A

SenseCode: Network Coding for Reliable Sensor Networks

LORENZO KELLER, EMRE ATSAN, KATERINA ARGYRAKI
and
CHRISTINA FRAGOULI, Ecole Polytechnique Federale Lausanne (EPFL)

Designing a communication protocol for sensor networks often involves obtaining the "right" trade-off between energy efficiency and end-to-end packet error rate. In this paper, we show that network coding provides a means to elegantly balance these two goals. We present the design and implementation of SenseCode, a collection protocol for sensor networks—and, to the best of our knowledge, the first such implemented protocol to employ network coding. SenseCode provides a way to gracefully introduce a configurable amount of redundant information in the network, thereby increasing end-to-end packet error rate in the face of packet loss. We compare SenseCode to the best (to our knowledge) existing alternative and show that it reduces end-to-end packet error rate in highly dynamic environments, while consuming a comparable amount of network resources. We have implemented SenseCode as a TinyOS module and evaluate it through extensive TOSSIM simulations.

Categories and Subject Descriptors: C.2.0 [**General**]: Data communications; C.2.2 [**Network Protocols**]: Protocol architecture

General Terms: Algorithms, Design, Reliability

Additional Key Words and Phrases: Data collection, Network coding, Multi-path

ACM Reference Format:
ACM Trans. Sensor Netw. V, N, Article A (January YYYY), 20 pages.
DOI = 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

1. INTRODUCTION

Sensor networks are promising to transform the way we deal with the physical world, with emerging applications ranging from environmental or animal monitoring, to health care and military surveillance. Each of these applications has different performance requirements and resource constraints, yet most of them have two needs in common: energy efficiency and low end-to-end packet error rates. In sensor networks, these are typically competing goals: decreasing end-to-end packet error rates requires introducing redundant information in the network (for example, transmitting each packet multiple times to compensate for packet loss), which, in turn, leads to higher energy consumption.

Researchers have already proposed multipath communication as a way to balance these two goals. However, existing multipath protocols leave room for improvement: they typically maintain multiple, preferably disjoint paths between communicating end-points, often resulting in complicated routing protocols; yet, in certain cases, they fail to relay a packet to its destination, even though a viable path does exist.

Author's address: L. Keller, EPFL IC IIF ARNI, Station 14, CH-1015 Lausanne
This work was funded in part by the Hasler Foundation ManCom Project No 2072 and the Swiss National Science Foundation Award No PP00P2128639.

DOI 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

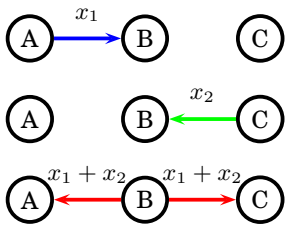


Fig. 1. A pattern that does *not* occur in many-to-one communication, e.g., collection of information from multiple sensor nodes to a common sink: Nodes A and C exchange packets x_1 and x_2 via relay B. B receives x_1 and x_2 and broadcasts their binary xor, $x_1 + x_2$. Node A uses its knowledge of x_1 to retrieve x_2, and node C uses x_2 to retrieve x_1.

In this paper, we show that network coding provides an alternative, elegant way to balance energy efficiency and end-to-end packet error rate. We present SenseCode, a collection protocol for sensor networks, which allows to gracefully introduce a configurable amount of redundant information in the network. SenseCode relies on network coding to enable a new form of multipath communication, where each node disseminates information through all available paths, without having to explicitly discover or monitor these paths—hence, without having to maintain multiple routing structures. Moreover, it uses simple, decentralized algorithms that do not exceed the sensors' modest processing and memory capabilities. To the best of our knowledge, our work provides the first actual implementation of a collection protocol for sensor networks that relies on network coding.

Network-coding ideas and techniques have already been implemented over wireless mesh networks [Wu et al. 2005; Li and Li 2006; Chachulski et al. 2007; Katti et al. 2008]. A natural question is, how are sensor networks different—why can we not simply apply to them the same ideas and techniques. The answer is that sensor networks differ in traffic patterns, performance metrics, and the amount of available processing and memory resources. SenseCode is applicable to sensor networks that require many-to-one communication, where a large number of sources send messages to a common sink. In such networks, the traffic patterns for which network coding is known to offer benefits (such as the well known pattern shown in Figure 1) do not typically occur. Moreover, while existing work focuses on throughput gains and bandwidth efficiency, our design goal is decreased end-to-end packet error rate at low energy cost.

Our contributions can be summarized as follows:

(1) We present SenseCode, a new collection protocol for sensor networks, which leverages network coding to balance energy efficiency and end-to-end packet error rate. It uses simple, decentralized algorithms that do not exceed the sensors' modest processing and memory capabilities.
(2) We have implemented SenseCode as a TinyOS [Levis et al. 2005] module. We evaluate it through TOSSIM [Levis et al. 2003] simulations.
We compare its performance to the collection tree protocol (CTP) [Gnawali et al. 2009], the best, to our knowledge, existing alternative, and show that, in highly dynamic environments, it achieves lower end-to-end packet error rate, while consuming a comparable amount of network resources.

The rest of the paper is organized as follows. Section 2 describes the basic idea behind our approach; Section 3 describes SenseCode's design and implementation; Section 4 presents our evaluation; Section 5 discusses alternatives; Section 6 reviews related work; and Section 7 concludes the paper.

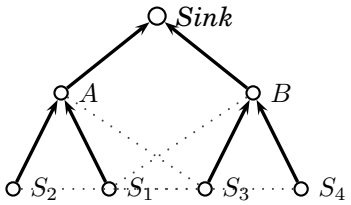


Fig. 2. A sensor network, where nodes form a tree rooted at the sink. Each source S_i, $i = 1 \dots 4$, has a message x_i to communicate to the sink. Each node sends packets only to its parent, however, each node can overhear the transmissions not only of its children, but of multiple neighbors. S_1 overhears S_2 and S_3; S_2 overhears S_1; S_3 overhears S_1 and S_4; S_4 overhears S_3; A overhears S_3; and B overhears S_1.

2. SETUP

2.1. Problem Statement and Background

We consider a sensor network that operates in rounds: in each round, each sensor node i needs to communicate a message x_i to a common collecting sink. We focus on the problem of designing a *collection protocol*, i.e., a communication protocol that enables the nodes' messages to reach the sink.

Our goal is to design a collection protocol that achieves a desired end-to-end packet error rate while consuming as little energy as possible. We define *end-to-end (E2E) packet error rate* as the average fraction of messages generated by the sources that are not successfully communicated to the sink during each round. This metric captures a protocol's ability to communicate source messages to the sink in the face of packet losses, which may be due to channel and node failures or simply congestion—in general, events that alter the connectivity graph of the network. For instance, E2E packet error rate 0.5 means that, on average, half of nodes could not successfully communicate to the sink their message during each round (in our setup each source produces one message per round). *Energy efficiency* refers to the amount of energy that the protocol consumes to communicate a certain amount of information to the sink. In sensor networks, one of the primary sources of energy consumption is the radio, hence, we want to design a collection protocol that achieves a certain E2E packet error rate while minimizing the time of radio utilization.

Our basis for comparison will be the Collection Tree Protocol (CTP) [Gnawali et al. 2009], which, to the best of our knowledge, is the best existing collection protocol in terms of E2E packet error rate and energy efficiency. In CTP, the nodes maintain a tree rooted at the sink; each node that has a message to communicate sends that message to its parent node; each node that receives a message from one of its children forwards it to its own parent. CTP achieves energy efficiency by trying to build a "shortest-path tree," i.e., a tree that connects each node to the sink through the path that results in the minimum number of packet transmissions. CTP achieves low E2E packet error rates by dynamically adapting the tree to network conditions: neighboring nodes continuously exchange information on the quality of the channel between them; when a node detects that its connectivity to its parent is degrading, it switches to a different parent. Note that information on channel quality is mostly piggy-backed on data traffic so as to not significantly affect the protocol's energy efficiency.

