SenseCode: Network Coding for Reliable Sensor Networks

Lorenzo Keller, Emre Atsan, Katerina Argyraki, Christina Fragouli

Ecole Polytechnique Federale Lausanne (EPFL)
Lausanne, Switzerland

Abstract

Designing a communication protocol for sensor networks often involves obtaining the "right" trade-off between energy efficiency and reliability. 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 protocol to employ network coding. SenseCode provides a way to gracefully introduce a configurable amount of redundant information in the network, thereby increasing reliability in the face of packet loss. We compare SenseCode to the best (to our knowledge) existing alternative and show that it improves reliability (in our experiments typically by $15\% - 20\%$), while consuming a comparable amount of network resources. We have implemented SenseCode as a TinyOs module, and evaluate it through extensive TOSSIM simulations.

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 healthcare 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 reliability (i.e., the capability of the network to operate in the face of packet loss). In sensor networks, these are typically competing goals: increasing reliability requires introducing redundant information in the network (for example, transmitting each piece of information 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.

In this paper, we show that network coding provides an alternative, elegant way to balance energy efficiency and reliability. 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 implementation of a collection protocol for sensor networks that relies on network coding.

Network-coding ideas and techniques have already been applied to wireless mesh networks [5]–[8]. A natural question is, how are sensor networks different—why can't we 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 increased reliability 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 reliability. It uses simple, decentralized algorithms that do not exceed the sensors' modest processing and memory capabilities.
2. We have implemented SenseCode as a TinyOs [26] module, and evaluate it through TOSSIM [25] simulations. We compare its performance to the collection tree protocol (CTP) [27], the best, to our knowledge, existing alternative, and show that it achieves higher reliability(typically $15 - 20\%$), while consuming a comparable amount of network resources.

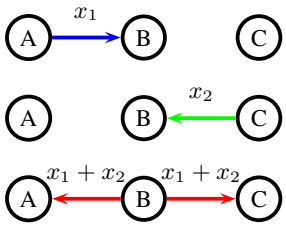

Figure 1. A pattern that does *not* occur in sensor networks that rely on "many-to-one" communication: Nodes A and C exchange packets x_1 and x_2 via relay B. B receives x_1 and x_2 and broadcasts their binary xor, $x_1 + x_2$. Node A uses its knowledge of x_1 to retrieve x_2, and node C uses x_2 to retrieve x_1.

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.

2. Setup

2.1 Problem Statement and Background

We consider a sensor network that operates in rounds: in each round, each sensor 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 sensors' messages to reach the sink.

Our goal is to design a collection protocol that achieves a desired level of reliability while consuming as little energy as possible. *Reliability* refers to the protocol's ability to communicate messages to the sink in the face of packet losses - these may be due to channel and node failures, or simply congestion - in general events that alter the connectivity graph of the network. We define reliability as the average fraction of messages that are successfully communicated to the sink during each round; for instance, reliability 0.5 means that, on average, half of the sensors successfully communicate their message to the sink during each 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, the primary source of energy consumption is the radio, hence, we target a collection protocol that achieves a certain level of reliability while minimizing the time of radio utilization.

Our basis for comparison will be the Collection Tree Protocol (CTP) [27], which, to the best of our knowledge, is the best existing collection protocol in terms of reliability and energy efficiency. In CTP, the network nodes build and maintain a tree rooted at the sink; each sensor 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 sensor to the sink through the path that results in the minimum number of packet transmissions. CTP achieves reliability 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.

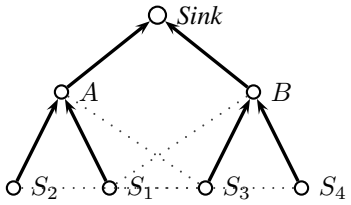

Figure 2. A sensor network, where nodes form a tree rooted at the sink. Each source S_i, $i = 1 \ldots 4$, has a message x_i to communicate to the sink. Each node sends packets only to its parent, however, each node can overhear the transmissions not only of its children, but of multiple neighbors. S_1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

2.2 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 packet 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 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 fails*[1]*, before forwarding them to the sink. 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 reliability 0.5 (only half the messages reach the sink).*

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

[1] i.e., either the node stops operating or the channels that connect it to its neighbors deteriorate

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 build and 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 reliability in the face of packet loss.

EXAMPLE 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 1. As in Example 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 reliability 1, 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 increased reliability 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 reliability/energy-efficiency tradeoff, where, to communicate N messages to the sink, we introduce $R \times N$ packets in the network, and, in exchange, we achieve reliability 1 as long as *any* set of N packets reach the sink.

2.4 Why Network 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 higher reliability in a more energy-efficient manner.

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	$x_1, x_1 + x_2 + x_3$	$x_3, x_4, x_1, x_1 + x_2 + x_3$

Table 1. The contents of the packets that were overheard and sent by each node from Figure 2 during the scenario described in Example 1. The sink can decode the linear combinations through the use of coding vectors.

