Multipath Diversity and Robustness for Sensor Networks

Christina Fragouli, Katerina Argyraki, and Lorenzo Keller

Abstract Balancing energy efficiency and reliability is a common underlying goal for most information collection protocols in sensor networks. Multipath diversity has emerged as one of the promising techniques to achieve such a balance. In this chapter, we provide a unified framework for the multipath techniques in the literature and discuss their basic benefits and drawbacks. We also discuss emerging techniques from the area of network coding.

1 Introduction

The goal of a sensor network is to gather information and communicate it to a collecting entity: sensors typically measure a physical quantity (e.g., temperature) and communicate their measurements (or functions of their measurements, e.g., sums or averages) to a common *sink*. For most applications, sensors communicate to the sink over a wireless channel, while the sink is connected to a wired network. In this chapter, we focus on the problem of designing a *collection protocol*, i.e., a protocol that conveys information from the sensors to the sink.

A typical requirement for such protocols is energy efficiency: sensors are typically powered through small, easily depletable batteries; moreover, they are often installed at inaccessible areas (e.g., forests or mountain slopes), making battery changing practically infeasible. Hence, a collection protocol must be "energy efficient," i.e., consume the minimum amount of energy necessary to meet the performance standards of the corresponding application. The main source of energy consumption in a sensor is the radio, i.e., transmitting data and listening for/receiving incoming data (a per-component breakdown of energy consumption for different sensor platforms can be found in [1]). Hence, one approach to designing energy-efficient collection protocols is to minimize the number of transmissions necessary

C. Fragouli (✉)
School of Computer and Communication Sciences, EPFL, Lausanne, Switzerland
e-mail: christina.fragouli@epfl.ch

G. Ferrari (ed.), *Sensor Networks,* Signals and Communication Technology,
DOI 10.1007/978-3-642-01341-6_5,

to convey the required information to the sink. Another, complementary approach is to enable sensors to predict for which periods they are unlikely to receive data and turn their radios off during those periods (a collection protocol that uses this approach is described in [2]).

Energy efficiency is at odds with another typical requirement for collection protocols: reliability in the face of channel and node failures. Compared to wired, wireless channels are relatively unreliable, subject to fading and random fluctuations. This is especially so in sensor networks where the environment can change unpredictably and the sensors themselves can fail (e.g., a landslide can destroy part of a sensor network monitoring forest conditions). To achieve reliable collection in the face of such failures, a collection protocol must generate a certain amount of redundant information – which unavoidably increases both the number of transmissions and the amount of time for which sensors must keep their radios on.

In this chapter, we describe existing as well as emerging collection protocols that balance the competing goals of energy efficiency and reliability.[1] We restrict our attention to the simplest scenario, where the goal is to communicate sensor measurements to the sink,[2] while sensors never turn off their radios. The point of the chapter is to give a flavor of the problems encountered in sensor-network design and outline the different classes of solutions – we far from cover all work done in the area. We start by defining a collection protocol and its associated cost in Sect. 2. Section 3 examines the use of single-path protocols, while Sect. 4 focuses on multipath protocols. Section 5 reviews basic ideas in the area of network coding, while Sect. 6 outlines an approach for deploying network coding in sensor networks. Finally, Sect. 7 concludes this chapter.

2 What is a Collection Protocol?

A fundamental characteristic of wireless communication is that transmissions are broadcast and can be overheard by multiple nodes in the transmitter's vicinity, albeit at different signal levels. For example, in Fig. 1, a transmission by node a_{11} is overheard by nodes a_{10}, a_9 and a_8 at different signal levels; we say that a_{10}, a_9 and a_8 are in a_{11}'s *range* or, equivalently, that they are a_{11}'s "neighbors." As a result, there can be multiple paths between each pair of nodes – Fig. 2 depicts all possible paths between nodes in our example network. A collection protocol essentially determines which of these paths to use and how, in order to convey information from the sensors to the sink.

---

[1] Certain applications can have additional/different requirements, e.g., in a sensor network monitoring for atmosphere poisoning, alarms must to be delivered with minimum latency.