2.2. Why a New Collection Protocol

Although CTP—and, in general, dynamic tree-based protocols—fare well in the face of many failure scenarios (e.g., when a channel degrades or a node fails), there are certain cases where we can do better. One limitation of tree-based protocols is that, at any point in time, the currently used tree was selected based on *past* network conditions. As

a result, it is possible that a message does not reach the sink, even though there does exist a path that would allow it to do so—but that path has not yet been "discovered" and incorporated in the tree. The following example illustrates such a scenario.

Example 2.1. Consider the network depicted in Figure 2, where each source S_i has a message x_i to communicate to the sink. The network runs a tree-based collection protocol, i.e., each source sends its message to its parent node, which then forwards it to the sink. Suppose node A acknowledges receiving messages x_1 and x_2 and then, before forwarding them to the sink, stops operating or the channels that connect it to its neighbors deteriorate. As a result, x_1 and x_2 are lost, even though they could have reached the sink through alternative paths (x_1 through B, and x_2 through S_1 and B). Hence, during this particular round, we achieve E2E packet error rate 0.5 (only half the messages reach the sink), though we could have achieved E2E packet error rate 0.

A tree-based collection protocol cannot recover from such a failure, because the sending nodes (S_1 and S_2, in our example) cannot know that their parent will fail, hence cannot choose in advance the right parent. This motivated us to consider a collection protocol that automatically leverages all available paths in the network, without having to explicitly discover them and incorporate them in the tree.

2.3. Main Idea

To preserve messages in the face of packet loss, we introduce redundant information in the network. In SenseCode, nodes still maintain a tree (as in CTP), however, each node propagates not only its own messages and the information sent by its children, but also a small amount of information that it has overheard from its neighbors. More specifically, each node propagates *linear combinations* of its own message, the packets it has received from its children, and a small number of the packets overheard from its neighbors—essentially, each node performs network coding. The following example illustrates how this approach improves E2E packet error rate in the face of packet loss inside the network.

Example 2.2. Consider again the network depicted in Figure 2, where each source S_i has a message x_i to communicate to the sink. Suppose that each node tries to send to its parent the packets specified in Table I. As in Example 2.1, node A acknowledges receiving the packets sent by its children, then fails to forward anything to the sink. As a result, the sink receives only the 4 packets sent by node B; these, however, contain 4 linearly independent combinations of the 4 messages x_i, which means that the sink can form a system of equations and recover all 4 messages. Hence, during this particular round, we achieve E2E packet error rate 0, even though half of the packets intended for the sink were lost.

In this example, each piece of information travels through multiple paths in the network, for instance, x_2 reaches both node A (as part of a message directly sent from S_2 to its parent A) and node B (as part of the linear combination sent by S_1 and overheard by node B). In the end, all 4 messages reach the sink, albeit "mixed" with one another into linear combinations. Since there are 4 messages, the sink needs to receive 4 linearly independent combinations of these messages in order to recover all of them. Notice that each of nodes A and B does send 4 linearly independent combinations; hence, the sink can recover all 4 messages whether node A or node B fails.

This decreased E2E packet error rate in the face of packet loss comes at the cost of extra packet transmissions: to communicate 4 messages to the sink, the sources introduce 2×4 packets in the network. In general, as we will see, SenseCode enables a straightforward E2E packet error rate/energy efficiency trade-off, where, to communicate N messages to the sink, we introduce $R \times N$ packets in the network, and, in

Table I. List of packets sent in Figure 2

Node	Overheard	Sent
S_1	x_2, x_3	$x_1, x_1 + x_2 + x_3$
S_2	x_1	$x_2, x_1 + x_2$
S_3	x_1, x_4	$x_3, x_1 + x_3 + x_4$
S_4	x_3	$x_4, x_3 + x_4$
A	$x_3, x_1 + x_3 + x_4$	$x_1, x_2, x_3, x_3 + x_1 + x_4$
B	$x_1, x_1 + x_2 + x_3$	$x_3, x_4, x_1, x_1 + x_2 + x_3$

The contents of the packets that were overheard and sent by each node from Figure 2 during the scenario described in Example 2.1. The sink can decode the linear combinations through the use of coding vectors.

exchange, we achieve E2E packet error rate 0 as long as *any* set of N packets reach the sink.

2.4. Why Coding

One could argue that, to introduce redundant information in the network, nodes could simply send each message multiple times, or forward some (or all) of the packets they overhear—why perform linear combinations, which add to the processing load of the nodes? The reason is that, compared to these alternatives, network coding can achieve lower E2E packet error rates in a more energy-efficient manner.

To illustrate, we consider the simplified model[1] of Figure 3, where a single source wants to communicate N messages x_i, $i = 1 \ldots N$, to a sink, over a single channel with packet-erasure probability ϵ. Suppose we are interested in a communication protocol that achieves E2E packet error rate 0, i.e., allows the source to communicate all N messages to the sink.

One possible communication protocol is for the source to copy each message x_i into a separate packet, hence send N packets to the sink. This protocol introduces no redundancy (the source sends each message exactly once) and achieves E2E packet error rate ϵ (of the N messages—and packets—sent to the sink, the latter is expected to receive $(1 - \epsilon)N$).

Now suppose that, to decrease the E2E packet error rate, the source is willing to introduce redundant information by a factor of 2, i.e., if it needs y bits to represent the N messages, it is willing to send $2y$ bits to the sink. Coding theory tells us that this amount of redundancy is, theoretically, sufficient for achieving E2E packet error rate 0, as long as $\epsilon < 0.5$; in other words, if the source sends twice as many bits as necessary to represent its messages, the sink should be able to recover all messages as long as it receives at least half the bits sent to it [MacWilliams and Sloane 1983]. Hence, given a communication protocol with redundancy factor 2, we will consider it "good," if it approaches this behavior (achieves E2E packet error rate close to 0 as long as $\epsilon < 0.5$).

We consider three collection protocols with redundancy factor 2.

(1) **No coding**: The source copies each message x_i into 2 separate packets. To successfully receive message x_i, the sink only needs to receive one of the two packets.
(2) **Full coding**: The source produces $2 \times N$ linear combinations of the N messages such that any N of these combinations form a linearly independent set and copies each linear combination into a packet. In this case, a single packet is not useful by itself—it does not carry a full message. However, as long as the sink receives *any*

[1]We do not use this model to prove any properties about sensor networks or SenseCode, merely to illustrate the intuition behind why network coding is more energy-efficient than the alternatives.

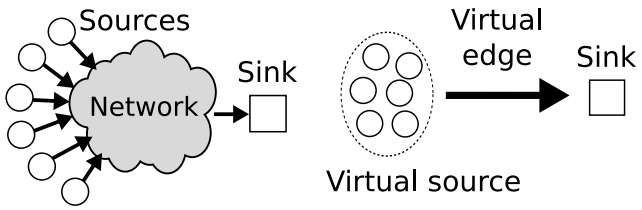


Fig. 3. Abstraction of a sensor network as a point-to-point network, where all sources are co-located, and packet loss is captured with a single erasure channel that connects the collocated sources to the sink.

of the N packets sent to it, it can form a linear system of equations and solve it to recover all N messages.

(3) **Systematic coding**: The source sends N "uncoded" packets, each containing a message x_i, as well as N "coded" packets, each containing a linear combination of the N messages.

Figure 4 shows the E2E packet error rate of these three protocols, as a function of the erasure probability ϵ. First, we see that non-coded communication achieves E2E packet error rate 0 only when $\epsilon = 0$; thereafter, its packet error rate increases with ϵ. In contrast, fully-coded communication achieves E2E packet error rate near 0 as long as $\epsilon < 0.45$; thereafter, its E2E packet error rate sharply increases. The reason is that, as long as fewer than half of the transmitted packets get lost, the sink receives at least N linear combinations of the N messages, hence, can successfully recover all of them; however, if more than half of the transmitted packets get lost, then the sink receives fewer than N linear combinations of the N messages and, as a result, cannot recover.