To illustrate, we consider the simplified model[2] 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 reliability 1, 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 reliability $1 - \epsilon$ (of the N messages—and packets—sent to the sink, the latter is expected to receive $(1 - \epsilon)N$).

Now suppose that, to increase reliability, the source is willing to introduce redundant information by a factor of 2—i.e., if it needs x bits to represent the N messages, it is willing to send $2x$ bits to the sink. Coding theory tells us that this amount of redundancy is, theoretically, sufficient for achieving reliability 1, 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. Hence, given a communication protocol with redundancy factor 2, we will consider it "good," as long as it approaches this behavior—i.e., achieves reliability 1 as long as $\epsilon < 0.5$.

We consider three communication protocols with redundancy factor 2.

1. **Non-coded communication**: 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. **Fully-coded communication**: The source produces $2 \times N$ linear combinations of the N messages, such that any

[2] 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.

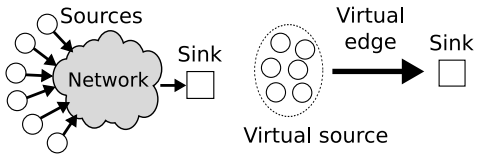

Figure 3. Abstraction of a sensor network as a point-to-point network, where all sources are co-located, and packet loss is captured with a single erasure channel that connects the collocated sources to the sink.

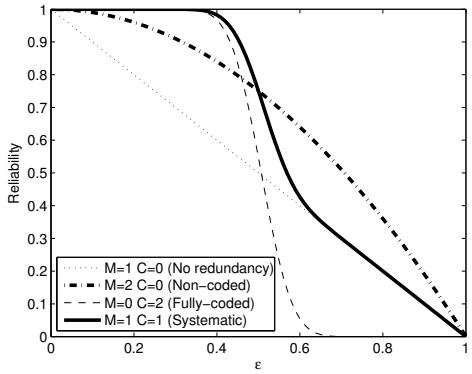

Figure 4. Reliability as a function of packet loss, for different communication protocols. Assume that the virtual source from Figure 3 has $N = 36$ messages to communicate to the sink. We plot reliability as a function of the erasure probability ϵ, when the virtual source sends M uncoded and C coded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

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* 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 communication**: 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 reliability of these three protocols, as a function of the erasure probability ϵ. First, we see that non-coded communication achieves reliability 1 only when $\epsilon = 0$; thereafter, its reliability degrades with ϵ. In contrast, fully-coded communication achieves reliability 1 as long as $\epsilon < 0.45$; thereafter, its reliability sharply drops. 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 reliability. 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 [19]. 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 reliability, yet significantly reduces the coding-vector overhead. This is the approach we adopt for SenseCode.

2.5 A Practical Challenge

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 high reliability, 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 combinations 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 4.3, 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 reliability needs to be boosted.

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

3.1 Design

Topology Construction The sensors use a routing protocol to create and maintain a routing structure that allows nodes to forward packets toward the sink. SenseCode can work on top of any routing protocol. It simply assumes that, at any point in time, each node has a routing table that specifies 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, from now on, that each node has a single parent.

Node Input Each node maintains two packet queues. In the first one, it stores all the messages it has generated itself and all the packets it has received from its children during the current round. In the second queue, it stores information it has overheard from its neighbors during the current round.

Relaying A node sends packets to its parent in two cases:

1. When it has a new message to communicate to the sink. In this case, the node creates a new packet x that includes the new message, marks it as "uncodable," and sends it out; we call these packets that include only a message from a single node *original* packets. Moreover, the node creates $R-1$ packets that are linear combinations of x and the packets in the node's queues, marks them as "codable," and sends them out. R is a configurable parameter, called *redundancy factor*.
2. When it receives a new packet y from a child. If y is an uncodable packet, the receiving node simply forwards it. If y is a codable packet, the receiving node sends out a linear combination of y and information from the node's queues.

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.

Decoding At the end of a round, a total of $R \times N$ packets have been sent to the sink, where N is the number of nodes that had a message to communicate during the round. Each of these received packets is a linear combination of the N messages. Hence, to recover all N messages, the sink needs to receive N linearly independent combinations. The sources append coding vectors to the packets to enable the sink to decode [3]. We discuss the overhead of the coding vectors in Section 5.2.

MAC Layer Collection protocols typically assume 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 also uses such a layer, but with the following modifications: When a node transmits a codable packet, 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 queue (instead of simply retransmitting). This has the following goal: When a node transmits a packet to its parent, even if the parent does not receive it, some other neighbor(s) may overhear it. By transmitting 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 "clean," predictable reliability/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 sensors, 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.

Relaying According to the SenseCode design, each node keeps a queue with information it has overheard during the current round. This queue has room for M packets; every time a new packet is overheard, it is added to each of the M queue slots (using a different coding coefficient for each slot). Based on our experiments, M can be as small as a few packets. We also experimented with an alternative approach, where each node kept only the k most recently overheard packets. It turned out that, in practice, this required a larger queue to achieve the same performance.

