two interfering signals. On URSP software radios, our decoding algorithm works with signal to interference ratio as low as $-3dB$. In contrast, typical interference cancellation schemes require a signal to interference ratio of $6dB$ [53]. (Note that these schemes do not use ANC and cannot achieve our capacity gains, as explained in 4.11.)

4.9.4 Alice-Bob topology

We compare ANC to both the traditional routing approach and COPE over the Alice-Bob topology in Fig. 4-1. Each run transfers 1000 packets in each direction, first using ANC, then using the traditional routing approach, and last using COPE. We repeat the experiment 40 times and plot the results in Fig. 4-9.

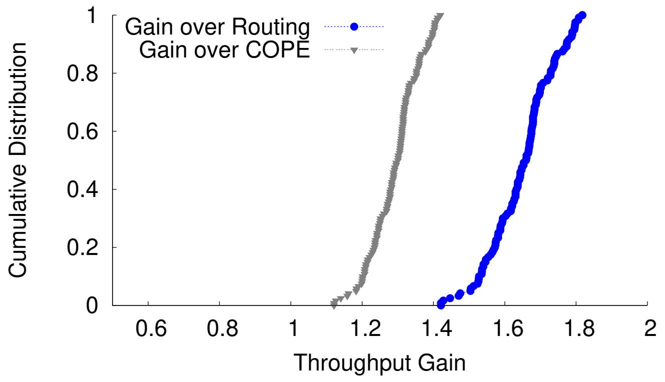
Fig. 4-9(a) plots the CDF of ANC's throughput gain over the traditional routing approach and COPE. The figure shows that ANC's average gain is 70% compared to the traditional routing approach and 30% compared to COPE.

Our practical throughput gains are significant, but less than the theoretical optimum. Theoretically, ANC doubles the throughput compared to routing and provides 50% gain over COPE. Practical gains are lower due to two reasons. First, the theoretical computation assumes that packets interfere perfectly, i.e., it assumes that Alice and Bob are perfectly synchronized. In practice, the average overlap between Alice's packets and those from Bob's is 80%. The imperfect overlap is due to the *random delay* our protocol introduces so that the pilot sequences are interference free. Further, because our implementation runs in user-space, there is significant jitter in how fast Alice and Bob transmit after receiving the "trigger" from the router. We believe that with a kernel-space implementation, one could get higher overlap in the packets and consequently higher gains.

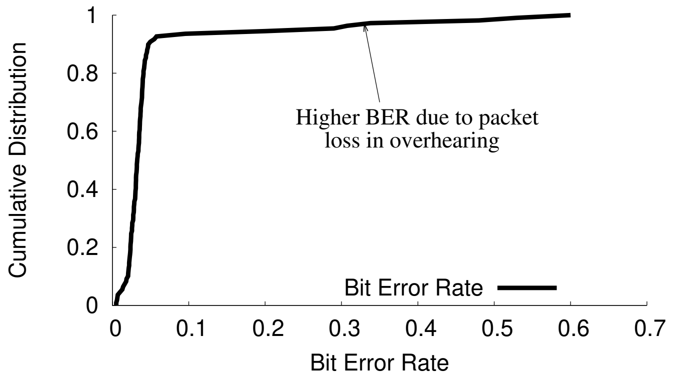
The second factor affecting ANC's practical gains is the non-zero bit error rate. Fig. 4-9(b) plots the CDF of bit error rates for Alice and Bob, when using our approach. The bit error rate is computed by decoding the packet from the interfered signal and then comparing it against the payload that was sent. The bit error rate for most packets is less than 4%. To compensate for this bit-error rate we have to add 8% of extra redundancy (i.e., error correction codes) compared to the traditional routing approach. This overhead is another reason why the practical gains are a little lower than the theoretical gains.

4.9.5 "X" Topology

Next, we evaluate ANC over the "X" topology in Fig. 4-11. This topology is analogous to the Alice-Bob, but in contrast to Alice who knows the interfering signal because she has generated it, the receivers in the "X" topology know the interfering signal because they happen to overhear it while snooping on the medium. In particular, S1 and S2 are sending to D1 and D2, respectively. Node D2 can overhear S1's transmission, and similarly D1



(a) CDF of throughputs for "X" topology



(b) CDF of BERs for "X" topology

Figure 4-10: Results for the X topology. Our approach provides an average of 65% gain over the traditional routing approach and 28% over traditional network coding. It is slightly less than the Alice-Bob topology due to packet losses in overhearing. The BERs for the experiments where there were packet losses in overhearing is correspondingly higher.

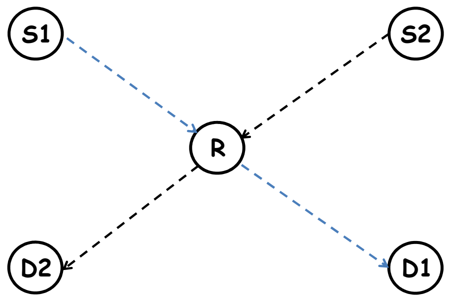


Figure 4-11: "X" topology with two flows intersecting at node R.

can overhear S2's transmission. Thus, we make S1 and S2 transmit simultaneously. The router R amplifies and retransmits the interfered signal to the destinations D1 and D2. The destinations use the overheard packets to cancel the interference and decode the packets they want.

Fig. 4-10(a) plots the CDF of throughput gains for the "X" topology. The figure shows that ANC provides a 65% increase in throughput compared to the traditional routing approach, and a 28% increase in throughput compared to COPE.

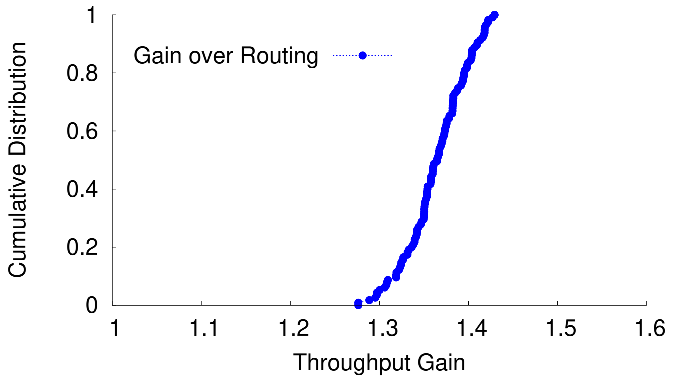
As expected, practical gains are lower than theoretical gains. Theoretically, ANC doubles the throughput when compared to the traditional routing approach, and increases the throughput by 50% when compared to COPE. The reasons for the difference between practical and theoretical gains are fairly similar to the Alice-Bob case. First, packets do not overlap perfectly. Second, the decoded packets have a non-zero BER, hence extra redundancy is required. There is, however, an additional error factor in the "X" topology, namely, imperfect decoding of overheard packets. Theoretical gains assume that when S1 transmits, D2 overhears the packet and correctly decodes it. This is not always true and D2 sometimes fails in decoding the overheard packets, particularly because node S2 is transmitting too; hence node D2's reception faces additional interference. When a packet is not overheard, the corresponding interfered signal cannot be decoded either. The same reason holds for node D1 overhearing S2's transmission. Hence, the throughput gain is slightly lower.

■ 4.9.6 Unidirectional Traffic: Chain Topology

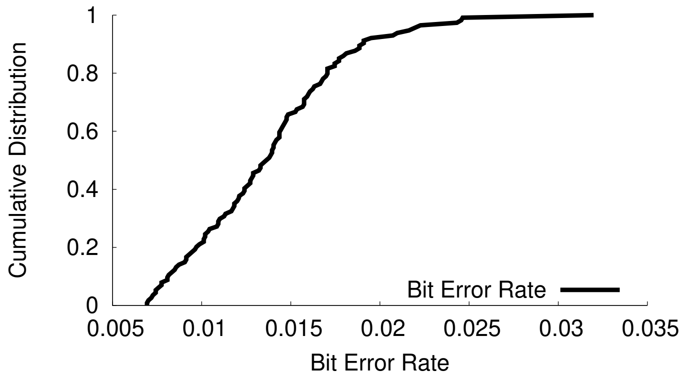
Unlike COPE, analog network coding is useful even when the flows are uni-directional. To demonstrate these gains, we evaluate our approach in the chain topology shown in Fig. 4-2, where traffic is flowing from node S to node D.

Figure. 4-12(a) plots the CDF of the throughput gains with our approach compared to the traditional routing approach. ANC increases the throughput by 37% on average.

Note that for the chain topology, the throughput gain is close to the theoretical prediction. Theoretically, ANC has a gain of 50%, since it reduces the number of time slots required to deliver a packet on average from 3 to 2. The slight loss in gain is due to the same factors as before. Packets do not overlap perfectly and we have to provision for extra redundancy to correct for the slightly higher bit error rate. But interestingly, the bit error rate is lower for the chain than for the other topologies. Fig. 4-12(b) plots the BER CDF at node R1. The average bit error rate is 1%, which is significantly lower than the 4% bit error observed in the Alice-Bob topology. This is because in the chain, decoding is done



(a) CDF of throughputs for chain topology



(b) CDF of BERs for chain topology

Figure 4-12: Results for the chain topology. Our approach provides an average of 36% gain over the traditional approach. The average BER is 1.5%, which is lower than the Alice-Bob topology since here the router directly decodes the interfered signal and does not amplify and broadcast it.

at the node that first receives the interfered signal. In the Alice-Bob case, the interference happens at the router, but the router has to then amplify and broadcast the signal to Alice and Bob. This also amplifies the noise in the interfered signal, resulting in higher bit error rates at Alice and Bob.

4.9.7 Impact of relative signal strengths

Is ANC's decoding algorithm sensitive to the relative signal strengths of the interfering signals? For example, does Alice's ability to decode Bob's packet depend on the relative received strengths of Alice's and Bob's signals? We evaluate this by calculating the bit error rate for the decoded packet as the relative signal strengths vary.

In order to quantify the relative signal strengths of Alice and Bob's signals we define

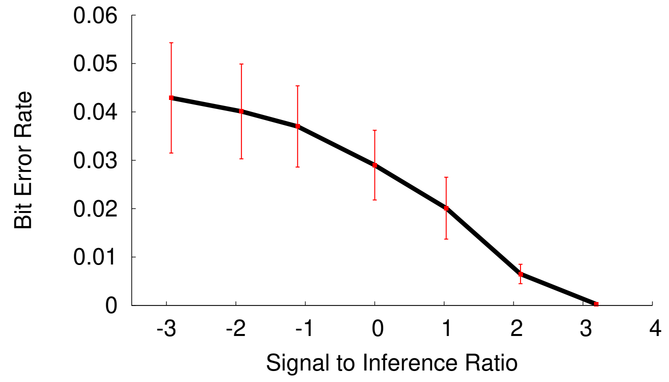


Figure 4-13: BER vs. Signal-to-Interference Ratio (SIR) for decoding at Alice. Even for low SIR, i.e., when the signal Alice wants to decode has relatively low signal strength, the BER is less than 5%.

Signal to Interference Ratio (SIR),

$$SIR = 10\log_{10}(\frac{P_{Bob}}{P_{Alice}}) \tag{4.26}$$

where P_{Bob} and P_{Alice} are the received powers for Bob's and Alice's signals respectively at Alice. The intuition behind this definition is simple, since Alice wishes to decode Bob's packet, her own signal which is mixed up with Bob's signal is treated as interference.

We vary Bob' transmission power, while Alice's power is kept constant. Fig. 4-13 plots the BER of the decoded Bob's packet as a function of the received SIR at Alice. Even when Bob's signal strength is half that of Alice's signal, i.e., for a SIR of -3dB, the BER is less than 5%. When the signals are of equal strength (SIR = 0dB), the BER drops to 2%. At the other end of the spectrum, when Bob's signal is twice as strong as Alice's signal, the BER drops to 0.

Hence, our approach works very well even when the relative signal strengths are vastly different or the same. Prior work on blind signal separation usually works only when the signal being decoded has a SIR of 6dB [53], i.e., the signal being decoded is four times stronger than the signal interfering with it. On the other hand our ability to leverage already known network level information allows us to decode signals that have very low signal strength compared to the signal that is interfering with it.

■ 4.10 Extending ANC to Other Modulation Schemes

How can a node perform analog network coding with a different modulation scheme? In this section we outline a general algorithm for disentangling a known signal from an interfered signal which works for most of the commonly used modulation schemes. The decoding algorithm can be summarized as follows: estimate the channel and system transformation, apply the transformation to the known signal, subtract the transformed known signal from the interfered signal and finally decode using a standard decoder. We describe each component briefly below, using the same Alice and Bob example discussed earlier in the chapter.

■ 4.10.1 Estimating the Channel Transformation

Both Alice and Bob have to first estimate the channel parameters from the colliding packet's preamble. We will focus on Alice; the discussion for Bob will be similar. We need to estimate three values: the channel attenuation, the frequency offset and the sampling offset.

Estimating Channel Filter

The preamble of a packet has a known pseudorandom sequence, the *access code*. We can use the pattern to estimate the attenuation for Alice h'. The trick is to correlate the samples from the received preamble with the known sequence. This correlation C when the sequences are perfectly aligned can be shown to be:

$$C = h' \sum_{i=1}^{L} |s[i]|^2 \tag{4.27}$$

Due to the property of pseudorandom sequences, the value of C will be maximum when the known and the received sequences are perfectly aligned. Alice can use this to estimate the correlation above, by simply trying different alignments and using the maximum value obtained. From there, she can easily compute h' since $\sum_{i=1}^{L} |s[i]|^2$ is already known.

Estimating Frequency Offset

It is virtually impossible to manufacture two radios centered at the same exact frequency. Hence, there is always a small frequency difference, δf, between transmitter and receiver. The frequency offset causes a linear displacement in the phase of the received signal that increases over time, i.e.,

$$y[n] = h'x[n]e^{j2\pi n\delta fT} + w[n] \tag{4.28}$$

The frequency offset does not change significantly over the length of a packet. Traditional decoders already use standard algorithms [97] to estimate this offset, we can leverage these algorithms to get an estimate of the frequency offset at Alice, δf_A.

Sampling Offset

The transmitted signal is a sequence of complex samples separated by a period T. However, when transmitted on the wireless medium, these discrete values have to be interpolated into a continuous signal. The continuous signal is equal to the original discrete samples, only if sampled at the exact same positions where the discrete values were. Due to lack of synchronization, a receiver cannot sample the received signal exactly at the right positions. There is always a sampling offset, μ. Further, the drift in the transmitters and receivers clocks results in a drift in the sampling offset. Hence, decoders have algorithms to estimate μ and track it over the duration of a packet. Alice can run the traditional decoder over the uninterfered part of the signal to estimate μ_A, and use that estimate throughout the rest of the packet.

■ 4.10.2 Re-encoding the Known Packet

Once Alice knows the channel parameters, she uses them to apply the same transformation to the known packet, to get an estimate of how the known packet would have looked like after traversing the channel. She can subtract this estimate from the interfered signal, to get Bob's signal.

There are three parts to the transformation: applying the channel filter, the frequency offset and the sampling offset. Let $x_A[1], \ldots, x_A[K]$ denote the original symbols in Alice's known packet. Assuming we have estimated the channel filter h' and frequency offset δf_A, the original symbol $x_A[n]$ after being transformed can be estimated as

$$y_A[n] = h' x_A[n] e^{j2\pi\delta f_A T} \tag{4.29}$$

But we still have to account for the sampling offset. Alice's ADC samples the interfered signal μ_A seconds away from Alice's samples. Thus, given the samples $y_A[1], \ldots, y_A[K]$, Alice has to interpolate to find the samples at $y_A[1+\mu_A], \ldots, y_A[K+\mu_A]$. To do so, we observe that real-world wireless signals are band-limited. Hence by Nyquist's theorem, we can interpolate the signal at any point of time, using the *sinc* function as basis functions. Specifically,

$$y_A[n+\mu_A] = \sum_{i=-\infty}^{i=+\infty} y_A[i] sinc(\pi(n+\mu_A-i)) \tag{4.30}$$

In practice, due to the decaying nature of the *sinc* function, we only need to take the summation over a few symbols around the neighborhood of the sample of interest.

■ 4.10.3 Subtracting and Decoding

Once Alice has estimated her transformed known samples, the rest of the process is straightforward. She subtracts the estimate $y_A[n]$ from the received interfered signal, and passes the resulting signal through the normal decoder for the modulation scheme used by Bob. Thus the decoder is a blackbox, and ANC operates independent of it. This also implies that ANC can be used even if Alice and Bob use different modulation schemes.

■ 4.11 Related Work

Prior work falls into three main categories: theoretical work on the capacity of the 2-way relay, schemes for addressing interference, and traditional network coding.

(a) Theoretical Work on the Capacity of the 2-Way Relay Network: ANC builds on prior work on the capacity of the 2-way relay channel (i.e., the Alice-Bob topology) [117, 95] and a recent paper on physical network coding [137]. Prior work, however, is focused on theoretical bounds and thus allows impractical assumptions. First, it assumes that the interfering signals are synchronized at symbol boundaries. Second, it assumes the channel functions are known a priori and do not change. Third, it assumes that different channels do not introduce different phase-shifts, which is unlikely given the propagation delays in the channels. In contrast, ANC make no such assumptions. We further implement our design in a software-radio testbed, showing the practicality of our approach.

(b) Addressing Interference: Typically, wireless networks try to avoid interference by probing the medium for idleness [102], scheduling senders in different time slots [124], or using small control packets called RTS-CTS [9]. Our work allows correct reception despite interference.

Multiple access techniques like CDMA [104], FDMA [124], and spatial reuse [124] allow multiple transmissions at the same time. These approaches, however, are simply means to avoid interference in time, code, space, or frequencies. They just divide channel capacity among multiple users. In contrast, analog network coding expands the capacity of the network.

Co-operative diversity [80, 79] and MIMO systems [124] allow multiple concurrent transmissions. There are two main differences between this work and analog network coding. First, this work is for point-to-point communication (i.e., one-hop), while ours is for a multi-hop network. Second, this work usually assumes antenna arrays and coherent combining at the receiver. Our work makes no such assumptions.

Also some related work falls in the area of interference cancellation and blind signal separation. These schemes decode two signals that have interfered without knowing any of the signals in advance [126, 16, 53]. Practical work in this domain, however, is limited to signals that differ significantly in their characteristics. They usually assume that the wanted signal has much higher power than the signal they are trying to cancel out. Our technique makes no such assumptions. Further, the prior work does not increase achievable network capacity. Analog network coding increases the achievable capacity of the network due to its ability to exploit network layer information.

(c) Traditional Network Coding: As discussed before, work on network coding started with a paper by Ahlswede et al. which establishes the benefits of coding in routers and bounds the capacity of such networks [2]. This work has been extended by papers on linear network codes [85, 77, 63], randomized coding [58], wireless network coding [36, 107, 130, 100], and network coding for content distribution [47]. All of the above mix bits in routers or hosts. In contrast, analog network coding makes the senders transmit concurrently and has the wireless channel mix the analog signals representing the packets. We further show that analog network coding achieves higher throughput than traditional network coding and applies to scenarios that cannot benefit from traditional network coding.