So, coded communication clearly outperforms non-coded communication in terms of E2E packet error rate. However, it comes at the cost of longer packets: To decode a received coded packet, the sink needs to know the linear combination that the packet contains. This can be achieved by appending to each packet what is called a *coding vector*, which specifies the contained linear combination [Chou et al. 2003]. This coding vector can constitute a significant fraction of the payload, especially when the messages sent by the source are short.

Systematic communication reduces the coding-vector overhead, by employing coding (and incurring the corresponding overhead) only in half of the transmitted packets. Thus, it offers a practical trade-off between non-coded and fully-coded communication, i.e., performs as well as the former for $\epsilon < 0.5$ in terms of E2E packet error rate, yet significantly reduces the coding-vector overhead. This is the approach we adopt for SenseCode.

The challenge in implementing coded or systematic communication in a sensor network lies in the fact that the information that needs to be communicated to the sink is distributed throughout the network: unlike the setting depicted in Figure 3, in a sensor network, there is no single source that knows all N messages, hence, no straightforward way to create linearly independent combinations of these messages.

Our approach is to implement what we term *spatial coding*, where a packet may enter the network "uncoded" (i.e., carrying a single message), and the "coding" (i.e., mixing with other messages) happens opportunistically along the way. I.e., before forwarding the packet to the next hop toward the sink, each intermediate node further "encodes" it by linearly combining its contents with other information it has collected.

To achieve low E2E packet error rates, it is important that the sink receives linearly independent combinations of many (ideally all) messages. This, in turn, requires that each intermediate node collect and mix messages (or linearly independent com-

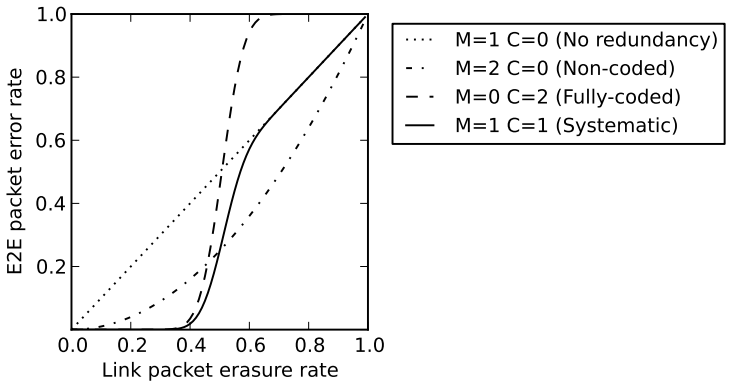


Fig. 4. End to end packet error rate as a function of packet loss, for different communication protocols. Assume that the virtual source from Figure 3 has $N = 36$ messages to communicate to the sink. We plot E2E packet error rate as a function of the erasure probability ϵ, when the virtual source sends M uncoded and C coded packets per message.

binations of messages) from multiple other nodes. Such mixing can be achieved in two ways: First, through a small amount of opportunistic overhearing. The nature of wireless networks is such that, when one node transmits a packet, that is potentially overheard by multiple neighbors. In Section 4.2, we show that, even if nodes overhear a small fraction of the traffic transmitted by their neighbors, they are able to acquire enough information to properly perform spatial coding. Second, mixing can be achieved by leveraging routing dynamics. If the tree changes during a round, the packets flowing along a certain path carry information from packets previously flowing through different paths. I.e., there is more information mixing when the routing topology is unstable—precisely the situation in which the E2E packet error rate needs to be improved.

To summarize, we propose a collection protocol for sensor networks which, instead of propagating messages to the sink, propagates linear combinations of these messages; these linear combinations are opportunistically added to packets as the latter travel though the network.

3. THE SENSECODE PROTOCOL

In this section, we describe the design and implementation of our protocol.

3.1. Design

Our protocol consists of the following building blocks:

Topology Construction. The sensor nodes use a routing protocol to maintain a routing structure that allows them to communicate packets to the sink. SenseCode can work on top of any routing protocol. It only requires that, at any point in time, each node has one or more "parents" (the next-hop neighbors to which the node addresses all the packets it transmits). For simplicity, and without loss of generality, we will assume that each node has a single parent.

Node Operation. Each node maintains two queues:

— The *storage queue* is used for mixing the node's own messages with information received or overheard from other nodes. When we say that the node "adds" packet

x to the storage queue, we mean that the node does the following: for each slot of the storage queue, it creates a linear combination of x and the current contents of the slot (using a different coding coefficient for each slot), then replaces the current contents of the slot with the outcome.

— The *transmit queue* stores the packets that the node wants to send to its parent. When we say that the node "adds" packet x to the transmit queue, we mean that the node inserts x in an empty slot of the transmit queue; if the transmit queue is full, x is dropped. When a node adds a packet to the transmit queue, the node tries to send this packet to its parent using a reliable MAC layer (discussed below).

A node updates its queues in the following cases:

(1) When the node has a new message to communicate to the sink. In this case, the node does the following: First, it creates a new packet x that includes the new message, marks x as "uncodable," and adds x to both the storage queue and the transmit queue; we call any such packet (which includes a single message from a single node) an *original* packet. Second, the node creates $R-1$ packets that are linear combinations of x and the packets in the storage queue, marks them as "codable," and adds them to the transmit queue; R is a configurable parameter, called *redundancy factor*.
(2) When the node receives a packet y from a child. If y is an uncodable packet, the receiving node adds it both to the storage queue and the transmit queue. If y is a codable packet, the receiving node first adds y to the storage queue, then creates a linear combination of y and the contents of the storage queue and adds the result to the transmit queue.
(3) When the node overhears a packet y, it adds y to the storage queue.

When a node creates a packet that is a linear combination of other packets, it appends a coding vector to the payload to enable decoding of the packet by other nodes [Fragouli et al. 2006]. We discuss the overhead of coding vectors in Section 5.2.

In summary, each node that has a new message to communicate to the sink sends out R packets (1 uncodable, the rest codable), while each node that acts as an intermediary relays exactly as many (uncodable and codable) packets as it receives from its children. At the end of each round, a total of $R \times N$ packets have been sent toward the sink, where N is the number of nodes that had a new message to communicate during the round. Each of the packets received by the sink is a linear combination of the N original packets. To recover all N messages, the sink needs to receive N linearly independent combinations.

MAC Layer. CTP uses a reliable MAC layer, where a sending node transmits each packet multiple times until (1) it receives an acknowledgment from the parent or (2) a maximum number of retransmissions is reached.

SenseCode uses the same MAC layer, but with the following modification: when a node transmits a codable packet x, if the packet is not acknowledged by the parent, the node transmits a *new* linear combination of x and the packets in the node's transmit queue (instead of simply retransmitting). This has the following effect: when a node sends a packet to its parent, even if the parent does not receive it, some other neighbor(s) may overhear it; by sending new information with every "retransmission," we are supplying these overhearing neighbors with more information, hence, increasing the level of information spreading in the network.

Controlling Information Redundancy. The redundancy factor R enables a well defined E2E packet error rate/energy-efficiency trade-off: Assuming that information is sufficiently spread across the network, a redundancy factor of R allows the sink to

recover *all* information sent by the nodes, even if the network loses $(R-1)N$ of the packets sent. This happens at the cost of each sensor generating R (as opposed to 1) packets for each new piece of information. R does not have to be an integer: we can set it, for instance, to 1.5, in which case each sensor generates *on average* 1.5 packets for each new piece of information. Note that this trade-off, although conceptually straightforward, is not easily achievable with traditional multi-path protocols.

3.2. Implementation Choices

We now describe and justify our basic implementation choices.

Amount of Overhearing. On a first thought, storing overheard packets seems to conflict with the goal of energy-efficiency: Being in listening mode (necessary to receive or overhear packets) typically consumes comparable energy with being in transmitting mode [Texas Instruments 2007]. This has led to the design of energy-efficient sensor platforms, where each node is mostly in sleeping mode and wakes up only when it senses that a packet is being sent to it.