In an earlier implementation attempt, we ensured that each packet added to the queue was linearly independent from all the other packets in the queue, i.e., added new information—an optimization frequently used with network coding [3]–[4]. This, however, required performing Gaussian elimination [20] for each newly overheard or received packet; we tested this operation on a TinyNode [28] sensor 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, the probability of overhearing or receiving linearly dependent packets becomes negligible. Hence, in our current implementation, each node simply adds packets to its queue without ensuring they are not redundant.

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 [12]. 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.

Overhearing In sensor networks, receiving a packet typically consumes comparable energy with transmitting one, hence, overhearing comes at an energy cost [29]. Luckily, we found that randomly selecting a small number of packets to overhear is sufficient for achieving the same performance achieved with constant overhearing (Section 4.3). Hence, SenseCode is compatible with energy-efficient sensor platforms, where the sensor is "asleep" most of the time and wakes up only when it hears a packet addressed to it. Limited overhearing can be easily implemented by programming such a platform, such that the sensor wakes up not only when it hears a packet addressed to it, but also, with a small probability, when it hears a packet addressed to a different node.

Coding Operations We use a finite field of size 16 for our coding operations. The reason for this choice is that, from our experiments, we found that this field size makes it unlikely that a node receives or overhears a "useless" packet (i.e., a packet that contains a linear combination that is not linearly independent from the combinations that the node has already overheard).

A concern with network coding is that, the larger the size of the employed field, the larger the processing requirements for the coding operations—hence, the higher the corresponding energy consumption. We measured the power consumed by a TinyNode sensor running SenseCode for performing coding operations and found it to be a small fraction (less than 5%) of the energy consumed transmitting packets.

4. Evaluation

4.1 Methodology

We start by presenting the setup that we used to test SenseCode performance and the alternative protocols that we compared to.

Routing We consider two types of tree topologies:

1. *Adaptive*, where the tree adapts to network conditions as dictated by CTP. We selected 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.
2. *Static*, where the tree topology remains unchanged for the duration of the experiment. To select the static tree, we let CTP run in absence of losses for a few rounds, and then "freeze" the chosen tree.

Alternative Collection Protocols We now describe the three alternative collection protocols to which we compare SenseCode.

1. *CTP*: Our first goal is to determine whether adding SenseCode on top of an already robust routing protocol like CTP improves reliability. To this end, we compare the reliability achieved by SenseCode to that achieved by plain CTP.
2. *CTP with redundancy*: To make a more fair comparison, we modify CTP, such that it also introduces a configurable level of redundancy. For instance, suppose that we run SenseCode with redundancy $R = 2$ (i.e., each node that has a new message to communicate injects two new packets in the network). It may be unfair to compare this to CTP, where each new message is injected in the network only once. Hence, we modify CTP, such that each node that has a new message to communicate also injects two packets in the network, i.e., performs "repetition coding" with redundancy R.
3. *Fully-coded SenseCode*: We also compare our version of SenseCode that employs systematic communication to a different version, in which all transmitted packets are coded. Our goal is to validate that fully-coded and systematic communication result in similar reliability, as expected based on our simplified modeling in Section 2.

In all protocols, we enable overhearing at the sink, i.e., the sink maintains and processes not only the packets it explicitly receives, but also all the packets it overhears. This trivial optimization increases the performance of all protocols and, since the sink typically does not have energy constraints, it not affect the energy efficiency of the network.

Environment We implemented SenseCode and the alternatives as TinyOs modules and tested them on a simulated square grid, created with the TOSSIM simulator [25]. The simulated network consists of $N = 36$ nodes, located on a 6×6 grid. One of the nodes in the corner of the topology is the sink, while all other nodes act as both sources and forwarding nodes. All the nodes are used according to their specifications, i.e., the physical distance between them (roughly 20m) is the maximum distance that enables reliable communication between neighbors.

Sampling Period Each simulation is divided in rounds. At the beginning of each round, the sink floods a request for messages to all nodes in the network using the standard dis-

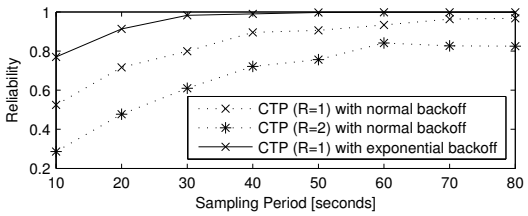

Figure 5. Reliability vs. Congestion

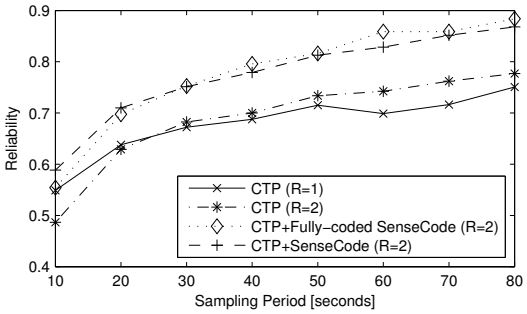

Figure 6. Reliability vs. Sampling rate

semination protocol of TinyOS. We define *Sampling Period* to be the duration of each round. After receiving the request, the source nodes insert packets in the network at random times during the first half of the round. We made this choice as we found in our experiments that the network becomes seriously congested if all the source nodes insert their transmitted packets in a synchronized manner.