[2] We do not consider the scenario where the goal is to communicate *functions* of sensor measurements, like sums or averages, to the sink.

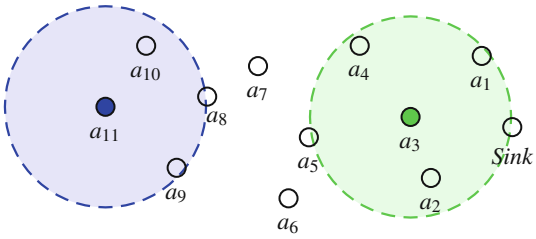

Fig. 1 Wireless transmissions are broadcast. There may be several nodes within the broadcasting range of every sensor node

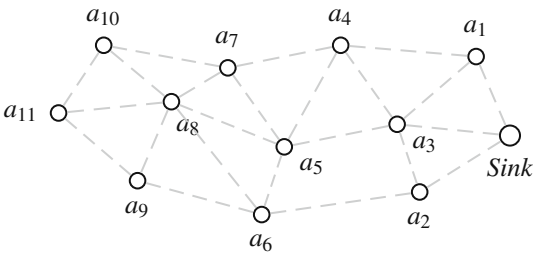

Fig. 2 Representation of the potential connections between neighboring nodes

More specifically, a collection protocol can be decomposed into the following tasks:

1. *Acquisition of channel information*. Each node learns its neighbors and estimates the channel quality between itself and each neighbor.
2. *Topology construction*. Each node builds a routing table that includes a subset of its neighbors; e.g., when the constructed topology is a tree, each node has a single next hop to the sink.
3. *Topology usage*. Each node determines how to route information through the constructed topology – that is, what exactly to send, how often, and to which neighbors.

2.1 Path Cost and Channel Quality

Recall that we are interested in balancing reliability and energy efficiency. As mentioned in Sect. 1, there are two approaches to reducing energy consumption: reduce transmissions and reduce the amount of time for which sensors keep their radios on. Since we are considering a simple scenario where sensors cannot do the latter, in our context, optimizing for energy efficiency means minimizing transmissions. Hence, we define the cost of communication as a function of the number of transmissions required to successfully transfer a certain amount of information between two nodes. More specifically:

Definition 1 We define the cost of communication from a node a to a neighbor b as the average number of transmissions that are necessary to successfully transmit a packet of length L from a to b.

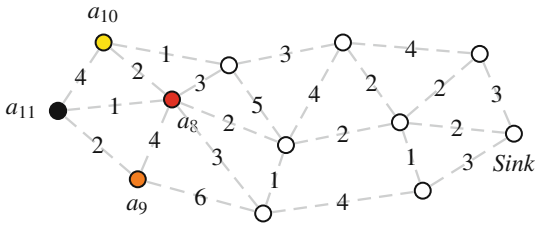

Fig. 3 A graph with vertices corresponding to sensor nodes and edges between any two nodes that are within the transmission range of one another. Each edge has a weight proportional to the number of retransmissions required to reliably transmit information between the nodes it connects

Suppose we know the communication cost between every pair of neighbors in our network at a given point in time. We can depict this information with a *channel-quality graph*, where each vertex represents a sensor and each edge represents a communication channel between two neighbors; each edge has a weight proportional to the corresponding cost. For example, the graph shown in Fig. 3 says that, if node a_{11} broadcasts once, only node a_8 is expected to successfully receive the transmitted packet; if it broadcasts twice, nodes a_8 and a_9 are expected to successfully receive the packet; if it broadcasts 4 times, all neighbors are expected to successfully receive the packet.

Definition 2 Given a channel-quality graph, we define the cost of a path as the sum of the costs of all the edges that make up the path. A *minimum-cost path* from a node a to the sink is a path whose cost is less or equal to the cost of any other path from node a to the sink.