Our implementation is compatible with such energy-efficient platforms: Each node stores only 10% of the packets it overhears. We chose this value, because, based on our experiments, storing more overheard packets does not improve the protocol's performance (Section 4.2). Hence, we can integrate our implementation into an energy-efficient sensor platform, by programming each node to wake up, not only when it senses a packet sent to it, but also, with probability $X\%$, when it senses a packet sent to a different node.

Storage Queue Size and Operations. In our implementation, the transmit queue has 13 slots (the default value in CTP) and the storage queue has 4 slots. We chose this value because, based on our experiments, a larger value does not improve the protocol's performance.

In an earlier implementation attempt, we ensured that each packet added to the storage queue contributed new information, i.e., was linearly independent from all the packets already in the queue—an optimization frequently used with network coding [Fragouli et al. 2006]–[Ho and Lun 2008]. However, this optimization required performing Gaussian elimination [Horn and Johnson 1990] for each newly received or overheard packet. We tested this operation on a TelosB mote and found that it took several milliseconds to complete, hence, would introduce non-negligible latency. It turned out that, in practice, if we use a field of size 16 or larger, this optimization does not improve the protocol's performance.

Recent work claims that randomized network coding is too complex to be used with sensor networks, using this as part of the motivation to design a new coding scheme [Kamra et al. 2006]. We assume that this conclusion was due to the expensive Gaussian elimination mentioned above. Our conclusion was different: even though we do agree that Gaussian elimination is not suitable for sensor networks, we found that, in practice, it is unnecessary—the simple, computationally inexpensive version of network coding that we use introduces negligible processing overhead.

Coding Field Size. We chose a field of size 16 for our coding operations because, based on our experiments, this field size makes it unlikely that a node receives or overhears a packet that does not contribute new information (is not linearly independent from the packets that the node has already received and overheard). Given this field size, in our TelosB testbed, performing coding operations consumes less than 5% of the energy consumed transmitting and receiving/overhearing packets; this addresses the common concern that the coding operations introduce energy consumption beyond typical mote capabilities.

Each uncodable packet contains a 2-byte header that specifies the source of the contained message. Each codable packet contains a coding vector that specifies the contained linear combination. Since we are using a field of size 16, each coding vector is a sequence of 4-bit symbols; if the coding vector contains an odd number of symbols, it is padded with zeros (we will discuss other options in Section 5.2).

4. EVALUATION

In this section, we present our evaluation of SenseCode. We first describe our evaluation setup (Section 4.1), then our results: SenseCode's performance (Section 4.2), how much overhearing is needed to achieve this improvement (Section 4.2), and whether adaptive routing is needed to achieve this performance (Section 4.3).

4.1. Setup

Simulation Environment. We simulate a network of $N = 36$ nodes, placed on a 6×6 grid, using the TOSSIM simulator [Levis et al. 2003]. One of the nodes in the corner of the topology is the sink, while all other nodes act as both sources and forwarding nodes. We set the physical distance between the nodes to 20 meters, according to their specifications. We use the default radio model (parameters stated in Table II) included in TOSSIM. Each experiment simulates the network for 10^4 seconds.

Synchronization. SenseCode works in rounds; we call the duration of each round the *sampling period*. At the beginning of each round, the sink floods the network with a request for messages, using the dissemination protocol included in TinyOS. After receiving the sink's request, each node randomly chooses a point in time during the first half of the round to create a new message for the sink (and inject new packets in the network). We made this choice, because, based on our experiments, the network becomes seriously congested when all the nodes inject packets in a synchronized manner.

Adaptive vs. Static Routing. We consider two types of routing: In *adaptive* routing, the nodes maintain a routing tree that adapts to network conditions; we always use the CTP adaptation scheme, because, given a certain energy budget, we found it to be the most reliable of the publicly available routing protocols for sensor networks. In *static* routing, the nodes use the same tree for routing throughout each experiment; to select the tree, we let CTP run in absence of losses for a few rounds, then "freeze" the current tree.

Alternative Protocols. We compare SenseCode to the following alternative collection protocols:

(1) *CTP*: The point of the comparison is to determine whether adding SenseCode on top of an already robust routing protocol decreases E2E packet error rate.

(2) *Opportunistic CTP*: This is a modified version of CTP that implements "opportunistic routing" [Biswas and Morris 2005]. The point of the comparison is to determine whether using redundancy as opposed to more flexible routing decreases E2E packet error rate. The gist of the approach is the following: When a node wants to send a packet, it picks 4 neighbors as "candidate forwarders" and specifies their identities and priorities in the packet's headers. The list never includes neighbors that are marked by CTP as congested or have a higher expected-transmission-count (ETX) number than the sending node; if no neighbor satisfies these constraints, the node sends the packet to its parent, as in plain CTP. Each candidate forwarder that hears the packet waits for some timeout, then sends an ACK; the timeout is proportional to the forwarder's priority. When a candidate forwarder F has not sent its ACK yet, but hears another forwarder's ACK, F reports the higher priority forwarder in its own ACK and discards

Table II. TOSSIM Parameters

Parameter	Value
Radio transmit power	0 dBm
Path-loss model	Log-distance
Path loss at 1 m	55 dB
Path-loss exponent	2.2
Shadowing σ	3.0 dB
SNR to PRR curve	Default [Lee et al. 2007]
Data rate	256 kbps (Default)
Noise model	CPM [Lee et al. 2007]
Noise trace	Meyer Library (first 2000 samples) [Lee et al. 2007]

the packet. As a result, each packet is forwarded by the best candidate forwarder that successfully heard it.

(3) *CTP with redundancy*: This is a modified version of CTP where nodes transmit multiple copies of each message. The point of the comparison is to determine whether using network coding to introduce redundancy is better than simply transmitting multiple copies of each message. For instance, suppose that we want to evaluate SenseCode with redundancy $R = 2$ (i.e., each node that has a new message to communicate injects two new packets in the network). We modify CTP such that each node that has a new message to communicate also injects two copies of the message in the network. We space out the transmissions of the two copies uniformly across the round, in order to give as much time as possible to CTP to adapt its tree to the current network conditions. Note that, in this modified version of CTP, two copies of the same message may take different routes—if the tree has changed between the two transmissions.

(4) *Fully-coded SenseCode*: We compare our version of SenseCode (which uses systematic coding) to an alternative version, in which all transmitted packets are coded. The point of the comparison is to validate that systematic and full coding result in similar E2E packet error rate, as expected based on our simplified modeling in Section 2.

In all protocols, we enable overhearing at the sink, i.e., the sink processes not only the packets it explicitly receives, but also all the packets it overhears. This trivial optimization increases the performance of all protocols without affecting energy-efficiency (since the sink does not typically have energy constraints).

Metrics. We compare the collection protocols based on the two following metrics:

(1) *End-to-end (E2E) packet error rate* quantifies the amount of useful information that reaches the sink. We compute the E2E packet error rate of a collection protocol during a particular round as the number of distinct messages that are successfully decoded by the sink during that round, divided by the total number of distinct messages sent to the sink during the round. For instance, E2E packet error rate 0 means that all messages sent to the sink were successfully decoded.

(2) *Transmit (TX) energy consumption* quantifies the amount of energy spent in transmitting packets. We compute the TX energy consumption of a collection protocol during a particular round indirectly, as the total number of data bytes transmitted by the nodes during that round.

Every data point presented in this section is an average obtained over multiple experiments (from 20 to 50, depending on the variability of the measured value). Each graph includes error bars that show the 95% confidence intervals for the corresponding data, computed using the bootstrap method[Le Boudec 2010] (which works for non-normally distributed estimators, as it is the case in our simulations).