Metrics We use two performance metrics.

Reliability quantifies the amount of useful information that reaches the sink. We compute the reliability 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, reliability 1 means that all messages sent to the sink were successfully decoded.

Transmission energy quantifies the amount of energy spent in transmitting packets. We compute it by measuring the time the radios of the nodes are in the transmit state and multiplying it by the power consumed by the transceiver in TX mode. Since TOSSIM simulates MicaZ sensors, we used the power-consumption values provided in [29].

Every data point plotted in the graphs in this section is an average measured over a sufficiently large number of experiments (ranging from 5 to 100). Each experiment simulates the network for 10^4 seconds.

Backoff mechanism When we started experimenting with CTP, we initially used the CTP code that comes with TinyOs. However, we observed that, when the sampling period is small, reliability is low because of network congestion. In such a situation, adding redundancy not only does not help, but is, in fact, detrimental to reliability, as it increases the network load, which in turn leads to more congestion. Figure 5 shows that adding redundancy $R = 2$ to a congested network decreases reliability by 25%. Then we realized that CTP's congestion-avoidance mechanism was simple backoff; we substituted this with exponential backoff, which led to a significant improvement in reliability (in certain cases up to 30%), as plotted in Figure 5. In the rest of our experiments, we consistently use exponential backoff.

4.2 Dynamically Changing Networks

In this section, our goal is to test how well our protocols adapt to a changing network topology. We thus study networks where nodes can be occluded and therefore are temporarily disconnected.

Node occlusion is modeled using a two-state Markov Chain. Each node can be either in a connected or in a disconnected (outage) state. We use as parameters of the model the *mean time between failures* (MTBF), i.e., the average time a node is in the connected state, and the *mean time to repair* (MTTR), i.e., the mean time a node stays disconnected. Every second and for every node a biased coin decides whether it should change state or not. If a disconnected (in outage) node happens to have packets to send, it will keep trying to contact its parent and these transmissions will fail. Eventually, if the outage lasts too long, the maximum number of retransmissions will be exceeded and the packet will be discarded.

We first study reliability as a function of the sampling period. Figure 6 shows, for each protocol, the reliability achieved, while Figure 7 shows the associated energy consumption per round. In these experiments we set MTTR to 1000 seconds and MTBF to 100 seconds, thus roughly 90% of the nodes are up at any given time.

First, we observe that when the network is congested, i.e., for very low values of the sampling period, introducing redundancy does not help, as it increases the network load, and can in fact deteriorate the performance in the case of CTP with redundancy. As the sampling rate increases and congestion is no longer the dominating cause of losses, we observe that CTP with redundancy slightly improves the reliability of CTP by at most 5%; on the other hand, SenseCode successfully communicates up to 20% more messages to the sink than CTP with redundancy.

One reason for the lower performance of CTP with redundancy is that, both packets follow the same route to the sink, and thus if the selected route is perturbed due to node disconnection, the tree adaptation is not sufficiently fast to successfully reroute packets already in transit to the sink. 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. Thus, redundancy alone without spatial spreading is not sufficient.

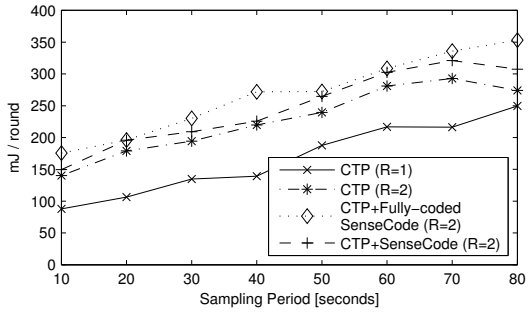

Figure 7. TX Energy vs. Sampling rate

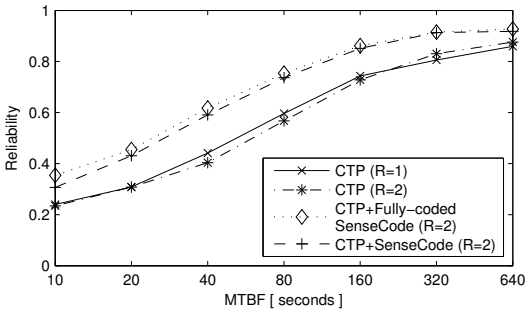

Figure 9. Reliability vs. MTBF

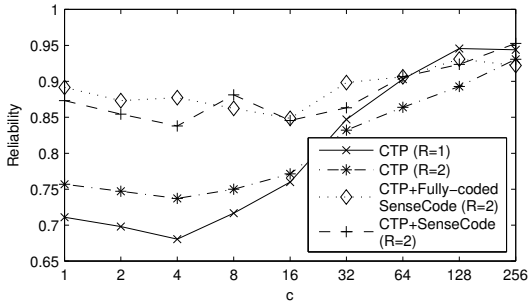

Figure 8. Reliability vs. Frequency: the MTBF of each experiment is set to be $100 \cdot c$ while the MTTR is set to be $10 \cdot c$