3 Routing on a Tree

A straightforward approach is to connect all sensors to the sink over a tree: (i) Construct a unique, minimum-cost path from each sensor to the sink (such a path is shown in Fig. 4), such that the union of all constructed paths forms a *spanning tree*

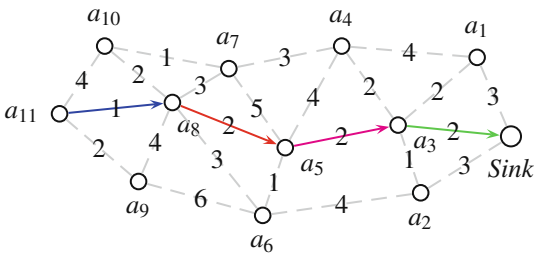

Fig. 4 A minimum-cost path from node a_{11} to the sink. This is also the minimum-cost path for nodes a_8, a_5, a_3 to the sink

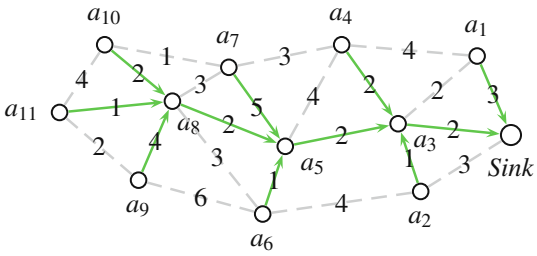

Fig. 5 A tree constructed from the union of the minimum-cost paths from the sensors to the sink

rooted at the sink (such a tree is shown in Fig. 5).[3] (ii) Route each message from sensor a to the sink along the unique path from a to the sink over the tree.

The Collection Tree Protocol (CTP) [3] implements this approach. Task #1 (acquisition of channel information) is accomplished in a setup phase, where each node sends out a sequence of numbered beacons that bear its identity; each node processes the beacons it hears and determines its neighbors; for each neighbor, it counts how many of its beacons it received, estimates the communication cost from that neighbor to itself, and sends this information to the neighbor; beacons are sent with an exponentially increasing interval, which is reset to a small value when certain routing conditions are met. Task #2 (topology construction) is accomplished in a distributed manner over multiple rounds: in each round, each node broadcasts its "distance" to the sink, i.e., the cost of the minimum-cost path from itself to the sink; moreover, each node processes its neighbors' advertisements, identifies the neighbor with the lowest distance to the sink, and chooses it as its "next hop." For example, in Fig. 5, in the first round, nodes a_1, a_2 and a_3 identify the sink as their neighbor; in the second round, nodes a_2, a_4 and a_5 identify node a_3 as their neighbor with the lowest distance to the sink and choose it as their next hop. Finally, task #3 (topology usage) consists of each node transmitting every message it receives to its next hop. For example, in Fig. 5, a message from node a_{11} is routed through nodes a_8, a_5 and a_3.

Routing on a tree faces certain practical challenges:

1. *Changing network conditions*. The tree can break as a result of a node or channel deterioration between two neighbors; moreover, due to channel fluctuations, what used to be an optimal tree may end up using bad-quality paths. To decrease the amount of loss or delay that can result from such events, it is possible to repeat tasks 1 and 2, i.e., monitor channel quality and *adapt* the tree accordingly (CTP follows this approach). However, it is not possible to adapt to all failures: suppose that node a_{11} in Fig. 5 sends a packet to the sink through the constructed

[3] We should clarify that a spanning tree constructed in this manner (i.e., the union of the minimum-cost paths connecting each sensor to the sink) is not necessarily a "minimum-cost spanning tree" as typically defined in graph theory, where the cost of a tree is equal to the sum of the costs of all links that take part in the tree. For example, the tree in Fig. 5, which has cost 25, is not a minimum-cost tree.

tree (nodes a_8, a_5 and a_3); as the packet reaches node a_5, all channels from node a_5 to its neighbors fail; as a result, there is no path connecting a_5 to the sink, and the packet is lost, i.e., a_{11} fails to communicate with the sink, even though there still exists a path from a_{11} to the sink (through nodes a_9, a_6 and a_2). So, it is possible that the tree is unable to adapt fast enough to certain failures, especially in large networks, resulting in temporary disconnection of parts of the network.