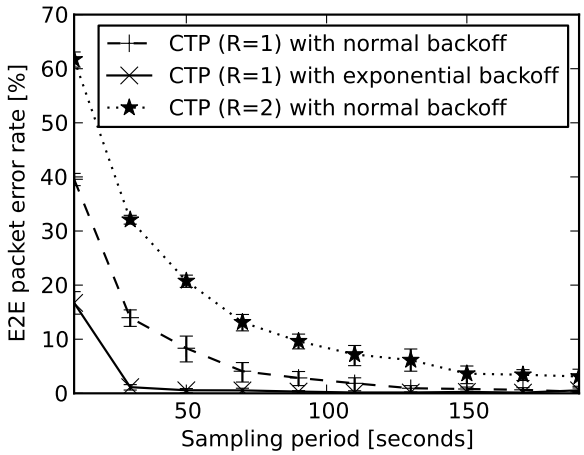


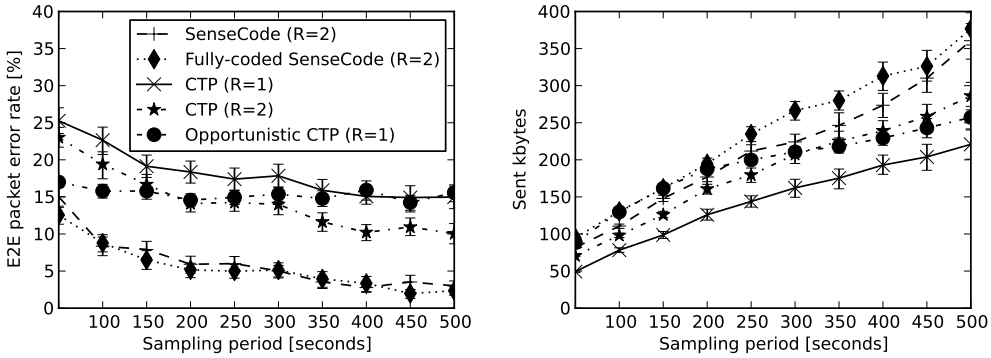
Fig. 5. CTP's E2E packet error rate as a function of the sampling period. For small sampling periods, normal backoff results in high packet loss due to congestion; adding redundancy does not help, because it increases congestion. For a sampling period of 25 seconds or more, exponential backoff results in near 0 packet loss.

Implementation Details. We implemented all protocols as TinyOS modules; our source code for SenseCode is available at http://arni.epfl.ch/software. We built on the TinyOS sources in the 2.x branch, as found in the TinyOS repository on February 26, 2010; the CTP version included in these sources is CTP Noe. We configured CTP to use the 8-bit link-quality estimator included with TinyOS. We set the maximum number of MAC-layer retransmissions to the default value of 30. When compiled for MSP430 (the microcontroller on TelosB motes), SenseCode consumes approximately 4 KB of ROM more than CTP. Assuming a maximum transfer unit (MTU) of 40 bytes, SenseCode's memory overhead compared to CTP is 1.6 KB for sensor nodes (which do not need to decode any linear combinations) and 4.3 KB for the sink (which does perform decoding).

Congestion Avoidance. All protocols use exponential backoff for congestion avoidance. We made this choice for the following reason: The CTP implementation distributed with TinyOS uses a fixed backoff for congestion avoidance. When the sampling period is small, a fixed backoff results in high E2E packet error rate due to network congestion (neighboring nodes contend for bandwidth, as they all try to retransmit their packets during a small period of time). In such a situation, adding redundancy not only does not help, but is detrimental to E2E packet error rate, as it increases network load, which in turn leads to more congestion. Replacing the congestion avoidance mechanism with exponential backoff reduces E2E packet error rate significantly by de-synchronizing retransmissions. Figure 5 shows these effects: for a sampling period of 25 seconds, CTP with fixed backoff results in 15% loss; adding redundancy $R = 2$ more than doubles the loss; CTP with an exponential backoff (and no redundancy) has close to 0 loss.

4.2. Performance of Adaptive SenseCode

We now evaluate the performance of our collection protocols under different network conditions. Since the point of this work is to achieve low E2E packet error rate in the face of packet loss, we consider scenarios where nodes can be "occluded," i.e., temporarily disconnected from their neighbors, resulting in packet loss.



(a) E2E packet error rate. (b) TX energy consumption.

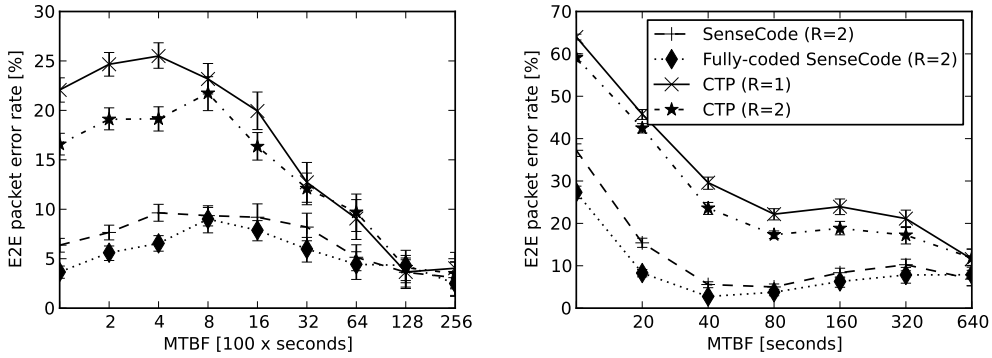
Fig. 6. Performance of different collection protocols as a function of the sampling period. For instance, SenseCode's E2E packet error rate at a sampling period of 350 seconds is below 5% (left), while its energy consumption is 10% more than CTP's with the same level of redundancy (right).

Node Occlusion. We model node occlusion using a two-state Markov Chain. Each node can be either in a "connected" or in a "disconnected" (outage) state. The parameters of the model are the *mean time between failures* (MTBF), i.e., the average time that elapses from the moment a node enters the connected state until it exits the state, and the *mean time to repair* (MTTR), i.e., the average time that elapses from the moment the node enters the disconnected state until it exits the state. Every second and for every node a biased coin decides whether the node should change state or not. If a disconnected node happens to have a packet in its transmit queue, it keeps trying to send it to its parent and failing; if the maximum number of retransmissions is exceeded before the node exits the disconnected state, the packet is discarded.

Error Rate and Energy Consumption. Figure 6(a) shows the E2E packet error rate of each protocol as a function of the sampling period (the duration of each round), while Figure 6(b) shows the corresponding TX energy consumption. In these experiments, we set MTBF to 1000 seconds and MTTR to 100 seconds (a setup that results in 10% of the nodes being occluded at any given time).

First, we observe that redundancy and spatial coding improve E2E packet error rate, whereas redundancy alone does not: SenseCode's E2E packet error rate ranges from 3% to 15%, while CTP's ranges from 15% to 25% (without redundancy) and from 10% to 22% (with redundancy). This is because of the following reason: In CTP with redundancy, each redundant copy of a message follows the path dictated by the routing tree; if this path is disrupted due to node occlusions, the tree may not have the time to adapt within the current round, causing all redundant copies to follow the same (disrupted) path. In contrast, SenseCode spreads pieces of each message across the network, making it more likely for enough pieces to reach the sink such that the latter can decode the message.

Second, we observe that spatial coding improves E2E packet error rate, even if the tree adapts to network conditions as fast as possible: Opportunistic CTP's E2E packet error rate ranges from 15% to 17%. This is due to two reasons. When a node enters into a disconnected state for a sufficiently long period of time, all packets queued for transmission on that node are lost, even if we use opportunistic routing. Moreover, since opportunistic CTP relies on the currently available ETX values at the node to select the forwarders list of each packet, when the tree is disrupted, packets can be sent to the wrong nodes.



(a) MTTR is $\frac{1}{10}$ MTBF. (b) MTTR is fixed.