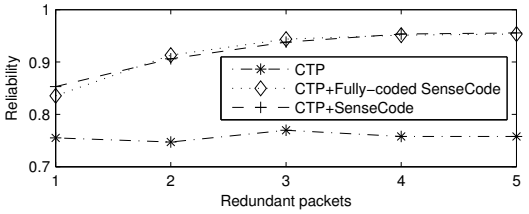

Figure 10. Reliability vs. Redundancy

Second, from Figure 7 we can see that introducing a redundancy factor of $R = 2$, increases the transmit energy consumption for all protocols, but does not necessarily double it: this is because control traffic used to broadcast requests and maintain the tree does not change with R.

Third, we note that as expected, SenseCode and fully-coded SenseCode have similar reliability, however, SenseCode has lower energy consumption as it uses fewer coded packets and thus avoids part of the overhead of the coding vectors. The experiments plotted in Figure 7 use a payload of 77 bytes, which amounts to an overhead of 23% in terms of bytes and per coded packet. This overhead, as compared to CTP with $R = 2$, results in an overhead in terms of energy of 8% for SenseCode and 17% for fully-coded SenseCode, i.e., the energy overhead doubles for fully-coded as expected. Note that part of the energy used for a packet transmission does not depend on the length of the payload; but we expect that, as the payload increases, the energy overhead per coded packet will become negligible. Indeed, when we used a payload of 99 bytes, which amounts to a per coded packet overhead of 18% in bytes, the energy overhead of fully-coded SenseCode reduced from 17% to 13%. We here discuss and depict only energy consumed in transmission; we discuss energy consumed in packet reception in Section 4.3.

We next look deeper in the performance of our protocols in a regime where dynamic changes (as opposed to network congestion) is the dominating cause of performance deterioration. We thus fix the sampling rate to 60 seconds, which leads to a network state without congestion. First, we plot in Figure 8 how the reliability behaves when the frequency with which nodes change state increases. We can see that as the network changes more slowly, CTP alone becomes sufficient, as tree adaptation has more time to rebuild the tree.

Second, we explore, in Figure 9, the reliability as a function of the average fraction of nodes that are up at any given time. In this case, we fix the MTTR to be 10 seconds and we vary the MTBF. We again observe that when nodes fail frequently, even if they recover rapidly CTP cannot adapt the tree sufficiently fast resulting in a low reliability, while SenseCode offers an improvement.

Finally, we explore in Figure 10 how adding redundancy larger than $R = 2$ may help. In this experiment the sampling rate is fixed to 60 seconds, the MTBF to 1000 seconds and the MTTR to 100 seconds. We can see, from the graph, that SenseCode allows to increase the reliability as more redundancy is added while with simple CTP the performance improvement quickly flattens out.

4.3 Amount of Overhearing

In typical sensor hardware [28, 29], the energy consumed for packet transmission and reception is comparable; thus, one could argue that the benefits of SenseCode our outweighed by energy consumed overhearing. Note that the current implementation of CTP also has nodes constantly overhear from their neighbors to collect connectivity information; however, one may imagine that an improved version of CTP could reduce the current amount of overhearing. This

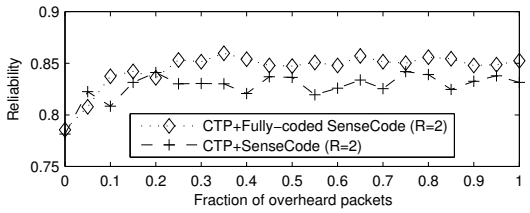

Figure 11. Reliability vs. Overhearing: in the same setup CTP (R=1) has reliability 70% and CTP (R=2) has performance 74%

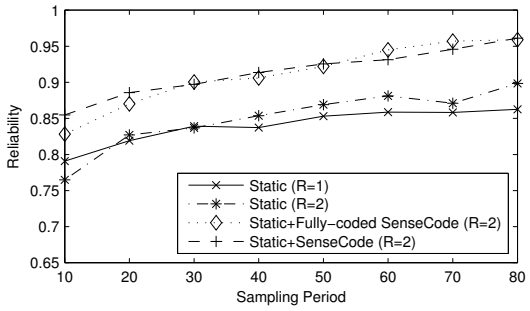

Figure 12. Reliability vs. Sampling rate

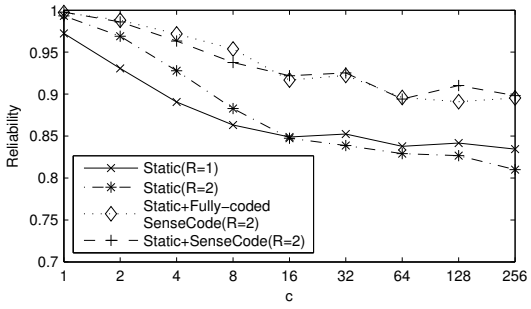

Figure 13. Reliability vs. Frequency: the MTBF of each experiment is set to be $100 \cdot c$ while the MTTR is set to be $10 \cdot c$