■ 4.12 Discussion

Finally, we would like to comment on the scope of ANC. The present design targets stationary wireless mesh networks with a few hardware modifications, or ones with software radios which usethe ANC software module. ANC does not need any modifications to the major hardware components, such as the oscillator, ADC etc. It does require some memory to store samples (on the order of a few KB), and modifications to the FPGA to execute the ANC decoding algorithm.

One of the important observations from our work on ANC, is a different approach to dealing with interference. Traditionally, radios have tried to decode interfered signals, treating the interfering signal as noise, and giving up on the reception if the interfering power is too high. ANC demonstrates that this is a pessimistic approach; often in wireless, due to broadcast and previous transmissions, nodes have side information about the interfering signal. In ANC, this is in fact a clean copy of the packet corresponding to the interfering signal. In many other scenarios, other forms of side-information could be available, such as another collided packet with the same set of interfering signals [48], a noisy version of the interfering signal etc. Smart decoding algorithms can exploit this side-information to significantly improve their ability to decode interfered signals, and increase throughput.

Subsequent work [48, 52] has applied the philosophy of analog network coding to deal with hidden terminals in wireless networks. Those works are complementary to analog network coding and they can be combined to provide higher throughput gains. Further, this subsequent work also simplifies the MAC protocol design problem for ANC, since receivers can decoded collided packets even if they do not already have one of the packets making up the collision, albeit by either requesting that senders transmit at a lower rate or requesting retransmissions.

CHAPTER 5
MIXIT: Symbol-level Network Coding

After demonstrating how network coding at two extreme granularities- the packet and the signal level - can be exploited to improve performance, this chapter takes the middle ground. We present the design, implementation and evaluation of MIXIT, which uses symbol-level network coding to build a system that significantly improves the throughput of a wireless mesh network compared to the best current approaches. In both traditional routing protocols as well as more recent opportunistic approaches [12, 18], an intermediate node forwards a packet only if it has no errors. In contrast, MIXIT takes a much looser approach: a forwarding node does not attempt to recover from any errors, or even bother to apply an error detection code (like a CRC).

Somewhat surprisingly, relaxing the requirement that a node only forward correct data improves throughput. The main reason for this improvement is a unique property of dense wireless mesh networks: *Even when no node receives a packet correctly, any given bit is likely to be received correctly by* some *node.*

In MIXIT, the network and the lower layers collaborate to improve throughput by taking advantage of this observation. Rather than just send up a sequence of bits, the PHY annotates each bit with SoftPHY hints [65] that reflect the PHY's confidence in its demodulation and decoding. The link layer passes up frames to the network layer with these annotations, but does not try to recover erroneous frames or low-confidence bits using link-layer retransmissions. Instead, the network layer uses the SoftPHY hints to filter out the bits with low confidence in a packet, and then it performs opportunistic routing on groups of high confidence bits.

The core component of MIXIT is a new network code that allows each link to operate at a considerably high bit-error rate compared to the status quo without compromising end-to-end reliability. Unlike COPE and ANC, the network code operates at the granularity of *symbols* [1] rather than packets: each router forwards (using radio broadcast) random linear combinations of the high-confidence symbols belonging to different packets. Thus, a MIXIT router forwards symbols that are likely to be correct, tries to avoid forwarding symbols that are likely to be corrupt, but inevitably makes a few incorrect guesses and forwards corrupt symbols.

MIXIT's network code addresses two challenges in performing such symbol-level opportunistic routing over potentially erroneous data. The first problem is *scalable coordination*: the effort required for nodes to determine which symbols were received at each node to prevent duplicate transmissions of the same symbol is significant. MIXIT uses the randomness from the network code along with a novel dynamic programming algorithm to solve this problem and scalably "funnel" high-confidence symbols to the destination, compared to a node co-ordination based approach like ExOR [12].

The second problem is *error recovery*: because erroneous symbols do seep through, the destination needs to correct them. Rather than the traditional approach of requesting explicit retransmissions, MIXIT uses a rateless end-to-end error correcting component that works in concert with the network code for this task. The routers themselves only forward random linear combinations of high-confidence symbols, performing no error handling.

MIXIT incorporates two additional techniques to improve performance:

- **Increased concurrency:** MIXIT takes advantage of two properties to design a channel access protocol that allows many more concurrent transmissions than CSMA: first, entire packets need not be delivered correctly to a downstream node, and second, symbols need to be delivered correctly to *some* downstream node, not a specific one.

- **Congestion-aware forwarding:** Unlike previous opportunistic routing protocols which do not consider congestion information [18, 12], MIXIT forwards coded symbols via paths that have both high delivery probabilities and small queues.

MIXIT synthesizes ideas from opportunistic routing (ExOR [12] and MORE [18]) and

[1] A symbol is a small sequence of bits (typically a few bytes) that the code treats as a single value.

partial packet recovery [65], noting the synergy between these two concepts. Prior opportunistic schemes [18, 12] often capitalize on sporadic receptions over long links, but these long links are inherently less reliable and likely to exhibit symbol errors. By insisting on forwarding only fully correct packets, prior opportunistic protocols miss the bulk of their opportunities. Similarly, prior proposals for exploiting partially correct receptions, like PPR [65], SOFT [132], and H-ARQ [25], limit themselves to a single wireless hop, incurring significant overhead trying to make that hop reliable. In contrast, we advocate eschewing reliable link-layer error detection and recovery altogether, since it is sufficient to funnel opportunistically-received correct symbols to their destination, where they will be assembled into a complete packet.

We evaluate MIXIT using our software radio implementation on a 25-node testbed running the Zigbee (802.15.4) protocol. The main experimental results are as follows:

- MIXIT achieves a $2.8\times$ gain over MORE, a state-of-the-art packet-based opportunistic routing protocol under moderate load. The gain over traditional routing is even higher, $3.9\times$ better aggregate end-to-end throughput. At lighter loads the corresponding gains are $2.1\times$ and $2.9\times$.

- MIXIT's gains stem from two composable capabilities: symbol-level opportunistic routing, and higher concurrency, which we find have a multiplicative effect. For example, separately, they improve throughput by $1.5\times$ and $1.4\times$ over MORE; in concert, they lead to the $2.1\times$ gain.

- Congestion-aware forwarding accounts for 30% of the throughput gain at high load.

MIXIT is the first system to show that routers need not forward fully correct packets to achieve end-to-end reliability, and that loosening this constraint significantly increases throughput. MIXIT realizes this vision using a layered architecture which demonstrates cross-layer collaborations using clean interfaces: the network code can run atop any radio and PHY that provides SoftPHY hints, the system can run with any MAC protocol (though ones that aggressively seek concurrency perform better), and the routers are oblivious to the error-correcting code. This modular separation of concerns eases implementation.

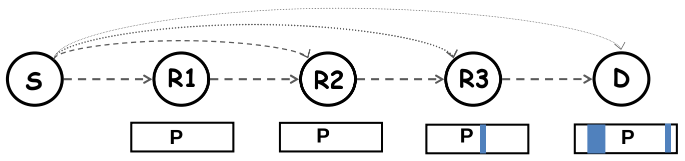


Figure 5-1: Example of opportunistic partial receptions: The source, S, wants to deliver a packet to the destination, D. The figure shows the receptions after S broadcasts its packet, where dark shades refer to erroneous symbols. The best path traverses all routers $R1$, $R2$ and $R3$. Traditional routing makes $R1$ transmit the packet ignoring any opportunistic receptions. Packet-level opportunistic routing exploits the reception at $R2$ but ignores that most of the symbols have made it to $R3$ and D. MIXIT exploits correctly received symbols at R3 and D, benefiting from the longest links.

5.1 Motivating Examples

This section discusses two examples to motivate the need for mechanisms that can operate on symbols that are likely to have been received correctly (i.e., on partial packets). These examples show two significant new opportunities to improve throughput: far-reaching links with high bit-error rates that allow quick jumps towards a destination *even when they might never receive entire packets correctly*, and increased concurrency using a more aggressive MAC protocol that induces higher bit-error rates than CSMA. The underlying theme in these examples is that one can improve throughput by allowing, and coping with, higher link-layer error rates.

First, consider Fig. 5-1, where a source, S, tries to deliver a packet to a destination, D, using the chain of routers $R1$, $R2$, and $R3$. It is possible that when the source broadcasts its packet, $R1$ and $R2$ hear the packet correctly, while $R3$ and D hear the packet with some bit errors. Traditional routing ignores the "lucky" reception at $R2$ and insists on delivering the packet on the predetermined path, i.e., it makes $R1$ forward the packet to $R2$ again. In contrast, recent opportunistic routing protocols (such as ExOR) capitalize on such lucky receptions (at $R2$) to make long jumps towards the destination, saving transmissions.

By insisting on forwarding fully correct packets, however, current opportunistic protocols miss a large number of opportunities to save transmissions and increase throughput; in particular, they do not take advantage of all the correct bits that already made it to $R3$ and even to the destination, D. Moreover, because of spatial diversity [96, 124], the corrupted bits at $R3$ and D are likely in different positions. Thus, $R3$ has to transmit only the bits that D did not receive correctly for the destination to get the complete packet. A scheme that can identify correct symbols and forward them has the potential to signifi-

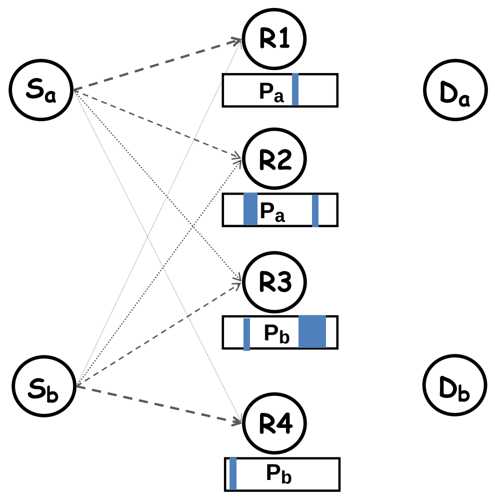


Figure 5-2: Concurrency example: The figure shows the receptions when the two sources transmit concurrently. Without MIXIT, the two sources S_a and S_b cannot transmit concurrently. MIXIT tolerates more bit errors at individual nodes, and hence is more resilient to interference, increasing the number of useful concurrent transmissions.

cantly reduce the number of transmissions required to deliver a packet.

Next, consider an example with potential concurrency as in Fig. 5-2, where two senders, S_a and S_b, want to deliver a packet to their respective destinations, D_a and D_b. If both senders transmit concurrently, the BER will be high, and no router will receive either packet correctly. Because current opportunistic routing protocols insist on correct and complete packets, the best any MAC can do is to make these senders transmit one after the other, consuming two time slots.

But interference is not a binary variable. In practice, different routers will experience different levels of interference; it is likely the routers close to S_a will receive packet, P_a, with only a few errors, while those close to S_b will receive packet P_b, with only some errors. A scheme that can identify which symbols are correct and forward only those groups of bits can exploit this phenomenon to allow the two senders to transmit concurrently and increase throughput. It can then "funnel" the correct symbols from the routers to the destination.

MIXIT aims to realize these potential benefits in practice. It faces the following challenges:

- How does a router classify which symbols in each received packet are likely correct?

- Given the overlap in the correct symbols at various routers, how do we ensure that routers do not forward the same information, wasting bandwidth?

- How do we avoid creating hotspots?
- When is it safe for nodes to transmit concurrently?
- How do we ensure that the destination recovers a correct and complete version of the source's data?

The rest of this chapter presents our solutions to these problems in the context of the MIXIT architecture, which we describe next.

5.2 MIXIT Architecture

MIXIT is a layered architecture for bulk transfer over static mesh networks. The layers are similar to the traditional PHY, link and network layers, but the interfaces between them, as well as the functions carried out by the network layer, are quite different. The physical and link layers deliver all received data to the network layer, whether or not bits are corrupted. Each packet has a MIXIT header that must be received correctly because it contains information about the destination and other meta-data; MIXIT protects the header with a separate forward error correction (FEC) code that has negligible overhead.

Rather than describe each layer separately, we describe the functions carried out at the source, the forwarders, and the destination for any stream of packets.

5.2.1 The Source

The transport layer streams data to the network layer, which pre-processes it using an error-correcting code as described in Section 5.7. The network layer then divides the resulting stream into batches of K packets and sends these batches to the destination sequentially. Whenever the MAC permits, the network layer creates a different random linear combination of the K packets in the current batch and broadcasts it.

MIXIT's network code operates at the granularity of symbols, which we define as a group of consecutive bits of a packet. The group could be the same collection of bits which are transmitted as a single physical layer symbol (*PHY symbol*) by the modulation scheme (e.g., groups of 4 bits in a 16-QAM scheme), or it could be larger in extent, covering a small number of distinct PHY symbols. The j^{th} symbol in a coded packet, s'_j, is a linear combination of the j^{th} symbols in the K packets, i.e., $s'_j = \sum_i v_i s_{ji}$, where s_{ji} is the j^{th} symbol in the i^{th} packet in the batch and v_i is a per-packet random multiplier. We call $\vec{v} = (v_1, \ldots, v_K)$ the *code vector* of the coded packet. Note that every symbol in the packet created by the source has the same code vector.

The source adds a MIXIT header to the coded packet and broadcasts it. The header describes which symbols were coded together. This description is easy to specify at the source because all symbols in a coded packet are generated using the packet's code vector, $\vec{v}$. The header also contains an ordered list of forwarding nodes picked from its neighbors, each of which is closer to the destination according to the metric described in Section 5.5.

5.2.2 The Forwarders

Each node listens continuously whenever it is not transmitting, attempting to decode whatever it hears. When the PHY detects a packet, it passes the subsequent decoded bits along with SoftPHY hints that reflect its confidence in the decoded bits. The network layer gets this information and uses it to classify symbols into *clean* and *dirty* ones. A clean symbol is one that is likely to be correct, unlike a dirty one. Section 5.3 describes how the MIXIT network layer classifies symbols.

When a node gets a packet without header errors, it checks whether it is mentioned in the list of forwarders contained in the header. If so, the node checks whether the packet contains new information, i.e., is "innovative" [77]. A packet is considered innovative if its code vector $\vec{v}$ is linearly independent of the vectors of the packets the node has previously received from this batch. Checking for independence is straightforward using Gaussian elimination over these short vectors [77]. The node ignores non-innovative packets, and stores the innovative packets it receives from the current batch, preserving the "clean" and "dirty" annotations.

When forwarding data, the node creates random linear combinations of the clean symbols in the packets it has heard from the same batch, as explained in Section 5.4, and broadcasts the resulting coded packet. It also decides how much each neighbor should forward to balance load and maximize throughput, as described in Section 5.5.

Any MAC protocol may be used in MIXIT, but the scheme described in Section 5.6.1 achieves higher concurrency than standard CSMA because it takes advantage of MIXIT's ability to cope with much higher error rates than previous routing protocols.

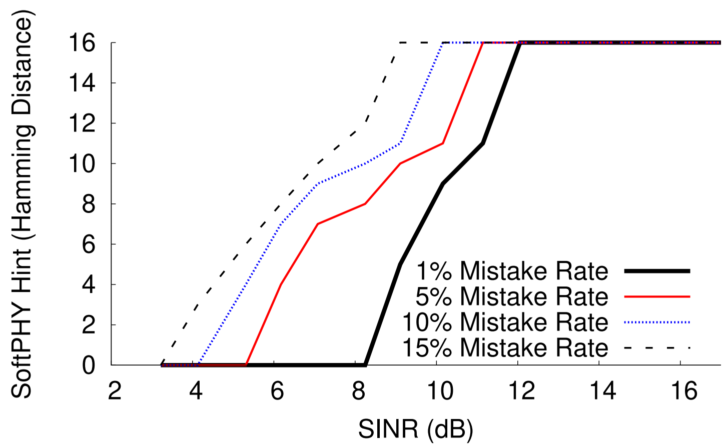


Figure 5-3: Decision boundary for different mistake rates as a function of SINR. At high SINR (> 12dB), all PHY symbols with Hamming distance less than 16 (the maximum possible in the Zigbee physical layer), will satisfy the mistake rate threshold. But at intermediate SINRs (5-12 dB), the PHY symbols have to be picked carefully depending on the mistake rate threshold.

■ 5.2.3 The Destination

MIXIT provides a rateless network code. Hence, the destination simply collects all the packets it can hear until it has enough information to decode the original data as described in Section 5.7. Furthermore, MIXIT provides flexible reliability semantics. Depending on application requirements, the destination can decide how much information is enough. For example, if the application requires full reliability, the destination waits until it can decode 100% of the original symbols, whereas if the application requires 90% reliability, the destination can be done once it decodes 90% of the original symbols. Once the destination decodes the required original symbols, it sends a batch-Ack to the source. The Ack is sent using reliable single path routing, and causes the source to move to the next batch. For the rest of the chapter, we will assume that the destination wants 100% reliability.

■ 5.3 Classifying Received Symbols

MIXIT operates over symbols, which are groups of *PHY symbols*. A symbol is classified as *clean* if none of the constituent PHY symbols are erroneous with a probability higher than γ. It is classified *dirty* otherwise. We call the threshold γ, the *mistake rate*, and it is a configurable parameter of the system. To satisfy the mistake rate threshold, MIXIT's network layer picks a decision boundary on the soft values [65, 132] of the PHY symbols. If all constituent PHY symbols in our symbol have soft values below this decision boundary, then the symbol is classified as clean, else it is dirty. The decision boundary depends on the mistake rate as well as the channel SINR [132, 124].

Fig. 5-3 supports this argument. The figure is generated using a GNU software radio implementation of the Zigbee protocol (see Section 5.8). The figure plots the decision boundary on soft values of PHY symbols for varying SINR at different mistake rates of 1%, 5%, 10% and 15%. Clearly, the boundary depends both on the mistake rate as well as the SINR. The SINR measures the channel noise and interference, and hence reflects how much we should trust the channel to preserve the correlation between transmitted and received signals [132]. Factoring in the specified mistake rate, we can use the above map to pick the right decision boundary to classify symbols.

MIXIT uses the SoftPHY interface proposed in [65, 132], which annotates the decoded PHY symbols with confidence values and sends them to higher layers. We also augment the interface to expose the SINR. The SINR can be estimated using standard methods like that in [69]. The map in Fig. 5-3 can be computed offline, since the relationship between SINR, the confidence estimate, and the decision boundary is usually static [93]. The MIXIT network layer uses the PHY information to classify symbols as clean and dirty, and then performs symbol-level network coding over the clean symbols as described in the next section.

■ 5.4 The MIXIT Network Code

When the MAC permits, the node may forward a coded packet. The symbols in a coded packet are linear combinations of the clean symbols received in packets from the same batch. To see how the coding works let us look at an example.

■ 5.4.1 MIXIT in Action

Consider the scenario in Fig. 5-4, where the source S wants to deliver two packets, P_a and P_b, to the destination. Let the bit error rate (BER) be relatively high such that when the source S broadcasts P_a and P_b, the nodes in the network receive some symbols in errors. The network layer at each node classifies the symbols as either clean or dirty using the SoftPHY hints as described in Section 5.3. Fig. 5-4 illustrates the dirty symbols using shaded cells.