Fig. 7. E2E packet error rate as a function of network conditions. The higher the frequency with which the network-graph changes and the higher the number of nodes that are disconnected at any point in time, the bigger the benefit that SenseCode brings over CTP with the same level of redundancy.

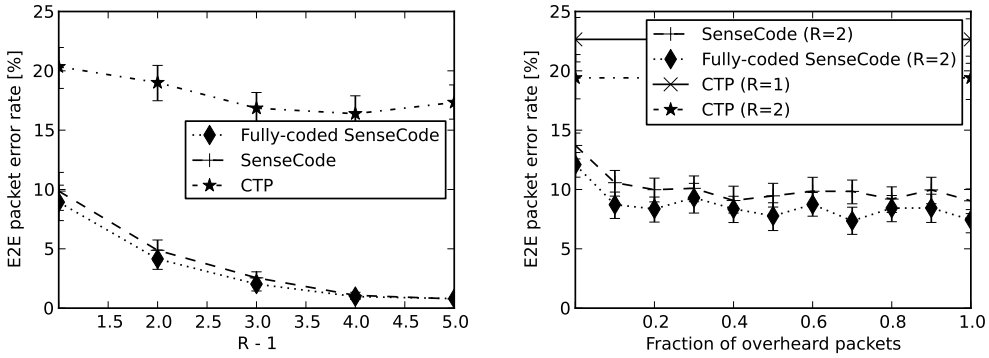
Third, we observe that systematic spatial coding is sufficient, i.e., full spatial coding is not needed: SenseCode and fully-coded SenseCode achieve roughly the same E2E packet error rate. This is precisely what we expected based on our simple modeling in Section 2.

Finally, we observe that, as expected, introducing redundancy increases the transmit energy consumption of all protocols. CTP with redundancy spends less energy than both versions of SenseCode (because it avoids the overhead of coding vectors), while SenseCode spends less energy than fully-coded SenseCode (because it uses fewer coded packets and avoids part of the coding-vector overhead). In our experiments, we use a payload of 77 bytes, which results in a byte overhead of 23% per coded packet; this byte overhead translates into a transmit-energy overhead of 8% for SenseCode and 17% for fully-coded SenseCode, compared to CTP with redundancy $R = 2$.

Error Rate vs. Network Conditions. Figure 7(a) shows the E2E packet error rate of each protocol as a function of MTBF, when MTTR changes proportionally to MTBF; with this setup, smaller values of MTBF correspond to higher-frequency changes in the network-connectivity graph. Figure 7(b) also shows E2E packet error rate as a function of MTBF, but when MTTR is fixed to 10 seconds; with this setup, smaller values of MTBF correspond to a larger fraction of the nodes being disconnected at any point in time. In these experiments, we set the sampling period to 100 seconds, which results in no congestion (no packet loss in the absence of node occlusions).

We observe that the faster the network-graph changes, the bigger the gap between SenseCode's and CTP's E2E packet error rate (because CTP has no time to adapt its routing tree to the network connectivity graph). When MTBF is 400 seconds and MTTR is 40 seconds (a connected node is expected to disconnect in 400 seconds and stay disconnected for 40 seconds), SenseCode's E2E packet error rate is 10%, while CTP's is 25%. When MTBF is 40 seconds, SenseCode's E2E packet error rate is 7%, while CTP's is 30%. On the other hand, as we move to the right end of the x-axis (in both graphs), the performance of all protocols converges to the same value, which means that CTP alone is enough.

Higher Redundancy Factors. Figure 8(a) shows the E2E packet error rate of CTP with redundancy, SenseCode, and fully-coded SenseCode, as a function of the redundancy factor. In these experiments, we set the sampling period to 100 seconds, the



(a) The x-axis corresponds to the redundancy factor. (b) The x-axis corresponds to the fraction of overheard packets that each node stores.

Fig. 8. E2E packet error rate as a function of redundancy and overhearing. SenseCode's performance improves faster with the redundancy factor than CTP's (left). SenseCode achieves its maximum performance when nodes store 10% of the packets they overhear (right).

MTBF to 1000 seconds, and the MTTR to 100 seconds (a setup that results in no congestion in the absence of node occlusions and 10% of the nodes being occluded at any point in time).

We observe that, SenseCode's E2E packet error rate decreases significantly more sharply with the redundancy factor than CTP's E2E packet error rate, which is what we expected based on our simple model from Section 2. We should clarify that we are not advocating to use a higher redundancy factor than $R = 2$: going from $R = 2$ to $R = 3$ reduces packet error rate from 10% to 5%, while it increases energy consumption by 50%. Our point is that, if we need redundancy, then spatial network coding is a better way of introducing it than simply sending out redundant copies of each message.

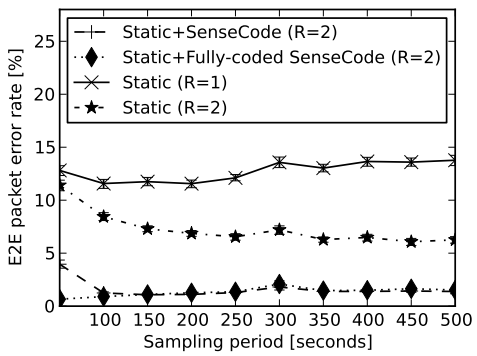
Amount of Overhearing. Figure 8(b) shows SenseCode's E2E packet error rate as a function of the fraction of overheard packets that each node stores. In these experiments, the sampling rate is set to 100 seconds, the MTBF to 1000 seconds, and the MTTR to 100 seconds.

We observe that storing only 10% of the overheard packets yields the same E2E packet error rate with constant overhearing. This is because each packet is overheard by multiple nodes, hence, even if few of them store the packet, there is a good chance that its contents will spread through the network via more than one path. Note that SenseCode achieves lower E2E packet error rate than CTP even with 0 overhearing. This is because, when the network-connectivity graph changes frequently, a node is likely to switch parents in the middle of sending out a packet; in this case, the packet ends up being received by more than one parent. Given that each node sends out linear combinations of all the packets it receives, some information ends up reaching the sink through multiple paths—so, we do introduce redundancy, even if the nodes do not explicitly send out multiple copies of each message.

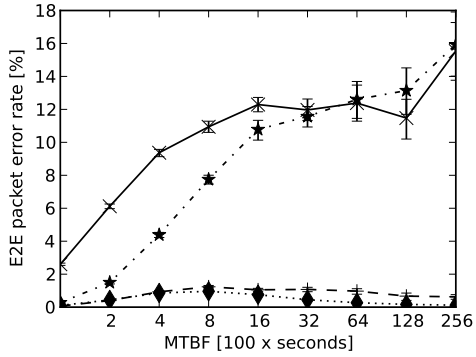
4.3. Adaptive vs. Static Routing

We have seen how SenseCode performs when combined with adaptive routing (as performed by CTP); we now examine how it performs on top of static routing.

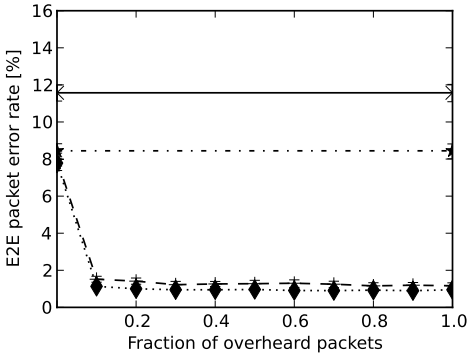
First, we look at E2E packet error rate as a function of the sampling period, setting MTBF to 1000 seconds and the MTTR to 100 seconds; we observe that SenseCode achieves lower E2E packet error rate on top of static routing (Figure 9(a)) than on top



(a) E2E packet-error rate vs. Sampling period



(b) E2E packet-error rate vs. MTBF. MTTR is $\frac{1}{10}$ MTBF.