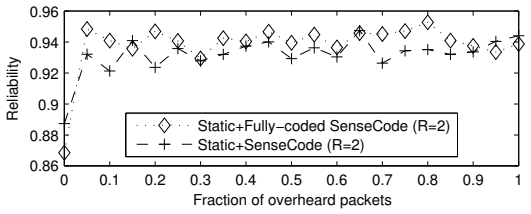

Figure 14. Reliability vs. Overhearing: in the same setup Static (R=1) has reliability 85% and Static (R=2) has performance 88%

section studies how reduced overhearing affects the performance of SenseCode.

Figure 11 shows the performance of SenseCode as a function of the percentage of overheard packets. In our experiment we flip a biased coin each time a packet is overheard by CTP to decide whether we use it in SenseCode. The sampling rate is set to 60 seconds, the MTBF to 1000 seconds and the MTTR to 100 seconds. We see that even with 0% overhearing, SenseCode offers a significant improvement in reliability as compared to CTP: this is because the tree adaptation alone can result in packet mixing. Moreover, overhearing a small fraction of packets we get the same advantage as constant overhearing.

4.4 Adaptive vs. Static Routing

In this section, we examine whether it is sufficient to rely solely on SenseCode to recover from dynamic changes in the network. Thus, we run SenseCode (and fully-coded SenseCode) on top of a static, as opposed to an adaptive tree.

Figure 12 shows the reliability of the protocols when the MTBF is set to 1000 seconds and the MTTR to 100 seconds. We observe that the achieved reliability is higher than when running SenseCode on top of an adaptive tree (compare with Figure 6). This indicates that, when the network changes are fast, attempting to adapt the tree might obstruct the information flow. Several not essential packets may get misdirected towards the same surviving paths causing congestion. In such cases, it is better to rely on spatial coding for recovery. Notice that even at low sampling periods the network is not in congestion, and therefore introducing redundancy through SenseCode provides an advantage.

In Figure 13, we see that, when the tree is static, higher frequencies of node-state changes entail higher reliability. Contrast this with CTP (see Figure 8) where the opposite occurs. This happens because, if the nodes recover fast, they can forward their packets before the round finishes.

Figure 14 shows how overhearing contributes to the performance of SenseCode when using static routing. We can see that SenseCode doesn't have a significant advantage over CTP when no transmissions are overheard: this is expected since there are no routing path changes that can help mix the information from the descendants of a failed node. However, notice that as soon as few packets are overheard the performance of SenseCode reaches its maximum.

5. Discussion

5.1 Multipath

We now compare our approach to traditional multipath communication, the most popular existing approach for balancing reliability 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 [6, 16, 17, 30].

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) [16]. 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 [21]. For instance, we expect a 2-multipath protocol to consume the same amount of energy as SenseCode with redundancy factor 2, yet achieve lower reliability (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 [19]. A different approach that has recently been proposed is subspace coding [22, 23]. Unfortunately, this approach proves also difficult to deploy in sensor networks [24].

In parallel and complementary to this work, we proposed a novel design with much lower overhead based on the use of *compressed* coding vectors [24]. 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 [24]. 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 [1, 2], see also for tutorial articles and monographs [3, 4], and has been successfully applied in wireless ad-hoc and mesh networks, see for example [5]–[8]. 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 [7] is in bandwidth efficiency through per-flow network coding; we are instead in optimizing for energy and reliability coding for inverse multicast traffic, thus coding across several information sources.

Protocols for network coding over sensor networks have been proposed for example in [9]–[10], 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 [11, 12] from a viewpoint of distributed storage. Between these, the work closer to ours is [12], which proposes a form of spatial coding to improve partial recovery of data in emergency situations. The emphasis in [12] 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 [12] 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. 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 [13, 14], 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 [15–17]. The use of coding we introduce allows to take advantage of broadcasting, with reduced control information, and allows to achieve different points in the energy-reliability trade-off.

Synopsis diffusion [18] 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. These differences have lead us to develop a generalized framework, which we term clue diffusion, and that accepts both network coding and synopsis diffusion as special cases. For lack of space, this is provided in Appendix 7.