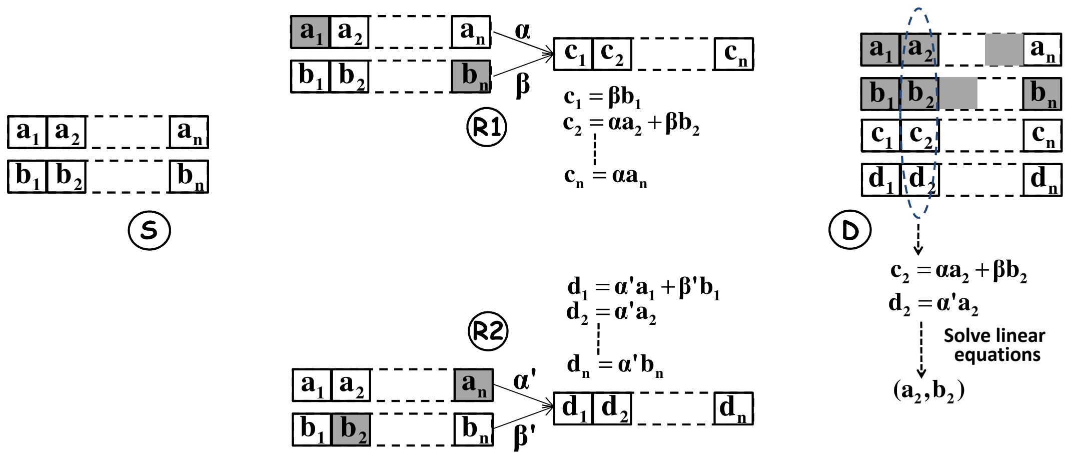


Figure 5-4: Example showing how MIXIT works: The source broadcasts P_a and P_b. The destination and the routers, $R1$ and $R2$, receive corrupted versions of the packets. A shaded cell represents a dirty symbol. If $R1$ and $R2$ forward the clean symbols without coding, they generate redundant data and waste the capacity. With symbol-level network coding, the routers transmit linear combinations of the clean symbols, ensuring that they forward useful information to the destination.

The objective of our symbol-level codes is to minimize the overhead required to funnel the clean symbols to their destination. Specifically, most symbols are received correctly by both $R1$ and $R2$. Hence, without additional measures, the routers will transmit the same symbols to the destination, wasting wireless capacity. To avoid such waste, MIXIT makes the routers forward random linear combinations of the clean symbols they received. Assuming a_i and b_i are the ith symbols in P_a and P_b respectively, router $R1$ picks two random numbers α and β, and creates a coded packet P_c, where the ith symbol, c_i is computed as follows:

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

If both a_i and b_i are dirty, no symbol is sent. Similarly, $R2$ generates a coded packet P_d by picking two random values α' and β' and applying the same logic in the above equation. Since $R1$ and $R2$ use random coefficients to produce the coded symbols, it is unlikely that they generate duplicate symbols [58].

When $R1$ and $R2$ broadcast their respective packets, P_c and P_d, the destination receives corrupted versions where some symbols are incorrect, as shown in Fig. 5-4. Thus the destination has four partially corrupted receptions: P_a and P_b, directly overheard from the source, contain many erroneous symbols; and P_c and P_d, which contain a few erroneous symbols. For each symbol position i, the destination needs to decode two original symbols a_i and b_i. As long as the destination receives two uncorrupted independent symbols in location i, it will be able to properly decode [58]. For example, consider the symbol position $i = 2$, the destination has received:

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

Given that the header of a coded packet contains the multipliers (e.g., α and β), the destination has two linear equations with two unknowns, a_2 and b_2, which are easily solvable (the details of the decoder are explained in Section 5.7). Once the destination has decoded all symbols correctly, it broadcasts an ACK, causing the routers to stop forwarding packets.

■ 5.4.2 Efficient Symbol-Level Codes

The difficulty in creating a network code over symbols is not the coding operation, but in how we express the code efficiently. The length of a symbol is small, one or a few bytes. The MIXIT header in the forwarded packet has to specify how each symbol is derived from the native symbols so that the destination can decode. If all symbols in a packet are multiplied by the same number, then effectively we have a packet-level code, which can be easily expressed by putting the multiplier in the header. However, in MIXIT we want to code clean symbols and ignore dirty ones; i.e., only clean symbols are multiplied by a non-zero number.

Consider a simple example where the batch size is $K = 2$ with the two packets; P_a and P_b. Say that our forwarder has received two coded packets $P_c = \alpha P_a + \beta P_b$ and $P_d = \alpha' P_a + \beta' P_b$. Now our forwarder picks two random numbers v_1 and v_2 and creates a linear combination of the two packets it received.

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

Thus, the newly generated packet has a code vector $\vec{v} = (v_1\alpha + v_2\alpha', v_1\beta + v_2\beta')$. This vector would be sufficient to describe the whole packet if the forwarder received only clean symbols. Specifically, the clean symbol in the jth position in packet P, called s_j, is coded as follows:

$$\begin{aligned} s_j &= v_1c_j + v_2d_j, \qquad \text{where } c_i \text{ and } d_j \text{ are clean} \\ &= (v_1\alpha + v_2\alpha')a_j + (v_1\beta + v_2\beta')b_j \end{aligned}$$

But because some received symbols are dirty, we need a more detailed description of how individual symbols in the packet P are derived from the native symbols. Depending on whether the forwarder has cleanly received the jth symbols in P_c and P_d, called c_j and d_j respectively, the generated symbol s_j might take one of four possible values, with respect to the native symbols.

$$s_j = \begin{cases} (v_1\alpha + v_2\alpha')a_j + (v_1\beta + v_2\beta')b_j & c_j \text{ and } d_j \text{ are clean} \\ v_1\alpha a_j + v_1\beta b_j & \text{only } c_j \text{ is clean} \\ v_2\alpha' a_j + v_2\beta' b_j & \text{only } d_j \text{ is clean} \\ 0 \times a_j + 0 \times b_j & c_j \text{ and } d_j \text{ are dirty} \end{cases} \tag{5.1}$$

Each different value of the symbol is associated with a different code vector, the header has to specify for each symbol in a transmitted packet what is the symbol's code vector.

We address this issue using the following two mechanisms.

(1) Run-length encoding: Because wireless errors are bursty [96, 131], a sequence of consecutive symbols will have the same code vector. We can therefore use run-length encoding to describe the encoding of the transmitted symbols in an efficient manner. The header specifies a sequence of runs, each of which is described as $[(\texttt{Code Vector of run}), (\texttt{Runstart} : \texttt{Runend})]$. For example, in Fig. 5-5, the header of the first outgoing coded packet will specify two runs, $[(\gamma\vec{v}_3), (1, 1000)]$ and $[(\alpha\vec{v}_1 + \beta\vec{v}_2), (1001, 1500)]$.

(2) Pick codes that give longer runs: We force the overhead to stay small by intentionally discarding clean symbols that fragment our run-length-encoding. Said differently, a for-

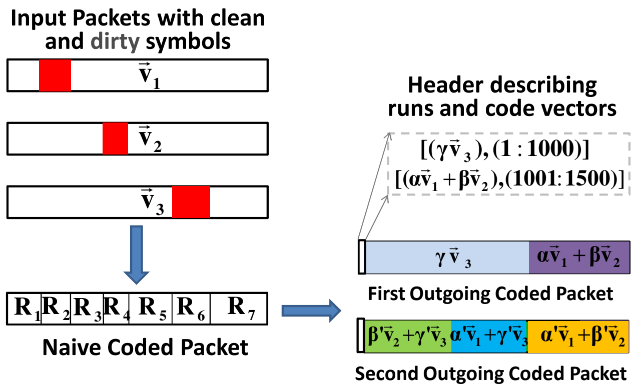

Input Packets with clean and dirty symbols
$\vec{v}_1$
$\vec{v}_2$
$\vec{v}_3$
R_1 R_2 R_3 R_4 R_5 R_6 R_7
Naive Coded Packet
Header describing runs and code vectors
$[(\gamma\vec{v}_3),(1:1000)]$
$[(\alpha\vec{v}_1+\beta\vec{v}_2),(1001:1500)]$
$\gamma\vec{v}_3$
$\alpha\vec{v}_1+\beta\vec{v}_2$
First Outgoing Coded Packet
$\beta'\vec{v}_2+\gamma'\vec{v}_3$
$\alpha'\vec{v}_1+\gamma'\vec{v}_3$
$\alpha'\vec{v}_1+\beta'\vec{v}_2$
Second Outgoing Coded Packet

Figure 5-5: Creating coded packets with longer runs. The forwarder received 3 packets with code vectors v_1, v_2 and v_3. All packets contain dirty symbols represented as shaded areas. Naively coding over all clean received symbols results in a coded packet with 7 different runs. However, by ignoring some of the clean symbols, the node can generate coded packets with much fewer runs.

warder can decide to ignore some clean symbols to ensure the header has longer runs of symbols with the same code vector, and thus can be encoded efficiently.

Consider the example in Fig. 5-5, where the forwarder has received 3 packets, each with some dirty symbols. Naively, applying the symbol-level network code along with the run-length encoding described above, we get a coded packet that has seven different runs. But, we can create fewer runs of longer lengths by ignoring some clean symbols in the coding procedure. For example, in the first five runs in the naive coded packet, we can ignore clean symbols from the first and second received packets. As a result, the five runs would coalesce to a single longer run with the code vector $\gamma\vec{v}_3$, where γ is a random multiplier and $\vec{v}_3$ is the code vector of the third received packet. Similarly for the last two runs, if we ignore clean symbols from the third received packet, we are left with a single longer run with the code vector $\alpha\vec{v}_1 + \beta\vec{v}_2$, where α and β are random multipliers and $\vec{v}_1$ and $\vec{v}_2$ are the code vectors of the first and second received packets. The resulting coded packet shown in Fig. 5-5 has only two runs with two code vectors, and requires less overhead to express.

But, what if the forwarder has to transmit a second coded packet? One option is to ignore the same set of clean symbols as above, but use different random multipliers, α', β', γ'. We would get a coded packet with two runs and their code vectors being $\gamma'\vec{v}_3$ and $\alpha'\vec{v}_1 + \beta'\vec{v}_2$. But this transmission will be wasteful, since the symbols in the first run are *not innovative* w.r.t the first coded packet the node already transmitted ($\gamma'\vec{v}_3$ is not linearly independent of $\gamma'\vec{v}_3$). The solution is to split the first long run into two smaller runs by including clean symbols from the first and second packets, which we had previously ignored. The second coded packet, shown in Fig. 5-5 has 3 runs with 3 different code vectors $\beta'\vec{v}_2 + \gamma'\vec{v}_3$, $\alpha'\vec{v}_1 + \gamma'\vec{v}_3$ and $\alpha'\vec{v}_1 + \beta'\vec{v}_2$. The new packet is innovative w.r.t the previously transmitted coded packet, and uses lower overhead in describing the codes.

5.4.3 Dynamic Programming to Minimize Overhead

We now present a systematic way to minimize the number of runs in a coded packet, while ensuring that each packet is innovative with respect to previously transmitted coded packets. We formalize the problem using dynamic programming. Let there be n input packets, from which we create the *naive coded packet*, as shown in the previous example. Say the naive packet contains the runs $R_1R_2 \ldots R_L$. The optimization attempts to combine consecutive runs from the naive coded packet into a single run, whose symbols all have the same code vector by ignoring some of the input clean symbols. Let C_{ij} be the combined run that includes the runs $R_i \ldots R_j$ from the naive coded packet. Note that the combined run C_{ii} is the same as run R_i.

Next, we show that each combined run can be assigned a cost, and that the optimization problem that minimizes the number of innovative combined runs exhibits the "optimal substructure" property, i.e., the cost of a combined run can be derived from the cost of two sub-runs.

The goal is to create an outgoing coded packet out of the smallest number of combined runs, while ensuring that the information we send out is innovative. Thus, we can formulate the cost of a combined run as follows:

$$Cost(C_{ij}) = \min\left\{\mathbf{f}(C_{ij}), \min_{i<k<j}\{Cost(C_{ik}) + Cost(C_{kj})\}\right\} \tag{5.2}$$

where $\mathbf{f}(C_{ij})$ is given by:

$$\mathbf{f}(C_{ij}) = \begin{cases} \sum_i^j |R_i| & \text{if } C_{ij} \text{ is not innovative} \\ (2\log S)/8 + K & \text{otherwise} \end{cases} \tag{5.3}$$

Intuitively, the function $\mathbf{f}(C_{ij})$ says that if the combined run C_{ij} is not innovative with respect to previous transmissions, the cost is the number of symbols in that combined run. But if the combined run is innovative with respect to previous transmissions, its cost is just the number of bytes required to describe it. This requires describing the start and end of the combined run, which can be done using $(2 \log S)/8$ bytes, where S is the packet size, and describing the combined run's code vector, which can be done using K bytes, where K is the batch size. The second component in Eq. 5.2 checks if splitting the combined run C_{ij} into two smaller runs incurs a smaller cost, and if it does, it finds the best way to split it.

The forwarder computes the dynamic program top-down using a table to memoize the costs. Because the algorithm coalesces runs in the naively coded packet, the table has at most as many entries as there are combined runs. The worst case complexity of the algorithm is $O(L^3)$, but in practice it runs faster due to the following heuristic. In Eq. 5.2, if C_{ij} is innovative, we do not need to check whether splitting it reduces the cost because $\mathbf{f}(C_{ij})$ will always be lower than the cost of the two sub runs, whose cost will at least be $2\mathbf{f}(C_{ij})$. Typically, the DP takes under a millisecond to run for a packet with $L \approx 15 - 20$.

5.5 Congestion-Aware Forwarding

In general, because wireless is a broadcast medium several downstream routers will hear any given symbol without error. For each symbol, the ideal situation is for the downstream forwarder with the best path quality to the destination to forward the symbol (after coding it). For example, in Fig. 5-6(a), routers $R1$ and $R2$ hear all the symbols from $S1$. However, $R2$ should be the one to forward the symbols because it can deliver them to the destination in fewer transmissions.

Path quality is not the only consideration in making this decision because one ultimately cares about the *time* it takes for a symbol to reach the destination. If a path has high quality (as measured by a low error rate), but also has long queues at its forwarders, it would not be advisable to use it. Fig. 5-6(b) shows an example where a second flow being forwarded through $R2$ causes $R2$'s queues to grow, so having $R1$ forward some of its traffic would improve performance and avoid creating a bottleneck at $R2$.

These requirements suggest the following approach for a node to decide how its downstream forwarders should forward on its behalf. First, for each path (via a downstream node), determine the expected time it would take to successfully send a symbol along that path. This time, which we call C-ETS (for "congestion-aware ETS"), incorporates both path

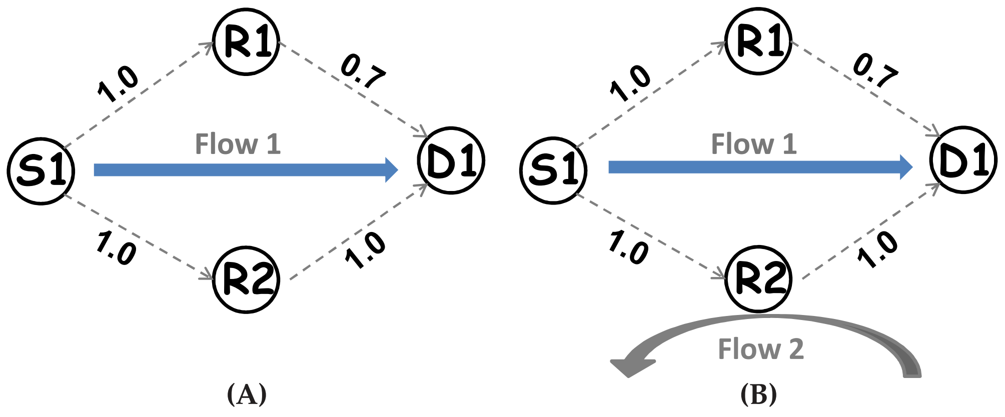


Figure 5-6: Example of congestion-aware forwarding. If there is a single flow in the network, $S1$ should always send all it's traffic through $R2$ since he has the better link to the destination. But if $R2$ is involved in a second flow, then $S1$ should also send some of his traffic through $R1$ to avoid creating a bottleneck at $R2$.

quality and node queue lengths (backlog). C-ETS via a downstream node i to a destination d is computed as C-ETS$(i, d) = \text{PQ}(i, d) + kQ(i)$. Here, PQ is the path quality from i to d, which depends on the symbol delivery probabilities and captures the time taken to deliver a symbol to the destination in the absence of any queueing. In our implementation, we approximate this term using the ETS metric (defined as the expected number of transmissions required to deliver a symbol), instead of using a more exact formula for the expected number of transmissions using opportunistic routing [18].[2] $Q(i)$ is the total number of symbols (backlog) across all flows queued up at node i yet to be transmitted (k is a constant dimensional scaling factor that depends on the time it takes to transmit one symbol).

We now discuss how a node decides how many of its queued up symbols, $Q(i, f)$ for flow f, each downstream node should forward. (Because of the random network code, we don't have to worry about downstream nodes sending the exact same information.) The high-level idea is to favor nodes with smaller C-ETS values, but at the same time apportioning enough responsibility to every downstream node because no link or path is loss-free in general. Each node assigns responsibility to its downstream nodes by assigning credits. Credit determines the probability with which the downstream node should forward symbols belonging to the flow when they receive transmissions from the node. The downstream node with best the C-ETS has credit 1; the next-best node has credit $(1 - p_1)$,

[2]Computing the ETS metric is simpler and does not change the paths used by much.

where p_1 is the symbol delivery probability to the best downstream node; the best one after that has credit $(1-p_1)(1-p_2)$, and so on. What we have done here is to emulate, in expectation, the best downstream node sending all the symbols it hears, the next-best one only forwarding a fraction that the best one may not have heard, and so on, until all the nodes down to the worst neighbor have some small responsibility.

How many transmissions should the node make? The node should make enough transmissions to make sure that every queued up symbol reaches some node with a lower C-ETS metric. If the symbol delivery probability to downstream neighbor j is p_j, then the probability that *some* downstream neighbor gets any given symbol is $P = 1 - \prod_j (1-p_j)$. Hence in expectation, the number of coded symbols from a batch that a node would have to send per queued up symbol is equal to $1/(1-P)$ before some downstream node gets it. Each node achieves this task by maintaining a decrementing per-flow *Transmit_Counter*, throttling transmission of the batch when its value reaches 0.

The above intuitions are formalized in Alg. 3.

3 Computing credit assignment at node i

```
while $Q(i,f) > 0$ do
    Update C-ETS of downstream nodes from overheard packets
    Sort downstream nodes according to their C-ETS
    $P_{left} = 1$
    for node $j$ in set of downstream nodes sorted according to C-ETS do
        $credit\_assgn(j) = P_{left}$
        $P_{left} = P_{left} * (1 - p(i,j))$
    end for
    Increment Transmit_Counter of flow $f$ by $1/(1 - P_{left})$
    Decrement $Q(i,f)$ by 1
end while
```

Distributed protocol: Each node, i, periodically measures the symbol delivery probabilities $p(i,j)$ for each of its neighbors via probes. These probabilities are distributed to its neighbors using a link state protocol. Node i includes the computed *credit_assgn* for each of its downstream nodes in the header of every packet it transmits. When downstream nodes receive a packet, they update their $Q(i,f)$ for that flow by the amount specified in the header. Further, whenever node i transmits a packet, it includes its C-ETS to the corresponding destination in the header. Upstream nodes which overhear this packet, use the C-ETS value in their credit assignment procedure.