(c) E2E packet-error rare vs. Overhearing

Fig. 9. E2E packet-error rate of different collection protocols when we use static routing.

of adaptive routing (Figure 6(a)). This is because, when the network-connectivity graph changes too fast for the tree to adapt successfully, trying to adapt the tree can make things worse: packets that have already been overheard by multiple nodes (hence need not be retransmitted) are re-directed through the few surviving paths of the tree causing congestion; in such cases, it is better to rely on redundancy and spatial coding for recovery.

Second, we look at E2E packet error rate as a function of network conditions, setting the sampling period to 100 seconds; we observe that, the E2E packet error rate achieved by static SenseCode *decreases* as network conditions change faster (Figure 9(b)); in contrast, the E2E packet error rate achieved by adaptive SenseCode *increases* as network conditions change faster (see Figure 7(a)). This is because of the following reason: when network conditions change fast, nodes fail frequently, but they also recover frequently; the faster nodes recover, the more likely they are to have time to transmit their packets during the current round. Adaptive SenseCode does not benefit from fast node recovery as much, because it tries to adapt the tree to the network-connectivity graph, which ends up making things worse.

Third, we look at E2E packet error rate as a function of the fraction of overheard packets that each node stores (Figure 9(c)). Like adaptive SenseCode, static SenseCode reaches its maximum performance as soon as each node stores 10% of the overheard packets. Unlike adaptive SenseCode, static SenseCode performs the same with

CTP when no overhearing occurs; this makes sense, because static SenseCode does not adapt the tree to the current connectivity graph, hence, when no overhearing occurs, each message travels though a single path toward the sink.

5. DISCUSSION

5.1. Multipath

We now compare our approach to traditional multipath communication, the most popular existing approach for balancing end-to-end packet error rate and energy efficiency.

In a k-multipath protocol, the sensor nodes build and maintain k trees, and sources inject each packet on each of the trees; the k paths from a node to the sink are selected such that at least one path is likely to survive spatially correlated failures [Ganesan et al. 2001; De et al. 2003; Pottie and Kaiser 2000; Li and Li 2006].

One disadvantage of traditional multipath is the need for additional control traffic, as each node needs to monitor its connection to k different parents. There is evidence that maintaining more than $k = 2$ paths per node (or more than two incoming and outgoing links per node) requires control traffic that outweighs the benefits of multipath, as it can consume a significant portion of network resources (energy, bandwidth, processing time) [Ganesan et al. 2001]. In contrast, SenseCode leverages the fact that multiple neighbors are likely to overhear each transmission, without forcing the transmitter to explicitly register these neighbors as parents. Hence, in some sense, SenseCode *is* a multipath protocol, albeit one that does not require maintaining multiple trees, but opportunistically "creates" them on the fly.

We should also note that building multiple trees such that at least one is likely to survive spatially correlated failures is not straightforward. For instance, selecting two edge-disjoint minimum-cost paths between a pair of vertices's (nodes) in a graph (network) is an NP-hard problem, even in a centralized setting (where a single entity knows the quality of all links).

Finally, traditional multipath protocols rely on non-coded communication, which, from a coding theory point of view, is a suboptimal use of the introduced redundancy [MacWilliams and Sloane 1983]. For instance, we expect a 2-multipath protocol to consume the same amount of energy as SenseCode with redundancy factor 2, yet achieve higher end-to-end packet error rate (see also Section 2).

5.2. Compressed Coding Vectors

A valid concern regarding deploying network coding in sensor networks is the overhead of the coding vectors. Such a vector needs to be appended to each codable packet, to keep track of the linear combination of the original packets the coded packet contains [Chou et al. 2003]. A different approach that has recently been proposed is subspace coding [Koetter and Kschischang 2008; Silva and Kschischang 2007]. Unfortunately, this approach proves also difficult to deploy in sensor networks [Jafari Siavoshani et al. 2009].

In parallel and complementary to this work, we proposed a novel design with much lower overhead based on the use of *compressed* coding vectors [Jafari Siavoshani et al. 2009]. Our observation was that, both coding vectors and subspace coding are designed under the strong assumption that *potentially all source packets get linearly combined in the network*. However, in practice, in a sensor network, packets from sources that are significantly separated topologically may never get mixed together. Given that allowing all source packets to get combined in a single coded packet might be a too strong and unnecessary requirement for many practical situations, we can relax it, and require that each coded packets contains a linear combination of at most m out the n source packets.

For this problem, we developed a design that reduces the length of the coding vectors from n to $\mathcal{O}(m \log n)$, where n is the number of source packets injected in the network [Jafari Siavoshani et al. 2009]. Our design is *transparent* to and has *no extra processing* at intermediate nodes, but comes at the cost of some additional processing at the sink. However, this additional processing can be performed with standard hardware implemented algorithms.

6. RELATED WORK

Network coding was introduced in [Ahlswede et al. 2000; Li et al. 2003], see also for tutorial articles and monographs [Fragouli et al. 2006; Ho and Lun 2008], and has been successfully applied in wireless ad-hoc and mesh networks, see for example [Wu et al. 2005]–[Katti et al. 2008]. Our work builds on this foundation, but is addressed to a fundamentally different problem, due to the special constraints and requirements of the sensor network environment. For example, the focus in [Chachulski et al. 2007] is in bandwidth efficiency through per-flow network coding; we are instead in optimizing for energy and end-to-end packet error rate coding for inverse multicast traffic, thus coding across several information sources.

Protocols for network coding over sensor networks have been proposed for example in [Adjih et al. 2007; Petrović et al. 2006; Toledo and Wang 2006; Guo et al. 2006], but the literature work is theoretical and through simulations. In contrast, the new design we propose is driven by implementation related issues, such as easy deployment on top of existing routing protocols, and as far as we know offers the first practical implementation.

Coding for sensor networks has also been proposed in [Dimakis et al. 2005; Kamra et al. 2006] from a viewpoint of distributed storage. Between these, the work closer to ours is [Kamra et al. 2006], which proposes a form of spatial coding to improve partial recovery of data in emergency situations. The emphasis in [Kamra et al. 2006] is in the cautious encoding of information packets to ensure that even a small number of the resulting coded packets allow to decode some information. Our work differs in several ways. First, we are interested in stable network operation, where energy consumption is of importance, unlike [Kamra et al. 2006] where energy is of no consideration and an arbitrary number of packets might be sent from each surviving node. Second, we leverage opportunistic broadcasting to implement our information mixing, as opposed to explicitly exchanging messages with neighboring nodes as in [Kamra et al. 2006] (an extension of Growth codes that takes into account broadcast has been proposed in [Karande et al. 2008]). Third, our experimental results compare against state-of-the-art adaptation protocols such as CTP.

End-to-end coding using algebraic codes has also been proposed for sensor networks [Sartipi and Fekri 2004; Wood and Stankovic 2008], however, unlike network coding this approach does not protect from spatially correlated failures. Multipath diversity, that was developed to address spatially correlated failures, for sensor networks is a well studied and active research field, see for example [Pottie and Kaiser 2005; Ganesan et al. 2001; De et al. 2003]. The use of coding we introduce allows to take advantage of broadcasting, with reduced control information, and allows to achieve different points in the end-to-end packet error rate/energy-efficiency trade-off.

Synopsis diffusion [Nath et al. 2008] offers a framework that abstracts properties of duplication and order insensitive protocols in sensor networks. Network coding, although it does have similar properties, cannot be directly cast in the synopsis framework for several reasons. More prominently, (i) unlike synopsis, in network coding information extracted from a single node may be scattered throughout the network and not fully contained to any single packet, and (ii) network coding offers probabilistic as opposed to hard guarantees of duplication insensitivity.

7. CONCLUSION

We presented SenseCode, a collection protocol for sensor networks that achieves high reliability (in our experiments, over 90%) in the face of highly dynamic network conditions; it achieves this through a combination of redundancy and network coding. By comparing SenseCode to different variations of the Collection Tree Protocol (CTP), we drew the following conclusions: First, CTP has excellent performance under normal network conditions (no node occlusions), i.e., it achieves more than 90% reliability. However, under highly dynamic network conditions (at least 10% of the nodes are disconnected at any point in time), CTP's reliability drops below 75% (even below 60%, in certain occasions). In such scenarios, combining redundancy with network coding improves reliability (over 90%), whereas redundancy alone or opportunistic routing do not.