7. Conclusion

We propose SenseCode, a communication protocol specifically designed for sensor networks that leverages network coding techniques, and can be efficiently deployed in sensor networks. SenseCode targets the case where we require increased redundancy inside the network. We showed through experimental results that with our techniques we can achieve an increased level of robustness without additional control information and minimal modifications of existing protocols. TOSSIM simulations show that SenseCode provides significantly increased reliability as compared to state-of-the art alternative approaches.

References

[1] R. Ahlswede, N. Cai, S-Y. R. Li, and R. W. Yeung, "Network information flow", *IEEE Trans. Information Theory,* vol. 46, pp. 1204–1216, July 2000.

[2] S-Y. R. Li, R. W. Yeung, and N. Cai, "Linear network coding," *IEEE Trans. Information Theory,* vol. 49, pp. 371–381, Feb. 2003.

[3] C. Fragouli, J. Widmer, and J.-Y. L. Boudec, "Network coding: An instant primer", *ACM SIGCOMM Computer Communication Review*, vol. 36, no. 1, pp. 63–68, Jan. 2006.

[4] T. Ho and D. S. Lun, "Network coding: An introduction," *Cambridge University Press, Cambridge, U.K.*, 2008.

[5] Y. Wu, P. A. Chou, and S. Y. Kung, "Minimum-energy multicast in mobile ad-hoc networks using network coding", *IEEE Trans. on Communications*, vol. 53, no. 11, pp. 1906–1918, Nov. 2005.

[6] Z. Li and B. Li, "Improving throughput in multihop wireless networks", *IEEE Trans. on Vehicular Technology*, vol. 55, no. 3, May 2006, pp. 762-773.

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

[8] Sachin Katti, Dina Katabi, Hari Balakrishnan, and Muriel Medard, "Symbol-level network coding for wireless mesh networks," *ACM SIGCOMM*, 2008.

[9] C. Adjih, S. Y. Cho, and P. Jacquet, "Near optimal broadcast with network coding in large sensor networks", *1st International Workshop on Information Theory for Sensor Networks (WITS)*, Jun. 2007.

[10] D. Petrovic, J. Rabaey, K. Ramchandran, "Overcoming untuned radios in wireless sensor networks with network coding", *Network Coding Workshop,* 2007.

[11] A. G. Dimakis, V. Prabhakaran, and K. Ramchandran, "Ubiquitous access to distributed data in large-scale sensor networks through decentralized erasure codes", *IEEE IPSN*, 2005.

[12] A. Kamra, V. Misra, J. Feldman, and D. Rubenstein, "Growth codes: maximizing sensor network data persistence", *ACM SIGCOMM*, 2006.

[13] M. Sartipi and F. Fekri, "Source and channel coding in wireless sensor networks using LDPC codes", *IEEE SECON*, 2004.

[14] A. Wood and J. Stankovic, "Rateless erasure codes for bulk transfer in asymmetric wireless sensor networks",*ACM SenSys*, 2008.

[15] G. Pottie and W. Kaiser, "Principles of embedded networked systems", *Cambridge University Press*, 2005.

[16] D. Ganesan, R. Govindan, S. Shenker and D. Estrin, "Highly-resilient, energy-efficient multipath routing for wireless sensor networks", *ACM MOBIHOC*, 2001.

[17] S. De, C. Qiao and H. Wu, "Meshed multipath routing with selective forwarding: an efficient strategy in wireless sensor networks", *The International Journal of Computer and Telecommunications Networking*, 2003.

[18] S. Nath, P. B. Gibbons, S. Seshan, and Zachary Anderson, "Synopsis diffusion for robust aggregation in sensor networks", *ACM SenSys*, 2004.

[19] P. A. Chou, Y. Wu, and K. Jain, "Practical network coding", *Allerton Conference on Communication, Control, and Computing*, 2003.

[20] Horn and Johnson, "Matrix Analysis", *Cambridge University Press*, 1990.

[21] F. J. MacWilliams and N. J. A. Sloane, "The theory of error correcting codes", *North-Holland Mathematical Library*, 1983.

[22] R. Koetter and F. Kschischang, "Coding for errors and erasures in network coding", *IEEE ISIT*, 2007.

[23] D. Silva and F. R. Kschischang, "Using rank-metric codes for error correction in random network coding", *IEEE ISIT*, 2007.

[24] M. Jafari, L. Keller, C. Fragouli, and K. Argyraki, "Compressed coding vectors", *IEEE ISIT*, 2009.

[25] P. Levis, N. Lee, M. Welsh, and D. Culler, "TOSSIM: Accurate and scalable simulation of entire TinyOS applications", *ACM SenSys*, 2003.

[26] "TinyOs", http://www.tinyos.net/.

[27] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss and P. Levis, "Collection tree protocol", ACM SenSys, 2009.

[28] "TinyNode", http://www.tinynode.com/.

[29] "TI CC2420 RF Transciever Datasheet", http://www.ti.com/lit/gpn/cc2420

[30] G. Pottie, and W. Kaiser, "Wireless Sensor Networks", *Communications of the ACM*, vol. 43, no. 5, pp. 51, May 2000.