The algorithm above improves on the routing algorithms used in prior packet based opportunistic routing protocols like MORE [18] in two ways. First, we use queue backlog information explicitly to avoid congested spots and balance network-wide load; prior work ignores congestion. Second, the algorithm works at the symbol-level, which is the right granularity for performing opportunistic routing on symbols. The algorithm is similar in spirit to theoretical *back-pressure* [106, 28] ideas, but the exact technique is different and simpler. We also present an actual implementation and evaluation of this algorithm in Section 5.9.

■ 5.6 Increasing Concurrency

Current wireless mesh networks allow a node to transmit only when they are sure that they can deliver the *packet* to the intended next hop with high probability. MIXIT however, has looser constraints:

1. It does not require the delivery of correct packets; it can work with partially correct packets.
2. Because of its opportunistic nature, MIXIT only needs to ensure that every symbol reaches *some* node closer to the destination than the transmitter; it does not need to ensure that a specific node gets the correct symbols.

MIXIT exploits the above flexibility to increase concurrency without affecting end-to-end reliability, improving throughput by enabling a more pipelined transmission pattern. MIXIT's concurrency design has two components: determining when concurrent transmissions are beneficial and building a distributed protocol to take advantage of concurrency opportunities. We describe both components below.

■ 5.6.1 When Should Two Nodes Transmit Concurrently?

MIXIT, similar to conflict maps [127], determines if two nodes should transmit concurrently by predicting the throughput under concurrent transmissions and comparing it to the throughput when the nodes transmit separately. The nodes independently pick the strategy with the higher expected throughput. Specifically, let $n1$ and $n2$ be two nodes transmitting packets of two flows l and k. $Ne(n1, l)$ and $Ne(n2, k)$ are the corresponding sets of downstream nodes for $n1$ and $n2$ for the respective flows. Symbol delivery probabilities on any link will depend on whether these nodes transmit concurrently or not. Let

$p^c(i,j)$ be the symbol delivery probability on link (i,j) when the two nodes transmit concurrently and $p(i,j)$ when they don't. The symbol delivery likelihoods achieved by node $n1$ for flow l with and without concurrent transmissions are given by

$$\begin{aligned} D^c(n1,l) &= 1 - (\textstyle\prod_{j \in Ne(n1,l)} (1 - p^c(n1,j))) \\ D(n1,l) &= 1 - (\textstyle\prod_{j \in Ne(n1,l)} (1 - p(n1,j))) \end{aligned} \tag{5.4}$$

The symbol delivery likelihood is the probability that at least one node in $Ne(n1,l)$ receives the symbol correctly when node $n1$ transmits. The symbol delivery likelihood depends on other concurrent traffic, and could differ if $n2$'s transmission interferes with $n1$'s. Similarly, $n2$ can compute its symbol delivery likelihood under both conditions.

Each node then computes the following *concurrency condition*:

$$D^c(n1,l) + D^c(n2,k) > (D(n1,l) + D(n2,k))/2 \tag{5.5}$$

The above equation compares overall delivery likelihood under the two scheduling strategies. If the above condition is true, it implies that more information gets delivered per time slot when nodes transmit concurrently than when they transmit separately. Each node independently evaluates the above condition and decides its strategy.[3]

■ 5.6.2 Estimating Symbol Delivery Probabilities

The concurrency condition above depends on the symbol delivery probabilities. Empirically measuring these probabilities for all pairs of concurrent transmissions has $O(N^2)$ cost, where N is the number of nodes. Instead, MIXIT uses $O(N)$ empirical signal-to-noise ratio (SNR) measurements to predict these probabilities for any set of concurrent transmissions. The approach works as follows.

1. The SNR profile of the network is measured when there is little traffic. Each of the N nodes broadcasts probe packets in turn, while the rest of the nodes measure the received SNR and the fraction of correctly received symbols. The measurements are of the form $SNR(i,j)$ and $p(x)$, where $SNR(i,j)$ is the received SNR at j when i transmits and $p(x)$ is the fraction of correct symbols received when the SNR is x.

[3]The above conditions assumes a single radio transmission bit-rate; it can be adapted easily to handle variable bit-rates.

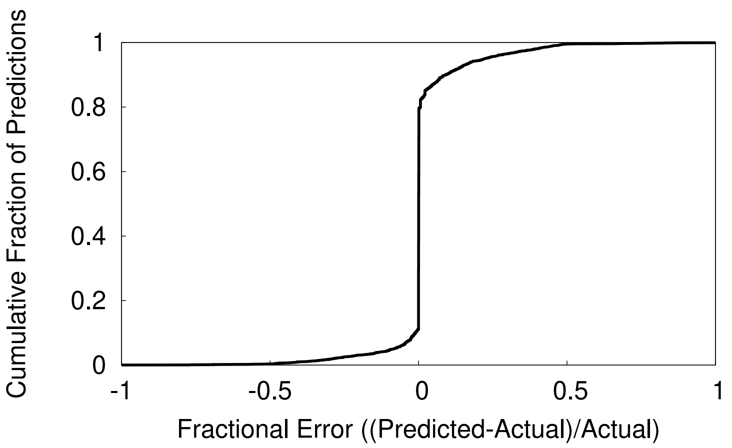


Figure 5-7: Prediction error CDF. The SNR based prediction model accurately predicts the symbol delivery probabilities under concurrent transmissions for 72% of the cases. The inaccuracies are primarily in cases where $(SNR(n1,m) - SNR(n2,m)) < 4dB$, i.e., when concurrent transmissions will result in a signal being received with low SINR at the receivers.

2. Nodes use the SNR profile to predict the signal-to-interference+noise ratio (SINR) at any node under concurrent transmissions. Specifically, if nodes $n1$ and $n2$ transmit concurrently, the SINR at node m is computed as $SINR(n1,n2,m) = SNR(n1,m) - SNR(n2,m)$ assuming $SNR(n1,m) > SNR(n2,m) \geq c$, where c is a threshold SNR below which no symbol can be decoded. The symbol delivery probability is then predicted to be $p(SINR(n1,n2,m))$, i.e., it is the same as if the signal was received at m with SNR of $SINR(n1,n2,m)$.

Fig. 5-7 plots the CDF of prediction errors using the above model. The results are from a 25-node testbed of GNURadio software nodes with USRP frontends, with two concurrent senders transmitting 802.15.4 packets. The figure demonstrates that the prediction model is quite accurate, with the inaccurate predictions occurring at low SINR (< 4 dB). But because the symbol delivery probability at low SINR is negligible, inaccuracies in the low SINR region do not affect performance. Furthermore, unlike prior proposals [120, 111] that try to predict packet delivery rates using a SINR model, MIXIT's model predicts symbol delivery likelihoods. The latter is simpler since packet delivery is a complex function of error rates, nature of interference etc. Finally, the concurrency condition is a binary decision, even if the predicted probabilities are slightly off, it is unlikely to affect the decision.

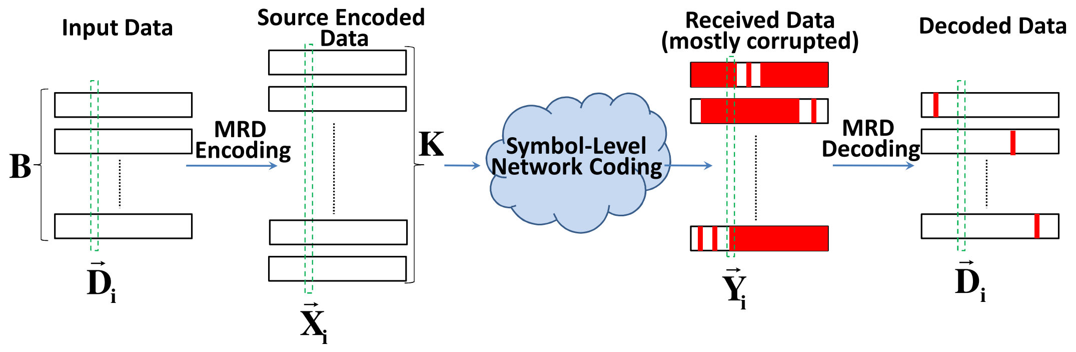


Figure 5-8: MIXIT's error correcting code design. The source preprocesses the original data packets with a MRD code in groups of B symbols and transmits them. The network applies symbol-level network coding on clean symbols. Erroneous clean symbols may end up corrupting all the received symbols at the destination, but the destination can use the MRD code to decode most of the original data symbols.

5.6.3 Distributed Channel Access Protocol

A node uses a two-step procedure when it has packets enqueued for transmission. First, if it has not heard any on-going transmissions, it simply goes ahead and transmits. But if it has heard an on-going transmission, then it uses Eq. 5.5 to determine if it should transmit concurrently or defer until the on-going transmission has finished.

How does a node know which other nodes are transmitting at that time instant? Similar to prior work [65, 127], MIXIT encapsulates every packet with a header and trailer. The header includes the identity of the transmitting node, and the flow to which the packet belongs. Other nodes overhearing a packet use the header to identify the beginning of an active transmission and the trailer to signify the end.

5.7 Error Correction

Up to now we have ignored the difference between clean and correct symbols and focused on delivering clean symbols to the destination. But clean symbols can be incorrect. Further, an erroneous symbol which was incorrectly classified clean may end up corrupting other correct clean symbols due to network coding. Thus the destination potentially, could get all symbols corrupted due to a single clean but erroneous symbol. Fortunately, MIXIT comes with in-built error correction capability which allows the destination to recover the original correct symbols. The error-correcting code is not affected even if all received symbols are corrupted, the only thing that matters is how many erroneous symbols were incorrectly classified clean. The code guarantees that if m erroneous symbols were incorrectly classified clean, then the destination needs only $B + 2m$ symbols to recover the original B symbols. This guarantee is theoretically optimal [136, 15]. The code is simple, rateless and end-to-end. Routers inside the network are oblivious to the existence of the error-correcting code and are only required to perform simple network coding operations.

MIXIT's error-correcting code is built on the observation that random network coding is *vector space preserving* [76]. Specifically, if we model the original data injected by the source as a basis for a vector space **V**, then the random network code acts only as a linear transformation **T** on the vector space. But vector spaces are preserved under linear transformations if no errors occur, and if errors do occur, the received vector space **U** is very close to the transmitted vector space **V** under an appropriately defined distance metric on vector spaces.

Recent work [76, 118, 62] has studied the problem of making network coding resilient to byzantine adversaries injecting corrupted packets. It has observed that low complexity Maximum Rank Distance (MRD) codes [43], with a small modification can be applied to exploit the vector space observation and correct adversarial errors. The network coding in MIXIT is different, but the basic algorithm in MRD can be adapted to work with MIXIT's symbol-level network code. Fig. 5-8 shows the high level architecture of how MRD codes are integrated within MIXIT.

5.7.1 Fundamentals of Rank Codes

In this section we describe some fundamentals of rank-codes that were introduced by Gabidulin in 1985 [43]. Most of the material in this section can be found in [118, 43, 44, 112], we present it here for completeness.

Let **x** be a codeword of length n with elements from $GF(q^N)$, where q is a power of a prime. Let us consider a bijective mapping

$$A : GF(q^N)^n \to \mathbf{A}_N^n \tag{5.6}$$

Definition 5.1 ***(Rank Metric over** $GF(q)$**)*** *The rank of* $\mathbf{x}$ *over* q *is defined as* $r(\mathbf{x}|q) = r(\mathbf{A}|q)$. *The rank function* $r(\mathbf{A}|q)$ *is equal to the maximum number of linearly independent rows or columns of* $\mathbf{A}$ *over* $GF(q)$.

It is well known that the rank function defines a norm. Indeed, $r(\mathbf{x}|q) \geq 0, r(\mathbf{x}|q) =$

$0 \iff \mathbf{x} = 0$. In addition, $r(\mathbf{x}+\mathbf{y}|q) \leq r(\mathbf{x}|q) + r(\mathbf{y}|q)$. Furthermore, $r(a\mathbf{x}|q) = |a|r(\mathbf{x}|q)$ is also fulfilled, if we set $|a| = 0$ for $a = 0$ and $|a| = 1$ for $a \neq 0$.

Definition 5.2 ***(Rank Distance)*** *Let* $\mathbf{x}$ *and* $\mathbf{y}$ *be two codewords of length n with elements from $GF(q^N)$. The rank distance is defined as $dist_r(\mathbf{x}, \mathbf{y}) = r(\mathbf{x} - \mathbf{y}|q)$.*

Similar to the minimum Hamming distance, we can determine the maximum rank distance of a code $\mathscr{C}$.

Definition 5.3 ***(Minimum Rank Distance)*** *For a code $\mathscr{C}$ the minimum rank distance is given by:*

$$d_r := \min\{\text{dist}_r(\mathbf{x}, \mathbf{y}) | \mathbf{x} \in \mathscr{C}, \mathbf{y} \in \mathscr{C}, \mathbf{x} \neq \mathbf{y}\}, \tag{5.7}$$

or when the code is linear:

$$d_r := \min\{r(\mathbf{x}|q) | \mathbf{x} \in \mathscr{C}, \mathbf{x} \neq 0\}. \tag{5.8}$$

Let $\mathscr{C}(n, k, d_r)$ be a code of dimension k, length n, and minimum rank distance d_r.

It was shown in [43] that there also exists a Singleton-style bound for the rank distance. Theorem 5.1 shows, how the minimum rank distance d_r is bounded by the minimum Hamming distance d_h and by the Singleton bound.

Theorem 5.1 ***(Singleton-style Bound)*** *For every linear code $\mathscr{C}(n, k, d_r) \in GF(q^N)^n$, d_r is upper bounded by:*

$$d_r \leq d_h \leq n - k + 1 \tag{5.9}$$

Definition 5.4 ***(MRD Code)*** *A linear (n, k, d_r) code $\mathscr{C}$ is called Maximum Rank Distance (MRD) code, if the Singleton-style bound is fulfilled with equality.*

In [43], a constructive method for the parity check matrix and the generator matrix of an MRD code is given as follows:

Theorem 5.2 ***(Construction of MRD Codes)*** *A parity check matrix* H *which defines an MRD* *code is given by:*

$$\mathrm{H} = \begin{bmatrix} h_0 & h_1 & \dots & h_{n-1} \\ h_0^q & h_1^q & \dots & h_{n-1}^q \\ h_0^{q^2} & h_1^{q^2} & \dots & h_{n-1}^{q^2} \\ \vdots & \vdots & \ddots & \vdots \\ h_0^{q^{d-2}} & h_1^{q^{d-2}} & \dots & h_{n-1}^{q^{d-2}} \end{bmatrix} \tag{5.10}$$

and the corresponding generator matrix can be written as:

$$\mathrm{G} = \begin{bmatrix} g_0 & g_1 & \dots & g_{n-1} \\ g_0^q & g_1^q & \dots & g_{n-1}^q \\ g_0^{q^2} & g_1^{q^2} & \dots & g_{n-1}^{q^2} \\ \vdots & \vdots & \ddots & \vdots \\ g_0^{q^{k-1}} & g_1^{q^{k-1}} & \dots & g_{n-1}^{q^{k-1}} \end{bmatrix} \tag{5.11}$$

where the elements $h_1, h_1, \dots, h_{n-1} \in GF(q^N)$ and $g_0, g_1, \dots, g_{n-1} \in GF(q^N)$ are linearly independent over $GF(q)$.

In the following we define $\mathscr{C}_{MRD}(n, k, d_r)$ as an MRD array code of length n, dimension k and minimum rank distance $d_r = n - k + 1$.

The decoding of Rank-Codes with the modified Berlekamp-Massey algorithm can be done based on linearized polynomials.

Definition 5.5 ***(Linearized Polynomials)*** *A linearized polynomial over $GF(q^N)$ is a polynomial of the form*

$$L(x) = \sum_{p=0}^{N(L)} L_p x^{q^p}, \tag{5.12}$$

where $L_p \in GF(q^N)$ and $N(L)$ is the norm of the linearized polynomial. The norm $N(L)$ characterizes the largest p where $L_p \neq 0$. Let $\otimes$ be the symbolic product of linearized polynomials defined as:

$$F(x) \otimes G(x) = F(G(x)) = \sum_{p=0}^{j} \sum_{i+l=p} (f_i g_l^p) x^{q^p}, \tag{5.13}$$

where $j = N(F) + N(G)$. It is known that the symbolic product is associative and distributive, but it is non-commutative.

5.7.2 Berlekamp-Massey Algorithm for Rank-Codes

In this section, we describe the decoding algorithm for Rank-Codes with the modified Berlekamp-Massey algorithm. For simplicity we will define the minimum rank distance in the following sections by d. Let c, r and e be the codeword vector, the received vector, and the error vector of length n with elements from $GF(q^N)$, respectively. The received vector is $\mathrm{r} = \mathrm{c} + \mathrm{e}$. Let $v = r(\mathrm{e}|q)$ be the rank of the error vector e. Now, we present a method of finding the correct codeword, if $2v < d_r$.

We can calculate the syndrome s by:

$$\mathrm{s} = \mathrm{r}.\mathrm{H}^T = (\mathrm{c} + \mathrm{e})\mathrm{H}^T. \tag{5.14}$$

Let us define a $(v \times n)$ matrix mY of rank v whose entries are from the base field $GF(q)$. Thus, we can write

$$\mathrm{e} = (E_0, E_1, \ldots, E_{v-1})\mathrm{Y}, \tag{5.15}$$

where $E_0, E_1, \ldots, E_{v-1} \in GF(q^N)$ are linearly independent over $GF(q)$. Now, we define the matrix Z as

$$\mathrm{Z}^T = \mathrm{Y}\mathrm{H}^T = \begin{bmatrix} z_0 & z_0^q & \cdots & z_0^{q^{d-2}} \\ z_1 & z_1^q & \cdots & z_1^{q^{d-2}} \\ \vdots & \vdots & \ddots & \vdots \\ z_{v-1} & z_{v-1}^q & \cdots & z_{v-1}^{q^{d-2}} \end{bmatrix} \tag{5.16}$$

It can be shown that the elements $z_0, z_1, \ldots, z_{v-1} \in GF(q^N)$ are linearly independent over $GF(q)$. Hence, equation 5.14 can be written as:

$$(S_0, S_1, \ldots, S_{d-2}) = (E_0, E_1, \ldots, E_{v-1}).\mathrm{Z}^T \tag{5.17}$$

or

$$S_p = \sum_{j=0}^{v-1} E_j z_j^{q^p} \qquad , p = 0, \ldots, d-2. \tag{5.18}$$

By raising each side of the equations to the power of q^{-p} we get:

$$S_p^{q^{-p}} = \sum_{j=0}^{v-1} E_j^{q^{-p}} z_j \quad , p = 0, \ldots, d-2. \tag{5.19}$$

Hence, we have a system of $d-1$ equations with $2-v$ unknown variables that are linear in $z_0, z_1, \ldots, z_{v-1}$. Note that also the rank v of the error vector is unknown. It is sufficient to find one solution of the system because every solution of $E_0, E_1, \ldots, E_{v-1}$ and $z_0, z_1, \ldots, z_{v-1}$ results in the same error vector e.