REFERENCES

ADJIH, C., CHO, S. Y., AND JACQUET, P. 2007. Near optimal broadcast with network coding in large sensor networks. *CoRR abs/0708.0975*.

AHLSWEDE, R., CAI, N., LI, S.-Y. R., AND YEUNG, R. W. 2000. Network information flow. *IEEE Transactions on Information Theory 46,* 4, 1204–1216.

BISWAS, S. AND MORRIS, R. 2005. ExOR: opportunistic multi-hop routing for wireless networks. *SIGCOMM Computer Communication Review 35,* 4, 133–144.

CHACHULSKI, S., JENNINGS, M., KATTI, S., AND KATABI, D. 2007. Trading structure for randomness in wireless opportunistic routing. *SIGCOMM Computer Communication Review 37,* 4, 169–180.

CHOU, P. A., WU, Y., AND JAIN, K. 2003. Practical network coding. In *Proceedings of the 42th Allerton Conference on Communication, Control, and Computing*.

DE, S., QIAO, C., AND WU, H. 2003. Meshed multipath routing with selective forwarding: an efficient strategy in sensor networks. *Computer Networks 43,* 4, 481–497.

DIMAKIS, A. G., PRABHAKARAN, V., AND RAMCHANDRAN, K. 2005. Ubiquitous access to distributed data in large-scale sensor networks through decentralized erasure codes. In *Proceedings of the 4th international symposium on information processing in sensor networks*. IEEE, IEEE, New York, NY, USA, 111–117.

FRAGOULI, C., WIDMER, J., AND LE BOUDEC, J.-Y. 2006. Network coding: An instant primer. *ACM SIGCOMM Computer Communication Review 36,* 1, 63–68.

GANESAN, D., GOVINDAN, R., SHENKER, S., AND ESTRIN, D. 2001. Highly-resilient, energy-efficient multipath routing for wireless sensor networks. *ACM SIGMOBILE Mobile Computing and Communications Review 5,* 4, 11–25.

GNAWALI, O., FONSECA, R., JAMIESON, K., MOSS, D., AND LEVIS, P. 2009. Collection tree protocol. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*. ACM, ACM, New York, NY, USA, 1–14.

GUO, Z., XIE, P., CUI, J.-H., AND WANG, B. 2006. On applying network coding to underwater sensor networks. In *Proceedings of the 1st ACM international workshop on Underwater networks*. ACM, ACM, New York, NY, USA, 109–112.

HO, T. AND LUN, D. S. 2008. *Network coding: An introduction*. Cambridge University Press, Cambridge, U.K.

HORN, R. A. AND JOHNSON, C. R. 1990. *Matrix Analysis*. Cambridge University Press, Cambridge, U.K.

JAFARI SIAVOSHANI, M., KELLER, L., FRAGOULI, C., AND ARGYRAKI, K. 2009. Compressed network coding vectors. In *Proceedings of the 2009 IEEE International Symposium on Information Theory*. IEEE, IEEE, New York, NY, USA, 109–113.

KAMRA, A., MISRA, V., FELDMAN, J., AND RUBENSTEIN, D. 2006. Growth codes: maximizing sensor network data persistence. *ACM SIGCOMM Computer Communication Review 36,* 4, 255–266.

KARANDE, S., MISRA, K., AND RADHA, H. 2008. Natural growth codes: Partial recovery under random network coding. In *Proceedings of the 42nd Annual Conference on Information Sciences and Systems*. IEEE, IEEE, New York, NY, USA, 540–544.

KATTI, S., KATABI, D., BALAKRISHNAN, H., AND MEDARD, M. 2008. Symbol-level network coding for wireless mesh networks. *SIGCOMM Computer Communication Review 38,* 4, 401–412.

KOETTER, R. AND KSCHISCHANG, F. R. 2008. Coding for errors and erasures in random network coding. *IEEE Transactions on Information Theory 54,* 8, 3579–3591.

LE BOUDEC, J.-Y. 2010. *Performance Evaluation of Computer and Communication Systems*. EPFL Press, Lausanne, Switzerland.

LEE, H. J., CERPA, A., AND LEVIS, P. 2007. Improving wireless simulation through noise modeling. In *Proceedings of the 6th International Symposium on Information Processing in Sensor Networks*. IEEE, IEEE, New York, NY, USA, 21–30.

LEVIS, P., LEE, N., WELSH, M., AND CULLER, D. 2003. Tossim: Accurate and scalable simulation of entire tinyos applications. In *Proceedings of the 1st international conference on Embedded networked sensor systems*. ACM, ACM, New York, NY, USA, 126–137.

LEVIS, P., MADDEN, S., POLASTRE, J., SZEWCZYK, R., WHITEHOUSE, K., WOO, A., GAY, D., HILL, J., WELSH, M., BREWER, E., AND CULLER, D. 2005. TinyOS: an operating system for sensor networks. In *Ambient Intelligence*, W. Werner, J. M. Rabaey, and E. Aarts, Eds. Vol. 35. Springer, Berlin, Heidelberg, 115–148.

LI, S.-Y. R., YEUNG, R. W., AND CAI, N. 2003. Linear network coding. *IEEE Transactions on Information Theory 49,* 2, 371–381.

LI, Z. AND LI, B. 2006. Improving throughput in multihop wireless networks. *IEEE Transactions on Vehicular Technology 55,* 3, 762–773.

MACWILLIAMS, F. J. AND SLOANE, N. J. A. 1983. *The theory of error correcting codes*. North Holland, Amsterdam, Netherlands.

NATH, S., GIBBONS, P. B., SESHAN, S., AND ANDERSON, Z. 2008. Synopsis diffusion for robust aggregation in sensor networks. *ACM Transactions on Sensor Networks 4,* 2, 7:1–7:40.

PETROVIĆ, D., RAMCHANDRAN, K., AND RABAEY, J. 2006. Overcoming untuned radios in wireless sensor networks with network coding. *IEEE/ACM Transactions on Networking (TON) 14,* SI, 2649–2657.

POTTIE, G. AND KAISER, W. 2000. Wireless integrated network sensors. *Communications of the ACM 43,* 5, 51–58.

POTTIE, G. AND KAISER, W. 2005. *Principles of embedded networked systems*. Cambridge University Press, Cambridge, U.K.

SARTIPI, M. AND FEKRI, F. 2004. Source and channel coding in wireless sensor networks using ldpc codes. In *Proceedings of the 1st Annual IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks*. IEEE, IEEE, New York, NY, USA, 309–316.

SILVA, D. AND KSCHISCHANG, F. R. 2007. Using rank-metric codes for error correction in random network coding. In *Proceedings of the 2007 IEEE International Symposium on Information Theory*. IEEE, IEEE, New York, NY, USA, 796–800.

TEXAS INSTRUMENTS. 2007. CC2420 RF Transceiver Datasheet. `http://www.ti.com/lit/gpn/cc2420`.

TOLEDO, A. L. AND WANG, X. 2006. Efficient multipath in sensor networks using diffusion and network coding. In *Proceedings of the 40th Annual Conference on Conference on Information Sciences and Systems*. IEEE, IEEE, New York, NY, USA, 87–92.

WOOD, A. AND STANKOVIC, J. 2008. Rateless erasure codes for bulk transfer in asymmetric wireless sensor networks. In *Proceedings of the 6th ACM conference on Embedded network sensor systems*. ACM, ACM, New York, NY, USA, 449–450.

WU, Y., CHOU, P. A., AND KUNG, S. Y. 2005. Minimum-energy multicast in mobile ad-hoc networks using network coding. *IEEE Transactions on Communications 53,* 11, 1906–1918.