[31] L. Keller, E. Atsan, C. Fragouli, K. Argyraki, "SenseCode: Network coding for reliable sensor networks", EPFL Technical Report, available at *http://infoscience.epfl.ch*

Appendix: Clue diffusion

We here describe the clue diffusion framework that generalizes the network coding and synopsis diffusion framework. To capture the differences of network coding with synopsis

diffusion: (i) we introduce a state associated with every network node. A projection operator produces packets to send that depend on the node state but do not neccessarily contain all the useful information the state has, and (ii) we introduce the notion of approximate (with high probability) validity of function properties, such as reconstruction at the sink.

Variables

The goal of the information collection protocol is for the sink to estimate a value in a set D that is a function of the network node inputs.

Each network node i:
(i) observes an input x_i,
(ii) maintains a state σ_i,
(iii) sends and receives packets c.
The input takes values in a set X. The state and packet values are drawn from the same set C. We call the elements of C clues.

Functions

All the information a network node has is reflected in its state. The state is initialized by the input the node measures. A node receiving packets uses them to update its state. A node sends packets that are a function of its current state. These network operations can be described through the following functions:
1. A *clue generation function* $\mathbf{CG} : X \to C$ that creates a clue from an input; this function is deterministic. This is the state initialization function.
2. A *clue fusion function* $\mathbf{CF} : C \times C \to C$ that aggregates the two clues to create a new one. This is the state update function, used to update the state once a new packet is received.
3. A *clue projection function* $\mathbf{CP} : C \to C$ that creates a clue from another one. This is the packet generation function that uses the state information to create the next packet to send.

If the sink had available all the inputs the network nodes had observed, it could calculate an *end-to-end function* $F : X \to D$ that given the readings on the nodes returns a value in the domain D.

Instead, the sink is going to use its received packets to update its state, and eventually, calculate an output that is a function of its state. We capture this through an *estimation function* $f : C \to D$ that is used to estimate the end-to-end function from a clue.

DEFINITION 1. *Given a protocol $\mathcal{P}$, a network state $x = (x_1, \ldots, x_N)$, and the estimate $\bar{f}$ produced by the sink, we say that $\bar{f}$ is:*

1. Correct, *if $\bar{f} = F(x)$.*
2. Correct with high probability, *if $\bar{f} = F(x)$ w.h.p.*
3. δ-strong Correct, *if given $|\cdot|$, a norm for elements of D*

$$|F(x) - \bar{f}| \leqslant \delta.$$

Special cases

Network Coding

1. The set of clues C is the set of all possible subspaces of $\mathbf{F}_q^\ell$, where ℓ is the packet length. Each subspace π can be determined by m vectors of $\mathbf{F}_q^\ell$, with $m = \dim(\pi)$.
2. The clue generation function $\mathbf{CG}$ associates each reading to a subspace π of $\mathbf{F}_q^\ell$.
3. The clue fusion function is union of the two input spaces $\mathbf{CF}(\pi_1, \pi_2) = \pi_1 \cup \pi_2$.
4. The clue projection function generates a subspace $\mathbf{CP}(\pi) = \mathbf{span}(v)$, where v is a vector randomly sampled from π. The clue generated can therefore be described by 1 vector instead of $\dim(\pi)$ vectors.

Synopsis diffusion

We here use the function notation in [18].

1. The set of clues C is the set of all synopses S.
2. The clue generation function $\mathbf{CG}$ is the function $\mathbf{SG}$.
3. The clue fusion function is the function $\mathbf{SF}$.
4. The clue projection function is $\mathbf{CP}(s) = s$.

Order and duplicate insensitive collection

DEFINITION 2. *(Duplicate insensitive functions) The triple of functions $\mathbf{CG} : X \to C$, $\mathbf{CF} : C \times C \to C$, $\mathbf{CP} : C \to C$ are called order- and duplicate insensitive if the following properties hold:*
P1: (Commutative property) $\forall s_1, s_2 \in S$, $\mathbf{CF}(s_1, s_2) = \mathbf{CF}(s_2, s_1)$.
P2: (Associative property) $\forall s_1, s_2, s_3 \in S$ $\mathbf{CF}(\mathbf{CF}(s_1, s_2), s_3) = \mathbf{CF}(s_1, \mathbf{CF}(s_2, s_3))$.
P3: (Same synopsis idempotent) $\forall s \in S$, $\mathbf{CF}(s, s) = s$.
P4: (Information conservation of projection) $\forall c_1, c_2 \in C : f(\mathbf{CF}(\mathbf{CP}(c_1), c_2)) = f(\mathbf{CF}(c_1, c_2))$.

Under the previous conditions, both synopsis diffusion and network coding achieve information dissemination transparent to the node topology and the ordering of the operations. The proof of this for synopis is provided in [18], while for network coding it is a direct implication of proof of the main theorem in [1].