2. *Depletion of specific nodes*. Nodes located close to the root of the tree (e.g., node a_3 in Fig. 5) may end up carrying significantly more traffic than the rest, as they participate in multiple paths; as a result, their batteries are depleted significantly faster. Note that these nodes are precisely the ones that need to work in order for the network to remain connected: if node a_3 in Fig. 5 fails, 9 of the remaining 10 nodes are disconnected from the sink.

4 From Tree to Multipath Routing

In the last section, we looked at a collection protocol that constructs a single path from each sensor to the sink. In contrast, multipath collection protocols construct multiple paths from each sensor to the sink, allowing more freedom in route selection. Intuitively, multipath protocols achieve higher reliability than single-path protocols at the cost of higher energy consumption.

4.1 Topology Construction

A node can be connected to the sink over disjoint or overlapping paths; we now discuss these two approaches, as well as some associated computer science problems.

4.1.1 Disjoint Paths

The idea is to construct m disjoint paths from each node to the sink. These paths can be *edge-disjoint* (Fig. 6), providing reliability in the face of channel deterioration; or *vertex-disjoint* (Fig. 7), providing reliability in the face of node failures. Alternatively, a recent proposal suggests that the paths should be "one-hop apart" (Fig. 8), i.e., if a node is used in one of the paths, none of its neighbors should be used in the remaining $m - 1$ paths; the point is to avoid spatially correlated failures [4].

Intuitively, the more constraints we add to path construction, the higher the resulting reliability: Fig. 6 shows two edge-disjoint paths; if node a_8 fails, both paths from a_{11} to the sink are broken. Figure 7 shows two vertex-disjoint paths; if node a_8 fails, a_{11} can still communicate with the sink through the other path. On the other hand, more constraints result in higher-cost paths: given the channel-quality graph of Fig. 3, Fig. 4 shows the minimum-cost path from node a_{11} to the sink. If we want to construct two paths from a_{11} to the sink that are one-hop apart, we

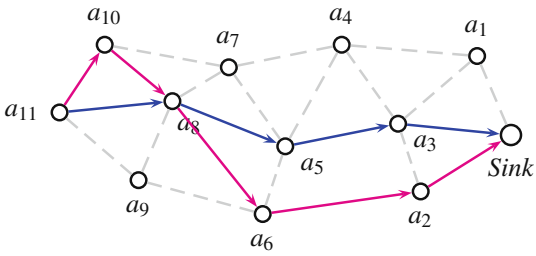

Fig. 6 Two edge-disjoint paths connecting node a_{11} to the sink

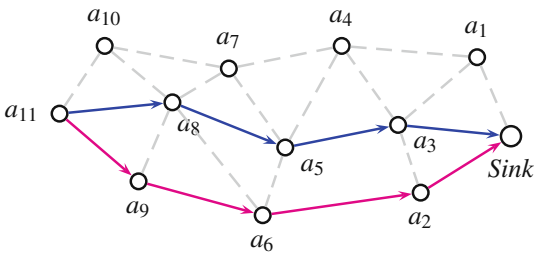

Fig. 7 Two vertex-disjoint paths connecting node a_{11} to the sink

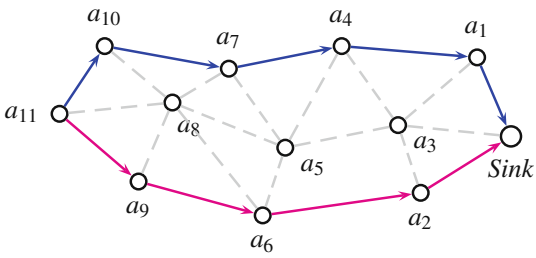

Fig. 8 Two one-hop separated paths connecting node a_{11} to the sink

necessarily have to choose the two paths shown in Fig. 8, none of which is equal to the minimum-cost path. In general, the further away a path is located from the minimum-cost path, the higher its expected cost.