Let $\Lambda(x) = \sum_{j=0}^{v} \Lambda_j x^{q^j}$ be a linearized polynomial, which has all linear combinations of $E_0, E_1, \ldots, E_{v-1}$ over $GF(q)$ as its roots and $\Lambda_0 = 1$. We call $\Lambda(x)$ the row error polynomial. Also, let $S(x) = \sum_{j=0}^{d-2} S_j x^{q^j}$ be the linearized syndrome polynomial.

Now, it is possible to define the key equation by the next theorem.

Theorem 5.3 ***(Key Equation)***

$$\Lambda(x) \otimes S(x) = F(x) \pmod{x^{q^{d-1}}} \tag{5.20}$$

where $F(x)$ is an auxiliary linearized polynomial that has norm $N(F) < v$.

Proof. From the definition of linearized polynomials we know that

$$\Lambda(x) \otimes S(x) = \sum_{p=0}^{v+d-2} \left(\sum_{i+l=p} \Lambda_i S_l^{q^i} \right) x^{q^p}. \tag{5.21}$$

Since all coefficients $p \geq d-1$ vanish because of the modulo operation of equation 5.20 and the symbolic product of two linearized polynomials results in another linearized polynomial, we just have to prove that $F_p = 0$ for $v \leq p \leq d-2$.

$$\begin{aligned} \sum_{i+l=p} \Lambda_i S_l^{q^i} &= \sum_{i=0}^{p} \Lambda_i S_{p-i}^{q^i} &&= \sum_{i=0}^{p} \Lambda_i \left(\sum_{s=0}^{v-1} E_s z_s^{q^{p-i}} \right)^{q^i} \\ &= \sum_{s=0}^{v-1} z_s^{q^p} \left(\sum_{i=0}^{p} \Lambda_i E_s^{q^i} \right) &&= \sum_{s=0}^{v-1} z_s^{q^p} \Lambda(E_s) = 0 \end{aligned} \tag{5.22}$$

because p is equal to $v = N(\Lambda)$ or larger and $E_0, E_1, \ldots, E_{v-1}$ are roots of $\Lambda(x)$. ■

Hence, we have to solve the following system of equations to get $\Lambda(x)$, if $2.v < d$.

$$-S_p = \sum_{i=1}^{v} \Lambda_i S_{p-i}^{q^i}, \quad p = v, \ldots, 2v-1 \tag{5.23}$$

This can be written in matrix form as:

$$\mathrm{S} = \begin{bmatrix} \Lambda_v \\ \Lambda_{v-1} \\ \Lambda_{v-2} \\ \vdots \\ \Lambda_1 \end{bmatrix} = \begin{bmatrix} -S_v \\ -S_{v+1} \\ -S_{v+2} \\ \vdots \\ -S_{2v-1} \end{bmatrix} \tag{5.24}$$

with S defined as:

$$\mathrm{S} = \begin{bmatrix} S_0^{q^v} & \ldots & S_{v-1}^{q^1} \\ S_1^{q^v} & \ldots & S_v^{q^1} \\ S_2^{q^v} & \ldots & S_{v+1}^{q^1} \\ \vdots & \ddots & \vdots \\ S_{v-1}^{q^v} & \ldots & S_{2v-2}^{q^1} \end{bmatrix} \tag{5.25}$$

It can be shown that the matrix S is nonsingular. Thus, the system of equations has a unique solution. This solution can be efficiently found with a modified Berlekamp-Massey algorithm. The algorithm can be implemented completely in hardware using shift registers [112]. Here we summarize the different steps of the decoding procedure.

1. Calculate the syndrome with equation 5.14.
2. Solve the key equation 5.24 with the modified Berlekamp-Massey algorithm to obtain $\Lambda(x)$.
3. Calculate the linearly independent roots $E_0, E_1, \ldots, E_{v-1}$ of $\Lambda(x)$. This can be done with the algorithm described in [7].
4. Solve the linear system of equations 5.19 for the unknown variables $z_0, z_1, \ldots, z_{v-1}$.
5. Calculate the matrix Y using equation 5.16.
6. Calculate the error vector e by equation 5.15 and the decoded codeword $\hat{\mathrm{c}} = \mathrm{r} - \mathrm{e}$.

5.7.3 Decoding Rank-Codes with Row Erasures

In MIXIT, the destination might want to decode packets as soon as it has enough information, i.e., use the network code as a rateless code. In this case, if l out of n packets are received, then we can treat the missing $n-l$ packets as row erasures in the matrix. In this case, the decoder has to correct erasures and errors. This involves a slight modification of the previous algorithm, essentially adding a preprocessing step. The number of row erasures is specified as s_r and the rank of the error matrix whose erased rows are filled with zeros is b. We can find the correct codeword, if $s_r + 2b < d$. Let $\psi(x)$ be a linearized polynomial that has all linear combinations of the row erasures as its roots and $\psi_0 = 1$. $\psi(x)$ is called the row erasure polynomial.

Let $\nu = s_r + b$ and let $s_r + 2b < d$. We define $\tilde{\Lambda}(x)$ as the row errata polynomial that has all linear combinations of $E_0, E_1, \ldots, E_{\upsilon-1}$ as roots. Without loss of generality we set $E_0, E_1, \ldots, E_{s_r-1}$ as the row erasures and $E_{s_r}, E_{s_r+1}, \ldots, E_{\upsilon-1}$ as the rank errors. The modified key equation 5.24 for the case with row erasures can be written as:

$$\tilde{\Lambda}(x) \otimes S(x) = F(x) \mod x^{q^d-1} \tag{5.26}$$

where the norm $N(F) < \nu$. The proof is similar to the proof in Section 5.7.2. Again we can represent equation 5.26 in matrix form:

$$\mathrm{S}_r \begin{bmatrix} \tilde{\Lambda}_\nu \\ \tilde{\Lambda}_{\nu-1} \\ \vdots \\ \tilde{\Lambda}_1 \end{bmatrix} = \begin{bmatrix} -S_\nu \\ -S_{\nu+1} \\ \vdots \\ -S_{b+\nu-1} \end{bmatrix} \tag{5.27}$$

with S_r defined as:

$$\mathrm{S}_r = \begin{bmatrix} S_0^{q^\nu} & S_1^{q^{\nu-1}} & \ldots & S_{\nu-1}^{q^1} \\ S_1^{q^\nu} & S_2^{q^{\nu-1}} & \ldots & S_\nu^{q^1} \\ \vdots & \vdots & \ddots & \vdots \\ S_{b-1}^{q^\nu} & S_b^{q^{\nu-1}} & \ldots & S_{b+\nu-2}^{q^1} \end{bmatrix} \tag{5.28}$$

Additionally, we know that the row erasures $E_0, E_1, \ldots, E_{s_r-1}$ are roots of $\tilde{\Lambda}(x)$. Hence, we get the following system of equations:

$$\begin{bmatrix} E_0^{q^\nu} & \dots & E_0^{q^1} \\ E_1^{q^\nu} & \dots & E_1^{q^1} \\ \vdots & \ddots & \vdots \\ E_{s_r-1}^{q^\nu} & \dots & E_{s_r-1}^{q^1} \end{bmatrix} \begin{bmatrix} \tilde{\Lambda}_\nu \\ \tilde{\Lambda}_{\nu-1} \\ \vdots \\ \tilde{\Lambda}_1 \end{bmatrix} = \begin{bmatrix} E_0 \\ E_1 \\ \vdots \\ E_{s_r-1} \end{bmatrix} \tag{5.29}$$

It is possible to show that the $s_r + b$ equations for the unknown variables $\tilde{\Lambda}_1, \tilde{\Lambda}_2, \dots, \tilde{\Lambda}_\nu$ are linearly independent and therefore have a unique solution. Also it is possible to show that the Berlekamp-Massey algorithm solves this system, if r and L_{s_r} are both initialized with s_r and the row errata polynomial $\tilde{\Lambda}^{(s_r)}(x)$ is initialized with $\psi(x)$.

The decoding procedure of Rank-Codes with row erasures can be summarized as follows.

1. Fill the erased rows of the codeword with zeros or any other symbols and calculate the syndrome by equation 5.14.

2. Solve the key equation 5.27 and 5.29 with the Berlekamp-Massey algorithm for row erasures.

Steps 3-7 are equivalent to the decoding algorithm described in Section 5.7.2.

5.7.4 Discussion

Finally, we summarize the main differences between the theoretical literature on decoding MRD codes and MIXIT.

- Symbol-level network coding along with the end-to-end MRD code can function as a rateless error-correcting code. The destination can attempt to decode the original data vector $\vec{D}_i$ as soon as it receives $B < K$ coded symbols for that position. If no erroneous symbols had seeped through, then it will be able to recover the original correct data. If not, it simply waits to receive more coded symbols until it can decode. The code guarantees that if there were m erroneous symbols incorrectly classified as clean which seeped through for that position, then the destination can decode as soon as it receives $B + 2m$ coded symbols for that position, it does not have to wait to receive K symbols.

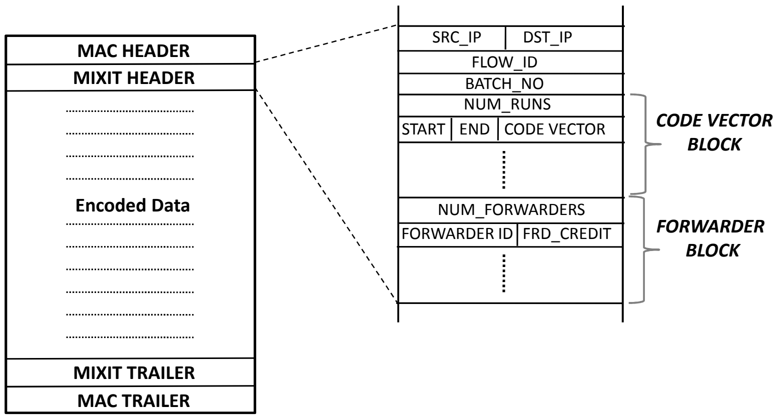


Figure 5-9: MIXIT's packet format.

- The rateless code allows MIXIT to provide flexible reliability semantics. Since the code works on groups of B symbols, there is no fate sharing across groups of B symbols. It's likely that when the destination receives a few packets, it will be able to decode most of the groups of B symbols, but not some since they had more errors. Depending on the application, the destination could wait to receive more coded symbols until it can decode, or ignore the undecoded symbols and ask the source to proceed to the next batch by sending a *batch-ack* to the source.

■ 5.8 Implementation

■ 5.8.1 Packet Format

MIXIT inserts a variable length header in each packet, as shown in Fig. 5-9. The header is also repeated as a trailer at the end of the packet to improve delivery in the face of collisions [65]. The header contains the source and destination addresses, the flow identifier, and the the batch identifier. These fields are followed by a variable length *Code Vector Block*, which describes how the symbols in this packet have been created. It has the format (Code Vector, Run Start, Run End); the values of these fields are obtained using the algorithm in Section 5.4.2. Following that is the variable length *Forwarder Block* that lists all the neighbors of this node ordered according to their C-ETS metrics. For each neighbor, the header also contains its credit assignment as described in Section 5.5. The *Code Vector Block* and the *Forwarder Block* are computed and updated by the forwarders. The other fields are initialized by the source and simply copied by each forwarder.

Experiment	Section	Result
MIXIT in a lightly loaded network	5.9.2	MIXIT improves median throughput by $2.1\times$ over MORE and $2.9\times$ over SPR
Impact of concurrency	5.9.2	MIXIT exploits loose packet delivery constraints to increase concurrency.
Impact of symbol level diversity	5.9.2	MIXIT with plain carrier sense still outperforms MORE by $1.5\times$.
Impact of mistake rate threshold	5.9.2	MIXIT's error correcting code allows us to be flexible with the mistake rate. This reduces the fraction of correct symbols incorrectly labeled dirty and increases throughput.
Impact of batch size	5.9.2	MIXIT is insensitive to batch size, providing large gains for sizes as small as 8.
MIXIT in a congested network	5.9.3	MIXIT improves median throughput by $2.8\times$ over MORE and $3.9\times$ over SPR.
Impact of forwarding algorithm	5.9.3	MIXIT's congestion-aware forwarding prevents hotspots and keeps network capacity from dropping during congestion.

Table 5-1: A summary of the major experimental contributions of this chapter.

■ 5.8.2 Node State

Each MIXIT node maintains per-flow state, which is initialized when the first packet from a flow that contains the node ID in the *Neighbor Block* arrives . The per-flow state includes:

- The *batch_buffer*, which stores the received clean symbols for each batch. This buffer is at most $K \times S$, where K is the batch size and S the packet size.
- The *credit_counter*, which stores the number of credits assigned to the node by the upstream neighbors for the batch. Upon the arrival of a packet from a node with a higher C-ETS, the node increments the credit by the corresponding credit assignment as indicated in the packet header.
- The *transmit_counter*, which is incremented by the credit assignment algorithm in Section 5.5. After a packet transmission, it decrements by one.

■ 5.8.3 Control Flow

MIXIT's control flow responds to packet receptions. On the receiving side, whenever a packet arrives, the node checks whether it's ID is present in the *Forwarder Block*. If it is, then it updates the *credit_counter* for the corresponding batch of that flow by the credit assigned to it in the *Forwarder Block*. Next, the node picks out clean symbols from the received packet using the SoftPHY hints and adds them to the *batch_buffer*. If the credit is greater than one, it runs the credit assignment algorithm from Section 5.5. It then creates *transmit_counter* coded packets using the technique in Section 5.4.2 and enqueues them. The MAC layer transmits these packets using the rule discussed in Section 5.6.1.

When the destination node receives a packet, it checks the symbol positions for which it has received at least B coded symbols and decodes whichever of them it can. It sends a batch-Ack to the source when it has decoded the required fraction (determined by the application's reliability requirements) of original symbols. The batch-Ack is sent periodically until packets from the next batch start arriving.

■ 5.9 Evaluation

We compare MIXIT with two routing protocols for wireless mesh networks: MORE, a state-of-the-art packet-level opportunistic routing protocol, and SPR, single path routing using the commonly used ETX metric. Our experimental results are summarized in Table 5-1.

■ 5.9.1 Testbed

We use a 25-node indoor testbed deployed in a lab. Each node is a Zigbee software radio. The hardware portion of the node is a Universal Software Radio Peripheral [59] with a 2.4 GHz daughterboard, the remainder of the node's functions (demodulation, channel decoding, network coding etc) are implemented in software. The peak data rate on the link is 250 Kbits/s when there are no other transmissions in progress. Paths between nodes are between one and five hops long, and the SNR of the links varies from 5 dB to 30 dB. The average packet loss rate on links in our network is 23% for 1500 byte packets.

■ 5.9.2 Single Flow

Throughput Comparison

Method: We run SPR, MORE, and MIXIT in sequence between 120 randomly picked source-destination pairs in our testbed. Each run transfers a 5 MByte file. The batch size of MIXIT is 12, but the error-correction preprocessing stage described in Section 5.7 converts it into 16 packets. To make a fair comparison, MORE uses a batch of 16 packets. We use the same batch sizes for MIXIT and MORE for all other experiments unless specifically noted otherwise. The packet size for all three protocols is 1500B. The mistake rate γ for MIXIT is fixed at 5% and the symbol size for MIXIT is 6 bytes unless otherwise noted. Before running an experiment, we collect measurements to compute pairwise packet delivery probabilities, which are then fed to SPR and MORE to be used in their route computations. The same measurement packets are used by MIXIT to compute the network's SNR profile as described in Section 5.6. We repeat the experiment for each source-destination pair five times and report the average throughput for each scheme.

Results: Fig. 5-10 plots the CDF of the throughput taken over 120 source-destination pairs in our testbed. MIXIT provides a median throughput gain of $2.1\times$ over MORE and $2.9\times$ over SPR.

We note that MIXIT improves performance across the entire throughput range. Packet-based opportunistic routing protocols, like MORE, provide large throughput gains for dead spots, i.e., scenarios where all paths between the source and destination are of poor quality. The gains for high quality paths were relatively minor [12, 18]. Both MORE and ExOR exploit diversity at the packet level to build better quality links out of many bad links. But for source-destination pairs that are connected via good links, diversity does not help. Naturally, this makes one wonder whether MIXIT's gains over packet based opportunistic routing protocols arise from its ability to exploit concurrency, a question that we address in the next section.

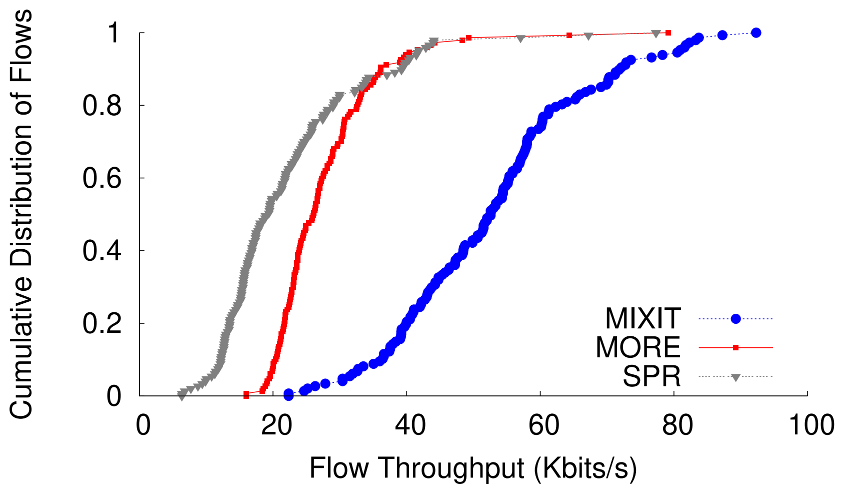


Figure 5-10: Throughput comparison. The figure shows that MIXIT has a median throughput gain of $2.1\times$ over MORE, the state-of-the-art packet level opportunistic routing protocol, and $2.9\times$ over SPR, a single path routing protocol based on the ETX metric.

Where do MIXIT's Throughput Gains Come From?

MIXIT exploits both wireless diversity and concurrent transmissions. We would like to measure how much each of these components contributes to MIXIT's throughput gains.

Method: We first compare MIXIT with a modified version of MORE that takes advantage of concurrency at the packet level, which we call MORE-C. Like MORE, MORE-C performs packet based opportunistic routing. But MORE-C also allows nodes to transmit concurrently. To check whether two transmissions should be transmitted concurrently, MORE-C uses the same algorithm used by MIXIT and described in Section 5.6, but after it replaces symbol delivery probabilities with packet delivery probabilities.

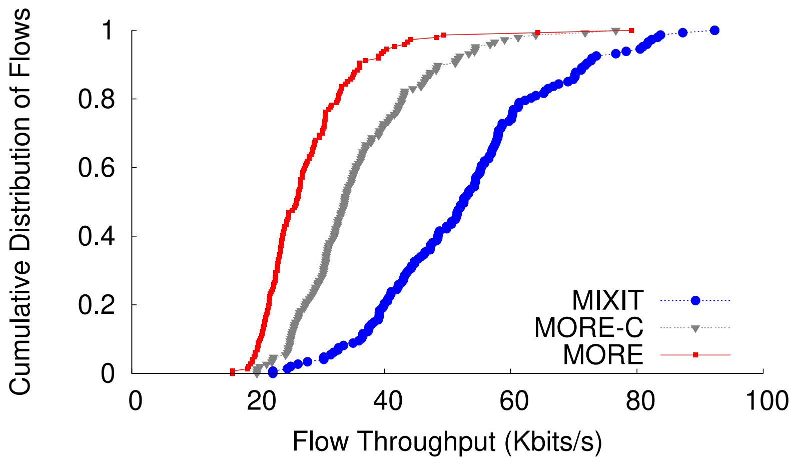


Figure 5-11: Impact of concurrency. The figure shows the throughput of MIXIT, MORE, and a concurrency-enabled version of MORE which we term MORE-C. Clearly concurrency helps but it is not sufficient to achieve the same throughput as MIXIT.

Results: Fig. 5-11 plots the CDF of the throughputs of MIXIT, MORE, and MORE-C taken over the same source-destination pairs as before. MIXIT provides a median throughput gain of $1.7\times$ over MORE-C. The main result is that even when compared against a protocol that exploits both diversity and concurrency like MORE-C, MIXIT still does significantly better. The only extra property that MIXIT has beyond MORE-C is its ability to work at the symbol level. Is the median gain of $1.7\times$ over MORE-C due mainly to MIXIT's ability to exploit clean symbols, i.e., is symbol-level diversity the dominant contributor to MIXIT's overall throughput gain?

Method: To answer the above question, we prevent MIXIT from aggressively exploiting concurrent transmissions and use plain carrier sense. The intent is to limit its gains over

MORE to be from being able to perform opportunistic routing over clean symbols. We call the resulting version MIXIT-CS.

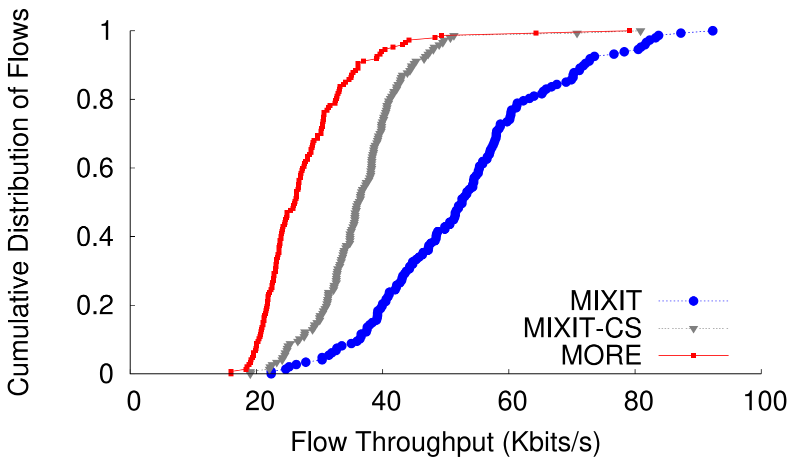


Figure 5-12: Throughputs for MIXIT with CS. The figure shows the throughput of MIXIT, MORE, and MIXIT-CS, a version of MIXIT which uses plain carrier sense and can only take advantage of symbol-level diversity. MIXIT-CS still performs better than MORE due to its ability to exploit long opportunistic receptions but with a few errors in them.

Results: Fig. 5-12 plots the CDF of the throughputs of MIXIT, MIXIT-CS and MORE. MIXIT-CS provides a median throughput gain of $1.5\times$ over MORE, i.e., significantly less gain than MIXIT. Thus, symbol-level diversity is not the dominant contributor to MIXIT's throughput gains. Indeed, comparing Fig. 5-12 with Fig. 5-11 shows that the overall gain of MIXIT over MORE is roughly *Gain of MIXIT-CS over MORE*$\times$*Gain of MORE-C over MORE*, i.e. $1.5 \times 1.4 = 2.1$. The multiplicative effect is due to the symbiotic interaction between concurrency and symbol-level opportunistic routing; concurrency tries to run the medium at high utilization and hence increases symbol error rate. But when the symbol error rate becomes high, almost every packet will have some symbols in error causing the whole packet to be dropped. Consequently, trying to exploit concurrency with a packet level protocol is limited by nature. Only a protocol that filters out incorrect symbols can push concurrency to its limits.

Impact of Letting More Errors Through

Method: We evaluate how the threshold on classifying clean symbols affects throughput. As explained in Section 5.3, MIXIT has the flexibility to choose the threshold mistake rate γ. We vary this threshold and compare the average throughput. For the Zigbee protocol,

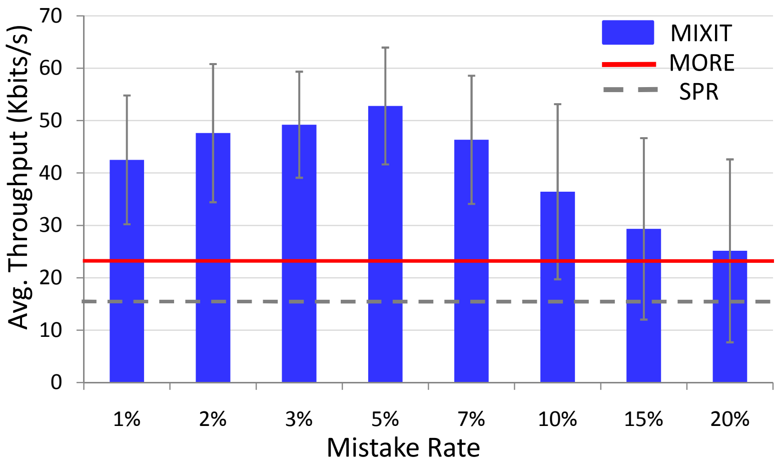


Figure 5-13: Impact of changing the mistake rate. The figure shows that many mistake rate thresholds provide significant throughput gains and hence MIXIT performs reasonably well even if the network is configured with a suboptimal threshold.

the PHY symbol is 4 bits long, while the MIXIT symbol size is 6 bytes.

Results: Fig. 5-13 plots the average throughput across all source-destination pairs for different mistake rates. The average throughput surprisingly increases as we let more errors through! It peaks when the mistake rate is around 5% and drops at higher error rates.

This may sound counter intuitive, but recall that we are talking about a *probability* of error; if the router would know for sure which PHY symbols are incorrect, the best it can do is to drop all incorrect PHY symbols. But a PHY symbol that has a 5% chance of being in error has also a 95% chance of being correct. For our topology, at 5% mistake rate, the cost of correcting the error end-to-end balances the opportunity of exploiting correct symbols that made it to their next hops, maximizing the throughput.

The right mistake rate threshold depends on the network. We assume that the administrator calibrates this parameter for her networks. A large mistake rate like 30% does not make sense for any network.[4] The results however show that a wide range of choices provide good throughput and outperform packet-based opportunistic routing.

Impact of Batch Size

We evaluate whether MIXIT's throughput is sensitive to batch size. Fig. 5-14 plots the throughput for batch sizes of $8, 12, 16$ and 32. The throughput is largely insensitive to the

[4]Even under optimal conditions, it takes at least two symbols to correct each incorrect symbol [136] and hence a mistake rate higher than 33% would never make sense.

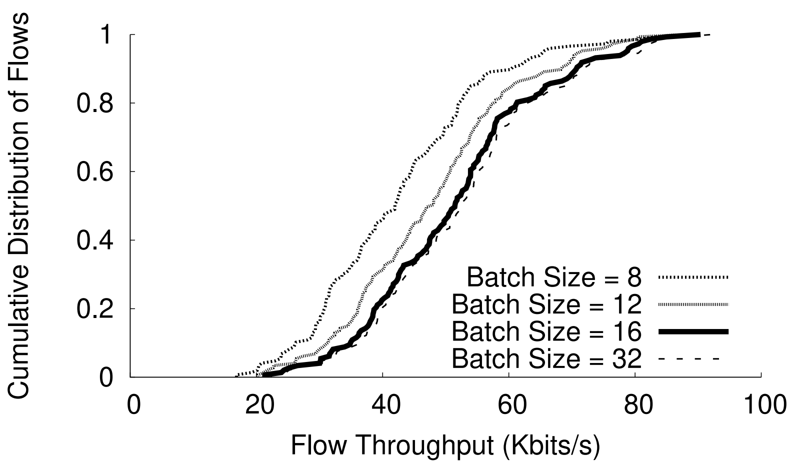


Figure 5-14: Impact of batch size. The figure shows the CDF of the throughput achieved by MIXIT for different batch sizes. It shows that MIXIT is largely insensitive to batch sizes.

batch size. The slight drop off at lower batch sizes is primarily because of higher overhead. A bigger batch size allows MIXIT to amortize the overhead over a larger number of packets, increasing throughput. Insensitivity to batch sizes allows MIXIT to vary the batch size to accommodate different transfer sizes. For any transfer larger than 8 packets, MIXIT shows significant advantages. Shorter transfers can be sent using traditional routing.

Comparison with Fragmented CRC

Method: MIXIT achieves significant gains due to its ability to perform opportunistic routing on clean symbols, which are classified using SoftPHY hints. We now investigate how much benefit we obtain from using SoftPHY. An alternative to SoftPHY is a fragmented CRC scheme [46]. The scheme splits each packet into multiple fragments, and sends multiple checksums per packet, one per fragment. This scheme allows the routers to identify which parts of a packet are correct and then apply symbol-level network coding on them. Fragment size is a design choice that affects end-to-end throughput. We pick 50 bytes to be the fragment size because it provides the highest median throughput in our experiments.

Results: Fig. 5-15 plots the CDF of throughputs obtained with MIXIT using the SoftPHY interface and MIXIT using the fragmented CRC scheme. The SoftPHY scheme has a median throughput gain of 56% over the fragmented CRC scheme. The main reason the fragmented CRC scheme performs worse is because of fate sharing across the entire fragment. Random bit errors are quite common on long lossy links as well as short high

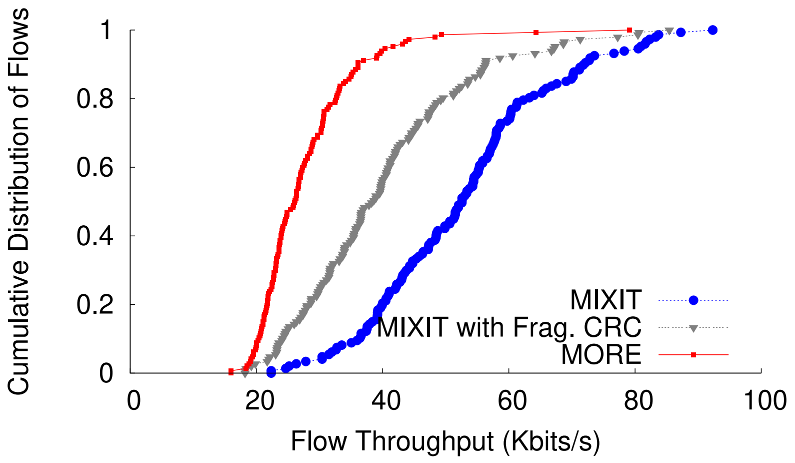


Figure 5-15: Comparison between MIXIT, MORE and MIXIT with the fragmented CRC scheme.

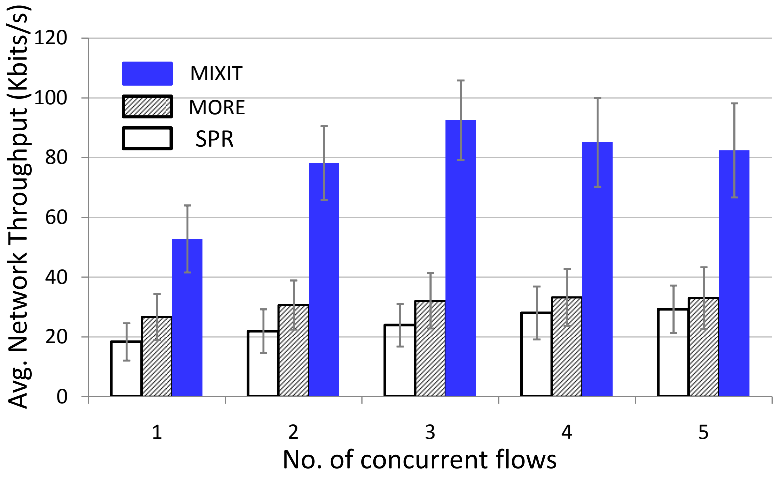


Figure 5-16: Average throughput with multiple active flows for MIXIT, MORE, and SPR. The figure shows that MIXIT's throughput scales as offered load increases until the network is saturated. MORE and SPR become similar as load increases and perform worse than MIXIT because of their inability to exploit concurrency opportunities.

quality links in the presence of concurrent transmissions. The fragmented CRC scheme wastes entire fragments, while the SoftPHY based scheme is more efficient in discarding incorrect bits.

■ 5.9.3 Multiple Flows

Throughput Comparison

Method: We run MIXIT, MORE and SPR in sequence, varying the number of random active flows in the network. The rest of the setup is similar to the single flow case. We run

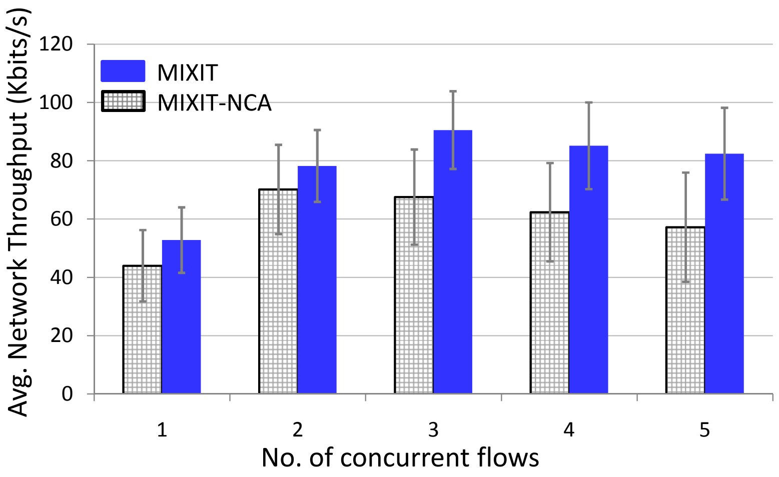


Figure 5-17: The role of congestion-aware forwarding. The figure shows that congestion-aware forwarding is particularly important when the number of active flows is large.

50 experiments for each choice of number of flows, with each experiment repeated 5 times. We calculate the average throughput for each run.

Results: Fig. 5-16 plots the average throughput for MIXIT, MORE, and SPR with increasing number of flows. We see that MIXIT's throughput gain generally increases with load, and at its peak reaches $2.8\times$ over MORE and $3.9\times$ over SPR.

The higher gains as load increases are due to MIXIT's ability to aggressively exploit concurrency and perform congestion-aware forwarding. Both MORE and SPR, which rely on carrier sense, become conservative in accessing the medium as the number of flows increases. Thus, they cannot fully exploit the spatial diversity in the network. MIXIT however, can maintain high levels of concurrency because of its ability to deal with partially correct packets.

The throughput gains drop slightly as the network gets heavily congested. The primary reason is hidden terminals, whose effect is exacerbated by the fact that the USRP nodes, which perform all processing in user mode on the PC, do not have support for synchronous acks, and thus cannot quickly detect hidden terminals and backoff.

Impact of Congestion Aware Forwarding

Method: We evaluate the impact of MIXIT's congestion-aware forwarding component on performance. Node congestion is built into MIXIT's routing algorithm due to its use of the backlog parameter $Q(i)$, the number of symbols queued up at node i yet to be transmitted. Nodes that are backlogged will not be assigned credits by their upstream parents and thus traffic will be routed around hotspots. We compare this scheme with one where this component is disabled. Specifically, parent nodes assign credits to their downstream nodes based only on the path quality, i.e. based on the path ETS, and ignore congestion information. We call this scheme MIXIT-NCA, for MIXIT with "No Congestion Aware" forwarding.

Results: Fig. 5-17 plots the average throughput for MIXIT and MIXIT-NCA for increasing number of flows. The figure shows that congestion-aware forwarding accounts for 30% of the throughput gain at high load. As load increases, the probability of the network experiencing local hotspots increases. MIXIT-NCA does not route around such hotspots, and insists on pushing the same amount of information through regardless of congestion. MIXIT adaptively routes around these hotspots and therefore increases throughput.

■ 5.10 Related Work

Laneman et al. [79] develop and analyze a series of information-theoretic schemes to exploit wireless co-operative diversity. MIXIT builds on the intuition, but with two important differences that admit a practical design. First, intermediate nodes use SoftPHY hints to "clean" the symbols before processing and forwarding them, rather than just receiving, combining, and forwarding information at the signal level. Second, nodes use intra-flow symbol-level network coding, which allows them to coordinate and collaborate without requiring finely synchronized transmissions that many "co-operative diversity" approaches entail.

MIXIT builds on prior work on opportunistic routing [12, 18], spatial diversity [96], and wireless network coding [72]. In particular, it shares the idea of intra-flow network coding with MORE [18], but with three key differences: first, MORE operates on packets and cannot deal with packets with errors; second, MIXIT's symbol-level network code is an end-to-end rateless *error correcting code* while MORE's network code cannot correct errors; and third, MIXIT designs a MAC which exploits the looser constraints on packet delivery to significantly increase concurrent transmissions, MORE uses carrier sense and requires correct packet delivery which prevents it from achieving high concurrency. MIXIT's network code also builds on recent advances in extending network coding to scenarios with errors and adversaries [62, 76]. In contrast to all these schemes, MIXIT only codes over symbols above a certain confidence threshold, while using coding coefficients that reduce overhead.

MIXIT also builds on prior work on "soft information", whose benefits are well known [124, 51, 128]. Soft information refers to the confidence values computed in some physical layers when it decodes symbols. Recent work [65] has developed the SoftPHY interface to expose this information to higher layers in a PHY-independent manner by annotating bits with additional hints. Thus far, the use of these hints at higher layers has been limited to improving link reliability by developing better retransmission schemes [65] or to combine confidence values over a wired network to reconstruct correct packets from erroneous receptions [132]. In contrast, MIXIT uses SoftPHY hints in a new way, eschewing link-layer reliability in favor of spatial diversity to achieve high throughput and reliability.

■ 5.11 Discussion

A key finding of MIXIT is that routers need not forward fully correct packets to achieve end-to-end reliability, and that loosening this constraint significantly increases network throughput. With MIXIT, as long as each symbol in every transmitted packet is correctly received by *some* downstream node, the packet is highly likely to be delivered to the destination correctly. Designing a network that has this attractive property is not an easy task because it needs to scalably coordinate overlapping symbol receptions and cope with erroneous symbol propagation. MIXIT solves these problems using a symbol-level network code that has an end-to-end rateless error correction component.

Instead of using link-layer error detection and recovery, MIXIT treats the entire wireless network as a single logical channel whose component links could run at high error rates. Because MIXIT can cope with individually high error rates, it encourages an aggressive MAC protocol that greatly increases concurrency compared to CSMA.

Although MIXIT exploits cross-layer information, its architecture is modular and layered: it can run atop any radio and PHY that provide suitable confidence hints, with the routers being oblivious to the end-to-end error correction mechanism. The gains may vary depending on the PHY and MAC used, but it can be used in any multi-hop wireless network with the following properties:

1. *Computational capabilities:* The coding/decoding algorithms in MIXIT are more de-

manding than traditional store and forward networks. In our proof-of-concept software implementation on software radios, the algorithms can achieve at most an effective throughput of 4.7Mb/s. But as shown in [112], we can build the entire decoder in hardware using simple shift registers, similar to traditional Reed-Solomon (RS) hardware decoders that achieve speeds of 80 Gigabits per second [82]. Hence, we believe that computational considerations will not limit the applicability of our algorithms at high data rates.

2. *Memory:* MIXIT's nodes need to store packets from recent batches. The default batch size is 16, and typically there are two or three batches in flight, requiring storage space of roughly 70 KBytes, a modest amount for modern communication hardware.

The ideas in MIXIT may be applicable in sensor networks to ship data to sink nodes. Because most traffic in these networks is uni-directional, data from different sensors can be coded together to improve throughput. In addition, MIXIT could also be used to multicast data in a mesh network. Because all destinations require the same data, routers can keep transmitting coded data until all destinations can decode them.

CHAPTER 6

Discussion & Conclusion

Despite the fact that wireless is increasingly the preferred mode of network access, today's wireless systems are not equipped to meet the demands of emerging high bandwidth applications. Current deployed wireless mesh networks are built using a framework rooted in wired network design, which ultimately limits throughput. This dissertation advocates an alternative architecture built around network coding and shows that it can provide large throughput and reliability gains. We conclude by examining the benefits of our network-coded wireless architecture and the remaining challenges.

■ 6.1 Network Coding: An Alternative Design for Wireless Networks

Current wireless networks are designed using the wired network as the blueprint. The design abstracts the wireless channel as a point-to-point link and grafts wired network protocols onto the wireless environment. Wireless channels however are fundamentally different, and making them obey such a wired design conflicts with the intrinsic characteristics of the wireless medium.

- The current design imposes an artificial point-to-point abstraction on the wireless channel, but wireless links (with omni-directional antennas) are broadcast links, and hence can deliver a packet to multiple destinations in one transmission, a property that the current design fails to exploit.

- Because of their broadcast nature, simultaneous wireless transmissions may interfere. The current wireless design considers interference to be always harmful, tries to avoid it as much as possible, and ignores it when it still happens (like in a hidden terminal scenario) [48].
- The current design tries to ensure reliability via retransmissions. Wireless networks, however enjoy spatial diversity, and hence, though a packet may not be received by its intended next hop, it is likely received by some node along its path to the destination. The current design ignores these lucky receptions and retransmits the packet until it is correctly received by its pre-determined next hop.
- The current design imposes layer isolation, where the lower layers deliver fully correct packets that the network layer routes towards their destinations. While this isolation has worked properly for the low-error wired network, it imposes strict limitations on their unpredictable error-prone wireless counterparts. In wireless networks, even when no node receives a packet correctly, any given bit is likely to be received correctly by some node closer to the destination than the transmitter. The current design fails to harness these correctly received bits to improve reliability.

Network coding presents a unified tool for harnessing the intrinsic properties of the wireless medium to improve throughput and reliability, as follows.

- Network coding can harness wireless broadcast to increase throughput. Wireless networks exhibit a large overlap in the information available to the nodes because at each hop the broadcast nature of a wireless transmission delivers the same packet to multiple nodes within the transmitter's radio range. We have shown that COPE, a network coding protocol, can leverage this redundancy to compress the transmitted data, delivering the same information in fewer transmissions, and thus improving the overall throughput.
- We have also shown that analog network coding (ANC) can exploit strategic interference to increase concurrency in the network, thus increasing network throughput and demonstrating that certain types of interference can be quite useful.
- Further, in MIXIT, we have shown that it is often unnecessary to retransmit a corrupted packet; our symbol-level network code leverages spatial diversity to recover

a correct packet from bits that were correctly received at various routers. It funnels groups of correct bits toward their destination, allowing the destination to recover the whole packet.

- Finally, rather than having the layers work in isolation, network coding allows the physical and network layers to cooperate on their common goal of high throughput and reliability. In particular, MIXIT loosens the contract between the layers and discards the correct packet abstraction. It allows the PHY and link layers to deliver likely correct bits, which the network layer routes to their destination. This new contract is enabled by a novel symbol-level network code that operates over groups of likely correct bits called symbols. The network code also functions as a rateless end-to-end error correcting code, allowing the destination to correct any erroneous bits that might seep through.

This dissertation builds the above ideas into a practical network-coded wireless architecture. Further, it integrates network coding within the current stack, develops practical coding and decoding algorithms, provides prototype implementations of the proposed designs, and testbed evaluations that show large performance gains in comparison with the state-of-the-art wireless network design. Thus, we believe that this work makes a strong case for an alternative wireless network design based on network coding.

■ 6.2 Remaining Challenges & Future Work

The systems in this dissertation addressed the major challenges involved in bringing the theory of network coding to practice and integrating it into the current network stack. Below, we enumerate a few challenging issues that the dissertation does not solve. These are not fundamental limitations however, we believe they can be solved within the network coding framework.

Coding is typically more computationally expensive than a store-and-forward approach, which has a linear complexity in the packet length. The systems in this dissertation strive to achieve linear coding and decoding complexity. We succeeded with COPE and ANC, both of which have linear complexity. MIXIT's network code, however, has quadratic complexity, primarily due to the sophisticated decoding algorithm used for correcting errors at the destination. These codes have the same structure as Reed-Solomon codes for which extremely efficient decoders exist. Hence we believe that even a MIXIT-style coding can have a relatively low computationally cost.

Network coding also brings up interesting questions in how to provide reliability in wireless mesh networks. Typically, TCP is used to provide end-to-end reliability. But as we saw in the evaluation of COPE in Section 3.5.4, TCP's performance suffers due to packet losses induced by collisions. On the other hand, the network code in MIXIT automatically provides end-to-end reliability without even requiring link-level reliability. Of course, TCP serves other purposes apart from reliability, mainly congestion control. An interesting avenue for future research is how to combine them, i.e., how to obtain reliability from network coding while keeping the congestion control functionality from TCP. MIXIT provides a partial answer, using back-pressure techniques to provide load balanced forwarding inside the network. But a general congestion control protocol for network-coded mesh networks remains an open problem.

Our work raises interesting bit-rate adaptation issues. Existing algorithms work on the assumption that the underlying link is a unicast link even though the medium itself is broadcast. Thus, they optimize bit-rates for the channel to the corresponding receiver on the unicast link. Our systems, however, exploit the broadcast nature of wireless. For example, in COPE, a coded packet is intended for multiple next hops. Rate-adaptation algorithms therefore have to find the best rate to maximize the delivery rate to all intended next-hops. Similarly, in MIXIT, we benefit from wireless spatial diversity where multiple next hops overhear the same packet, possibly with errors. The overhearing capability depends on the bit-rate used, and increases with lower bit-rates. Thus there is an interesting tradeoff, a lower bit-rate means longer time on the channel, but also increases overhearing which increases gains. How to make this tradeoff in a practical system remains an open problem.

The 802.11 technology provides multiple wireless channels. Our work has focused on one channel. It does not prevent the usage of multiple 802.11 channels, but it treats them independently. Recent work has shown that a joint design that assigns channels to links in a way that reduces interference improves the throughput of mesh networks [109]. Though the network coding techniques are orthogonal to the underlying channel allocation algorithm, the gains we get might be impacted. Specifically, both COPE and MIXIT depend on nodes overhearing packets; if nodes are operating on different frequencies, the possibility of overhearing will be reduced, potentially reducing gains. It is an interesting problem to determine what is the tradeoff in using multiple channels to reduce interference compared to the reduced overhearing opportunities, and the consequent smaller network coding gains.

■ 6.3 Looking Ahead

At the end of the day, this dissertation is about building an inter-disciplinary approach to design future wireless networks. The success of wireless networks is due to contributions from both electrical engineers and computer scientists. So far, however, these two groups have proceeded largely in isolation, having agreed a few decades ago that their contract would be a digital one: the electrical engineers would design components that present correct packets to the computer scientists, and in return, could ignore network layer questions; while computer scientists would design the network layer and overlook physical layer details. Techniques such as coding, soft information, back-pressure have been limited to the physical layer or worse, confined to the theoretical literature; while system designers have proceeded along using dated designs. This dissertation revisits this contract and questions whether this divide is suitable in *every* context. In particular, we show that, for wireless networks, by poking a hole in this contract, i.e., by employing network coding combined with richer information exchange between the physical and network layer, we can substantially increase network capacity. We believe that, because of the substantial gains possible, this inter-disciplinary approach should be the bedrock for future wireless system design.

References

[1] D. Aguayo, J. Bicket, S. Biswas, G. Judd, and R. Morris. Link-level measurements from an 802.11b mesh network. In *Proceedings of the ACM SIGCOMM*, Portland, Oregon, August 2004. (Cited on pages 49, 57, 89 and 100.)

[2] R. Ahlswede, N. Cai, S.-Y. R. Li, and R. W. Yeung. Network information flow. *IEEE Transactions on Information Theory*, 46(4):1204–1216, July 2000. (Cited on pages 28, 44, 47, 92 and 132.)

[3] H. Balakrishnan, V. Padmanabhan, S. Seshan, and R. Katz. A comparison of mechanisms for improving TCP performance over wireless links. *IEEE/ACM Transactions on Networking*, 5(6):756–769, December 1997. (Cited on pages 53 and 70.)

[4] H. Balakrishnan, S. Seshan, E. Amir, and R. Katz. Improving TCP/IP performance over wireless networks. In *1st ACM Conf. on Mobile Computing and Networking*, Berkeley, CA, November 1995. (Cited on page 53.)

[5] Z. Bar-Yossef, Y. Birk, T. S. Jayram, and T. Kol. Index coding with side information. In *FOCS*, pages 197–206, 2006. (Cited on page 37.)

[6] C. Bennett and G. Ross. Time-domain electromagnetics and its applications. *Proceedings of the IEEE*, 66(3):299–318, 1978. (Cited on page 50.)

[7] E. R. Berlekamp. *Algebraic Coding Theory*. Aegean Park Press, 1984. (Cited on page 161.)

[8] P. Bhagwat, B. Raman, and D. Sanghi. Turning 802.11 inside-out. In *HotNets*, 2003. (Cited on page 89.)

[9] V. Bharghavan, A. J. Demers, S. Shenker, and L. Zhang. MACAW: A media access protocol for wireless LAN's. In *ACM SIGCOMM*, 1994. (Cited on pages 50 and 132.)

[10] K. Bhattad and K. R. Narayanan. Weakly secure network coding. In *NetCod*, April 2005. (Cited on page 48.)

[11] J. Bicket. Bit-rate selection in wireless networks. Master's thesis, Massachusetts Institute of Technology, February 2005. (Cited on page 50.)

[12] S. Biswas and R. Morris. Opportunistic routing in multi-hop wireless networks. In *ACM SIGCOMM*, Philadelphia, PA, USA, 2005. (Cited on pages 32, 33, 36, 52, 78, 93, 135, 136, 137, 167 and 174.)

[13] J. Byers, M. Luby, and M. Mitzenmacher. A digital fountain approach to asynchronous reliable multicast. *IEEE Journal on Selected Areas of Communications*, 20(8):1528–1540, 2002. (Cited on page 47.)

[14] N. Cai and R. Yeung. Secure network coding. In *Proc. IEEE International Symposium on Information Theory*, 2002. (Cited on page 48.)

[15] N. Cai and R. W. Yeung. Network error correction, II: Lower bounds. *Commun. Inf. Syst.*, 6(1):37–54, September 2006. (Cited on page 156.)

[16] J. F. Cardoso. Blind signal separation: Statistical principles. *Proceedings of the IEEE*, 86(10):2009–2025, October 1998. (Cited on page 132.)

[17] C. Cetinkaya and E. Knightly. Opportunistic traffic scheduling over multiple network paths. In *Proc. INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, volume 3, pages 1928–1937 vol.3, 2004. (Cited on page 52.)

[18] S. Chachulski, M. Jennings, S. Katti, and D. Katabi. Trading structure for randomness in wireless opportunistic routing. In *ACM SIGCOMM*, Kyoto, Japan, 2007. (Cited on pages 8, 32, 33, 35, 36, 52, 94, 135, 136, 137, 150, 152, 167 and 174.)

[19] R. Chakravorty, S. Katti, I. Pratt, and J. Crowcroft. Using TCP flow-aggregation to enhance data experience of cellular wireless users. *Selected Areas of Communications, IEEE Journal on*, 23(6):1190–1204, June 2005. (Cited on page 53.)

[20] C. Chekuri, C. Fragouli, and E. Soljanin. On average throughput and alphabet size in network coding. *Information Theory, IEEE Transactions on*, 52(6):2410–2424, June 2006. (Cited on page 45.)

[21] C.-C. Chen, E. Seo, H. Kim, and H. Luo. Select: Self-learning collision avoidance for wireless networks. *IEEE Journal on Mobile Communications*, 7(3):305–321, March 2008. (Cited on page 86.)

[22] P. Chou, Y. Wu, and K. Jain. Practical network coding. In *41st Annual Allerton Conference on Communication, Control, and Computing*, Oct. 2003. (Cited on page 48.)

[23] R. Choudhury, X. Yang, R. Ramanathan, and N. Vaidya. On designing MAC protocols for wireless networks using directional antennas. *IEEE Transactions on Mobile Communications*, 5(5):477–491, 2006. (Cited on page 51.)

[24] I. C. S. L. M. S. Committee. *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*. IEEE Std. 802.11, New York, New York, 1997. (Cited on page 50.)

[25] R. Comroe and J. Costello, D. ARQ schemes for data transmission in mobile radio systems. *IEEE Journal on Selected Areas of Communications*, 2(4):472–481, July 1984. (Cited on page 137.)

[26] T. Cover and A. E. Gamal. Capacity theorems for the relay channel. *IEEE Transactions on Information Theory*, 25(5):572–584, September 1979. (Cited on pages 95 and 117.)

[27] D. S. J. De Couto, D. Aguayo, J. Bicket, and R. Morris. A high-throughput path metric for multi-hop wireless routing. In *9th ACM International Conference on Mobile Computing and Networking (MobiCom*, San Diego, CA, USA, September 2003. (Cited on pages 51, 63, 73, 82 and 93.)

[28] S. Deb and R. Srikant. Global stability of congestion controllers for the internet. *IEEE/ACM Transactions on Networking*, 48(6):1055–1060, June 2003. (Cited on pages 37 and 152.)

[29] R. Dougherty, C. Freiling, and K. Zeger. Insufficiency of linear coding in network information flow. *IEEE Transactions on Information Theory*, 51(8):2745–2759, 2005. (Cited on page 48.)

[30] P. Erdos and A. Renyi. On a classical problem of probability theory. *Magyar Tud. Akad. Mat. Kutato Int. Kozl*, 1961. (Cited on page 46.)

[31] E. Erez and M. Feder. Convolutional network codes. In *Proc. International Symposium on Information Theory ISIT*, pages 146–, 2004. (Cited on page 48.)

[32] E. Erez and M. Feder. Efficient network codes for cyclic networks. In *Proc. International Symposium on Information Theory ISIT*, pages 1982–1986, 2005. (Cited on page 48.)

[33] Ericsson. Definition and assessment of relay based cellular deployment concepts for future radio scenarios considering 1st protocol characteristics. Chapter 5. https://www.ist-winner.org/DeliverableDocuments/D3.4.pdf. (Cited on pages 94 and 100.)

[34] J. Feldman, T. Malkin, R. A. Servedio, and C. Stein. On the capacity of secure network coding. In *42nd Annual Allerton Conference on Communication, Control, and Computing*, Sept./Oct 2004. (Cited on page 48.)

[35] P. W. F.K. Hwang, D.S. Richards. The Steiner tree problem. *Annals of Discrete Mathematics*, 53, 1992. (Cited on page 45.)

[36] C. Fragouli, D. Katabi, A. Markopoulou, M. Medard, and H. Rahul. Wireless network coding: Opportunities &challenges. In *Proc. IEEE Military Communications Conference MILCOM 2007*, pages 1–8, October 2007. (Cited on pages 37 and 132.)

[37] C. Fragouli and A. Markopoulou. Network coding techniques for network monitoring: a brief introduction. In *Proc. International Zurich Seminar on Communications*, pages 82–83, Zurich, Switzerland, February 2006. (Cited on page 48.)

[38] C. Fragouli and E. Soljanin. A connection between network coding and convolutional codes. In *Proc. IEEE International Conference on Communications*, volume 2, pages 661–666 Vol.2, 2004. (Cited on page 48.)

[39] C. Fragouli and E. Soljanin. Decentralized network coding. In *IEEE Information Theory Workshop*, pages 310–314, 2004. (Cited on page 48.)

[40] C. Fragouli, J. Widmer, and J.-Y. le Boudec. On the benefits of network coding for wireless applications. In *4th International Symposium on Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks*, pages 1–6, April 2006. (Cited on page 92.)

[41] G. FSF. GNU Radio - GNU FSF Project. http://www.gnu.org/software/gnuradio. (Cited on pages 36 and 119.)

[42] Z. Fu, P. Zerfos, H. Luo, S. Lu, L. Zhang, and M. Gerla. The impact of multihop wireless channels on TCP throughput and loss. In *Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, volume 3, pages 1744–1753, April 2003. (Cited on page 86.)

[43] E. Gabidulin. Theory of codes with maximal rank distance. *Probl. Peredachi Inf*, 21(1):3–16, January 1985. (Cited on pages 156 and 157.)

[44] E. Gabidulin, A. Ourivski, B. Honary, and B. Ammar. A new family of rank codes and applications to cryptography. In *Proc. IEEE International Symposium on Information Theory*, 2002. (Cited on page 156.)

[45] D. Ganesan, R. Govindan, S. Shenker, and D. Estrin. Highly-resilient, energy-efficient multipath routing in wireless sensor networks. 5(4):11–25, 2001. (Cited on page 52.)

[46] R. K. Ganti, P. Jayachandran, H. Luo, and T. Abdelzaher. Datalink streaming in wireless sensor networks. In *ACM SenSys*, 2006. (Cited on page 171.)

[47] C. Gkantsidis and P. Rodriguez. Network coding for large scale content distribution. In *IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, volume 4, pages 2235–2245, March 2005. (Cited on pages 48 and 132.)

[48] S. Gollakota and D. Katabi. Zigzag decoding: Combating hidden terminals in wireless networks. In *ACM SIGCOMM*, Seattle, WA, USA, August 2008. (Cited on pages 51, 133 and 178.)

[49] R. Gowaikar, A. Dana, R. Palanki, B. Hassibi, and M. Effros. On the capacity of wireless erasure networks. In *Proc. International Symposium on Information Theory ISIT 2004*, pages 401–, 2004. (Cited on page 48.)

[50] P. Gupta and P. Kumar. Towards an information theory of large networks: An achievable rate region. *Information Theory, IEEE Transactions on*, 49(8):1877–1894, August 2003. (Cited on page 55.)

[51] J. Hagenauer and P. Hoeher. A Viterbi algorithm with soft-decision outputs and its applications. In *IEEE Global Telecommunications Conference, and Exhibition. Communications Technology for the 1990s and Beyond. GLOBECOM '89*, pages 1680–1686 vol.3, 1989. (Cited on page 175.)

[52] D. Halperin, T. Anderson, and D. Wetherall. Practical interference cancellation for wireless LANs,. In *ACM MOBICOM*, San Francisco, CA, USA, September 2008. (Cited on pages 51 and 133.)

[53] J. Hamkins. An analytic technique to separate cochannel FM signals. *IEEE Transactions on Information Theory*, 48(11):2980–2989, April 2000. (Cited on pages 123, 128 and 132.)

[54] N. J. A. Harvey, D. R. Karger, and K. Murota. Deterministic network coding by matrix completion. In *16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, page 489498, Jan 2005. (Cited on page 48.)

[55] M. Heusse, F. Rousseau, R. Guiller, and A. Duda. Idle sense: An optimal access method for high throughput and fairness in rate diverse wireless lans. In *ACM SIGCOMM*, Philadelphia, PA, USA, August 2005. (Cited on page 93.)

[56] T. Ho, B. Leong, R. Koetter, M. Medard, M. Effros, and D. Karger. Byzantine modification detection in multicast networks with random network coding. *Information Theory, IEEE Transactions on*, 54(6):2798–2803, June 2008. (Cited on pages 28 and 48.)

[57] T. Ho, M. Medard, and R. Koetter. An information-theoretic view of network management. *Information Theory, IEEE Transactions on*, 51(4):1295–1312, April 2005. (Cited on page 48.)

[58] T. Ho, M. Medard, R. Koetter, D. Karger, M. Effros, J. Shi, and B. Leong. A random linear network coding approach to multicast. *IEEE Transactions on Information Theory*, 52(10):4413–4430, October 2006. (Cited on pages 28, 42, 48, 92, 132, 144 and 145.)

[59] E. Inc. Universal software radio peripheral. http://ettus.com. (Cited on pages 119 and 166.)

[60] M. Inc. Meraki. http://meraki.com/. (Cited on page 49.)

[61] S. Jaggi, M. Langberg, T. Ho, and M. Effros. Correction of adversarial errors in networks. In *Proc. International Symposium on Information Theory ISIT*, pages 1455–1459, 2005. (Cited on page 48.)

[62] S. Jaggi, M. Langberg, S. Katti, T. Ho, D. Katabi, and M. Medard. Resilient network coding in the presence of byzantine adversaries. In *26th IEEE International Conference on Computer Communications. IEEE*, pages 616–624, May 2007. (Cited on pages 28, 48, 156 and 174.)

[63] S. Jaggi, P. Sanders, P. A. Chou, M. Effros, S. Egner, K. Jain, and L. Tolhuizen. Polynomial time algorithms for multicast network code construction. *IEEE Transactions on Information Theory*, 51(6):1973–1982, June 2005. (Cited on pages 28, 42, 48 and 132.)

[64] K. Jain, L. Lovasz, and P. A. Chou. Building scalable and robust peer-to-peer overlay networks for broadcasting using network coding. In *Symposium on Principles of Distributed Computing*, page 5159, 2005. (Cited on page 48.)

[65] K. Jamieson and H. Balakrishnan. PPR: Partial Packet Recovery for wireless networks. In *ACM SIGCOMM*, Kyoto, Japan, 2007. (Cited on pages 32, 135, 137, 142, 143, 155, 164 and 175.)

[66] E.-S. Jung and N. H. Vaidya. A power control MAC protocol for ad hoc networks. In *8th Annual International Conference on Mobile computing and Networking*, pages 36–47, Atlanta, Georgia, USA, 2002. (Cited on page 51.)

[67] A. Kamra, V. Misra, J. Feldman, and D. Rubenstein. Growth codes: Maximizing sensor network data persistence. *SIGCOMM Comput. Commun. Rev.*, 36(4):255–266, August 2006. (Cited on page 94.)

[68] B. Karp and H. Kung. Greedy perimeter stateless routing for wireless networks. In *Sixth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom 2000)*, pages 243–254, Boston, MA, USA, August 2000. (Cited on page 53.)

[69] S. Katti, S. Gollakota, and D. Katabi. Embracing wireless interference: Analog network coding. In *ACM SIGCOMM*, Kyoto, Japan, August 2007. (Cited on pages 8, 11 and 143.)

[70] S. Katti, D. Katabi, H. Balakrishnan, and M. Medard. Symbol-level network coding for wireless mesh networks. In *ACM SIGCOMM*, Seattle, WA, August 2008. (Cited on pages 8 and 11.)

[71] S. Katti, I. Maric, A. Goldsmith, D. Katabi, and M. Medard. Joint relaying and network coding in wireless networks. In *IEEE International Symposium on Information Theory ISIT*, pages 1101–1105, 2007. (Cited on page 118.)

[72] S. Katti, H. Rahul, D. Katabi, W. H. M. Medard, and J. Crowcroft. XORs in the air: Practical wireless network coding. In *ACM SIGCOMM*, Pisa, Italy, 2006. (Cited on pages 8, 11, 35 and 174.)

[73] R. Khalili and K. Salamatian. On the capacity of erasure relay channel: multi-relay case. In *Proc. IEEE Information Theory Workshop*, pages 5 pp.–, 2005. (Cited on page 48.)

[74] B.-J. Ko, V. Misra, J. Padhye, and D. Rubenstein. Distributed channel assignment in multi-radio 802.11 mesh networks. In *Proc. IEEE Wireless Communications and Networking Conference (WCNC)*, pages 3978–3983, 2007. (Cited on page 50.)

[75] Y.-B. Ko and N. H. Vaidya. Location-aided routing (LAR) in mobile ad hoc networks. In *Mobile Computing and Networking (MOBICOM)*, pages 66–75, Dallas, TX, USA, October 1998. (Cited on pages 52 and 93.)

[76] R. Koetter and F. R. Kschischang. Coding for errors and erasures in random network coding. *Information Theory, IEEE Transactions on*, 54(8):3579–3591, August 2008. (Cited on pages 28, 48, 156 and 174.)

[77] R. Koetter and M. Medard. An algebraic approach to network coding. *IEEE/ACM Transactions on Networking*, 11(5):782–795, October 2003. (Cited on pages 48, 62, 92, 132 and 141.)

[78] E. Kohler. *The Click modular router*. PhD thesis, Massachusetts Institute of Technology, November 2000. (Cited on pages 81 and 82.)

[79] J. Laneman, D. Tse, and G. Wornell. Cooperative diversity in wireless networks: Efficient protocols and outage behavior. *IEEE Transactions on Information Theory*, 50(12):3062–3080, December 2004. (Cited on pages 35, 52, 132 and 174.)

[80] J. Laneman and G. Wornell. Distributed space-time-coded protocols for exploiting cooperative diversity in wireless networks. *IEEE Transactions on Information Theory*, 49(10):2415–2425, October 2003. (Cited on page 132.)

[81] A. Lee and M. Medard. Simplified random network codes for multicast networks. In *Proc. International Symposium on Information Theory ISIT*, pages 1725–1729, 2005. (Cited on page 48.)

[82] H. Lee. A high-speed low-complexity Reed-Solomon decoder for optical communications. *IEEE Transactions on Circuits and Systems*, 52(8):461–465, August 2005. (Cited on page 176.)

[83] A. R. Lehman and E. Lehman. Complexity classification of network information flow problems. In *41st Annual Allerton Conference on Communication, Control, and Computing*, Oct. 2003. (Cited on page 48.)

[84] J. Li, C. Blake, D. D. Couto, H. I. Lee, and R. Morris. Capacity of ad hoc wireless networks. In *MOBICOM*, August 2001. (Cited on page 55.)

[85] S.-Y. Li, R. Yeung, and N. Cai. Linear network coding. *Information Theory, IEEE Transactions on*, 49(2):371–381, February 2003. (Cited on pages 28, 47, 92 and 132.)

[86] Z. Li and B. Li. Network coding: The case for multiple unicast sessions. In *Allerton*, Monticello, IL, USA, September 2004. (Cited on pages 76 and 92.)

[87] Z. Li and B. Li. Efficient and distributed computation of maximum multicast rates. In *Proc. IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies INFOCOM 2005*, volume 3, pages 1618–1628 vol. 3, 2005. (Cited on page 48.)

[88] Z. Li, B. Li, D. Jiang, and L. C. Lau. On achieving optimal throughput with network coding. In *Proc. IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies INFOCOM*, volume 3, pages 2184–2194 vol. 3, 2005. (Cited on page 48.)

[89] J. Liu, D. Goeckel, and D. Towsley. Bounds on the gain of network coding and broadcasting in wireless networks. In *26th IEEE International Conference on Computer Communications (INFOCOM). IEEE*, pages 724–732, May 2007. (Cited on page 93.)

[90] D. Lun, M. Medard, and M. Effros. On coding for reliable communication over packet networks. In *Allerton*, Allerton, IL, USA, September 2004. (Cited on pages 28, 47 and 48.)

[91] D. Lun, N. Ratnakar, M. Medard, R. Koetter, D. Karger, T. Ho, E. Ahmed, and F. Zhao. Minimum-cost multicast over coded packet networks. *Information Theory, IEEE Transactions on*, 52(6):2608–2623, June 2006. (Cited on pages 28 and 92.)

[92] S. Mascolo, C. Casetti, M. Gerla, M. Y. Sanadidi, and R. Wang. TCP Westwood: Bandwidth estimation for enhanced transport over wireless links. In *MobiCom '01: Proceedings of the 7th annual international conference on Mobile computing and networking*, Rome, Italy, 2001. (Cited on page 53.)

[93] F. J. McWilliams and N. J. A. Sloane. *The Theory of Error-Correcting Codes*. North-Holland, 1977. (Cited on page 143.)

[94] M. Medard, M. Effros, D. Karger, and T. Ho. On coding for non-multicast networks. In *41st Annual Allerton Conference on Communication, Control, and Computing*, Oct. 2003. (Cited on page 48.)

[95] E. C. V. D. Meulen. Three-terminal communication channels. *Adv. Appl. Prob.*, 3:120–154, June 1971. (Cited on pages 96 and 131.)

[96] A. Miu, H. Balakrishnan, and C. E. Koksal. Improving loss resilience with multi-radio diversity in wireless networks. In *11th annual international conference on Mobile computing and networking (Mobicom)*, 2005. (Cited on pages 33, 138, 146 and 174.)

[97] K. Mueller and M. Muller. Timing recovery in digital synchronous data receivers. *IEEE Transactions on Communications*, 24(5):516–531, May 1976. (Cited on page 130.)

[98] M. Muuss. Test TCP. http://ftp.arl.mil/ftp/pub/ttcp/. (Cited on page 82.)

[99] P. Pakzad, C. Fragouli, and A. Shokrollahi. Coding schemes for line networks. In *Proc. International Symposium on Information Theory ISIT 2005*, pages 1853–1857, September 2005. (Cited on page 47.)

[100] J.-S. Park, M. Gerla, D. Lun, Y. Yi, and M. Medard. Codecast: A network-coding-based ad hoc multicast protocol. *IEEE Wireless Communications Magazine*, 13(5):76–81, October 2006. (Cited on pages 28 and 132.)

[101] K. Park, G. Kim, and M. Crovella. On the relationship between file sizes, transport protocols, and self-similar network traffic. In *International Conference on Network Protocols*, pages 171–180, October 1996. (Cited on pages 82, 85 and 88.)

[102] I. . W. part 11a/11b/11g. Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications. Standard Specification, IEEE, 1999. (Cited on pages 70 and 132.)

[103] V. Paxson and S. Floyd. Wide area traffic: the failure of poisson modeling. *IEEE/ACM Transactions on Networking*, 3(3):226–244, June 1995. (Cited on page 82.)

[104] R. Pickholtz, L. Milstein, and D. Schilling. Spread spectrum for mobile communications. *IEEE Journal on Vehicular Technology*, 40(2):313–322, May 1991. (Cited on page 132.)

[105] J. P. R. Draves and B. Zill. Comparison of routing metrics for multi-hop wireless networks. In *SIGCOMM*, Portland, OR, USA, August 2004. (Cited on pages 51 and 93.)

[106] B. Radunovic, C. Gkantsidis, P. Key, and P. Rodriguez. An optimization framework for opportunistic multipath routing in wireless mesh networks. In *The 27th IEEE Conference on Computer Communications (INFOCOM)*, pages 2252–2260, April 2008. (Cited on pages 59 and 152.)

[107] A. Ramamoorthy, J. Shi, and R. Wesel. On the capacity of network coding for wireless networks. In *41st Annual Allerton Conference on Communication Control and Computing*, October 2003. (Cited on page 132.)

[108] A. Ramamoorthy, J. Shi, and R. Wesel. On the capacity of network coding for random networks. *IEEE Transactions on Information Theory*, 51(8):2878–2885, 2005. (Cited on page 48.)

[109] A. Raniwala and T. Chiueh. Architecture and algorithms for an ieee 802.11-based multi-channel wireless mesh network. In *Proc. Twenty-fourth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, volume 3, pages 2223–2234, 2005. (Cited on page 180.)

[110] K. Ratnam and I. Matta. WTCP: An efficient mechanism for improving TCP performance over wireless links. In *Third IEEE Symposium on Computers and Communications ISCC*, pages 74–78, June 1998. (Cited on pages 53 and 93.)

[111] C. Reis, R. Mahajan, M. Rodrig, D. Wetherall, and J. Zahorjan. Measurement-based models of delivery and interference in static wireless networks. In *ACM SIGCOMM*, 2006. (Cited on page 154.)

[112] G. Richter and S. Plass. Fast decoding of rank-codes with rank errors and column erasures. In *International Symposium on Information Theory (ISIT)*, pages 398–398, June 2004. (Cited on pages 156, 161 and 176.)

[113] S. Riis. Linear versus non-linear boolean functions in network flow. In *Conference on Information Sciences and Systems (CISS 2004)*, Mar. 2004. (Cited on page 48.)

[114] Y. Sagduyu and A. Ephremides. Crosslayer design for distributed MAC and network coding in wireless ad hoc networks. In *Proc. International Symposium on Information Theory ISIT 2005*, pages 1863–1867, 2005. (Cited on page 48.)

[115] P. Sanders, S. Egner, and L. M. G. M. Tolhuizen. Algorithms for network information flow. In *15th ACM Symposium on Paralel Algorithms and Architectures*, June 2003. (Cited on page 45.)

[116] S. Sengupta, S. Rayanchu, and S. Banerjee. An analysis of wireless network coding for unicast sessions: The case for coding-aware routing. In *26th IEEE International Conference on Computer Communications(INFOCOM)*, pages 1028–1036, May 2007. (Cited on pages 59, 63, 92 and 93.)

[117] C. E. Shannon. Two-way communication channels. In *4th Berkeley Symposium Math. Stat. Prob.*, volume 1, pages 611–644, 1961. (Cited on pages 95, 96 and 131.)

[118] D. Silva, F. Kschischang, and R. Koetter. A rank-metric approach to error control in random network coding. In *Proc. IEEE Information Theory Workshop on Information Theory for Wireless Networks*, July 2007. (Cited on pages 37 and 156.)

[119] R. Sinha. Internet packet size distributions: Some observations. http://netweb.usc.edu/ rsinha/pkt-sizes/. (Cited on page 66.)

[120] D. Son, B. Krishnamachari, and J. Heidemann. Experimental analysis of concurrent packet transmissions in wireless sensor networks. In *Fourth ACM SenSys Conference*, Boulder, Colorado, USA, November 2006. (Cited on page 154.)

[121] E. Sozer, M. Stojanovic, and J. Proakis. Underwater acoustic networks. *IEEE Journal on Oceanic Engineering*, 25(1):72–83, January 2000. (Cited on page 100.)

[122] Y. C. T. Ho and K. J. Han. On constructive network coding for multiple unicasts. In *44th Allerton Conference on Communication, Control and Computing*, 2006. (Cited on pages 76 and 92.)

[123] L. Tassiulas and A. Ephremides. Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks. *IEEE Trans. on Automatic Control*, 37(12), 1992. (Cited on page 35.)

[124] D. Tse and P. Vishwanath. *Fundamentals of Wireless Communications*. Cambridge University Press, 2005. (Cited on pages 33, 50, 104, 132, 138, 143 and 175.)

[125] A. Tsirigos and Z. Haas. Analysis of multipath routing, part 2: Mitigation of the effects of frequently changing network topologies. *IEEE Transactions on Wireless Communications*, 3(2):500–511, 2004. (Cited on page 52.)

[126] S. Verdu. *Multiuser Detection*. Cambridge University Press, 1998. (Cited on page 132.)

[127] M. Vutukuru, K. Jamieson, and H. Balakrishnan. Harnessing exposed terminals in wireless networks. In *5th USENIX Symposium on Networked Systems Design and Implementation*, San Francisco, CA, April 2008. (Cited on pages 51, 152 and 155.)

[128] M. Wang, W. Xiao, and T. Brown. Soft decision metric generation for qam with channel estimation error. *IEEE Transactions on Communications*, 50(7):1058–1061, July 2002. (Cited on page 175.)

[129] WCN. Wireless community network list. http://www.toaster.net/wireless/community.html. (Cited on page 89.)

[130] J. Widmer and J.-Y. L. Boudec. Network coding for efficient communication in extreme networks. In *SIGCOMM WDTN*, 2005. (Cited on page 132.)

[131] A. Willig, M. Kubisch, C. Hoene, and A. Wolisz. Measurements of a wireless link in an industrial environment using an IEEE 802.11-compliant physical layer. *IEEE Transactions on Industrial Engineering*, 49(6):1265–1282, December 2002. (Cited on page 146.)

[132] G. Woo, P. Kheradpour, and D. Katabi. Beyond the bits: Cooperative packet recovery using PHY information. In *ACM MobiCom*, Montreal, Canada, 2007. (Cited on pages 137, 142, 143 and 175.)

[133] Y. Wu, P. Chou, and K. Jain. A comparison of network coding and tree packing. In *IEEE International Symposium on Information Theory*, June 2004. (Cited on page 42.)

[134] Y. Wu, P. Chou, and S.-Y. Kung. Minimum-energy multicast in mobile ad hoc networks using network coding. *IEEE Transactions on Communications*, 53(11):1906–1918, 2005. (Cited on page 48.)

[135] S.-Y. K. Y. Wu, P. A. Chou. Information exchange in wireless networks with network coding and physical-layer broadcast. In *39th Annual Conference on Information Sciences and Systems (CISS)*, Baltimore, MD, Mar. 2005. (Cited on page 48.)

[136] R. W. Yeung and N. Cai. Network error correction, I: Basic concepts and upper bounds. *Commun. Inf. Syst.*, 6(1):19–35, September 2006. (Cited on pages 156 and 170.)

[137] S. Zhang, S. Liew, and P. Lam. Physical layer network coding. In *ACM MOBICOM*, 2006. (Cited on pages 96 and 131.)

[138] Y. Zhu, B. Li, and J. Guo. Multicast with network coding in application-layer overlay networks. *IEEE Journal on Selected Areas of Communications*, 22(1):107–120, 2004. (Cited on page 48.)