We should note that it is not always feasible to find paths that satisfy such constraints. For example, m edge-disjoint paths exist between two nodes only if the edge min-cut between them is greater or equal to m; similarly, m vertex-disjoint paths only exist if the vertex min-cut between the two nodes is greater than m.

4.1.2 Algorithmic Complexity of Disjoint-Path Construction

Consider a collection protocol that builds two disjoint paths from each sensor to the sink. To minimize the number of transmissions, for each sensor, the protocol must choose the two paths to the sink that have the minimum *overall* cost (i.e., the sum of their costs is less than or equal to the sum of the costs of any other two paths from this sensor to the sink). This problem is known in computer science as the "min-sum 2-path" problem, and it has been shown to be computationally hard to solve (NP-hard). The difficulty comes from the fact that the optimal solution may

involve two paths neither of which is the minimum-cost path, as Example 1 below illustrates.

Example 1 Figure 9 shows two sets of paths that connect node a_{11} to the sink: (i) The first set includes the minimum-cost path (of cost 7) and the second best path (of cost 13). The total cost is 20. (ii) The second set includes two paths, each of cost 9. The total cost is 18, which is the minimum overall cost. This example illustrates that the optimal solution cannot be obtained simply by ranking paths by cost and taking the two best paths.

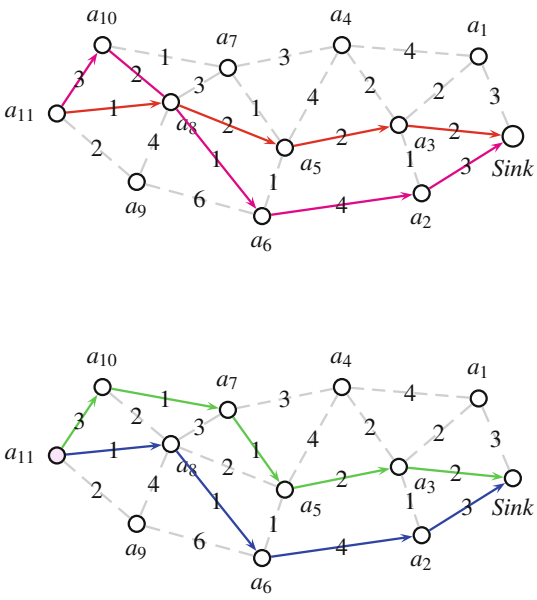

Fig. 9 Two sets of path choices to connect node a_{11} to the sink: the first set includes the minimum-cost path (going through a_1, a_5 and a_3), and the second best path (going through a_{10}, a_8, a_6 and a_2). The second set includes two paths that have the minimum sum cost

So, identifying, for each sensor, two disjoint paths to the sink with the lowest overall cost is hard. An alternative would be to construct *any* two disjoint paths, without looking to minimize the overall cost. We now illustrate that constructing disjoint paths using heuristics can easily lead to configurations where nodes use paths of significantly higher cost than necessary.

Example 2 A computationally reasonable way to select two disjoint paths from each sensor to the sink would be the following:

- Select a minimum-cost path from each sensor to the sink. The union of chosen paths forms a tree T_1.
- Remove all the edges of T_1 from the channel-quality graph.
- Considering only the remaining links, select again a minimum-cost path from every sensor to the sink, which results in a tree T_2.

Figures 10, 11 and 12 illustrate this procedure and show a problematic case, where, in the second tree, node a_3 is connected to the sink through a path of cost 9, whereas,

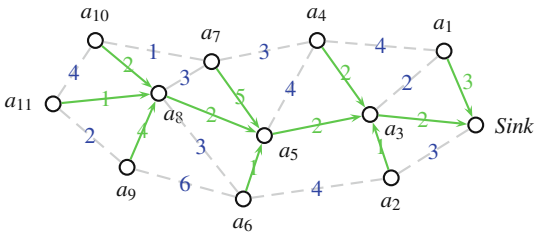

Fig. 10 The first tree T_1 on the cost graph

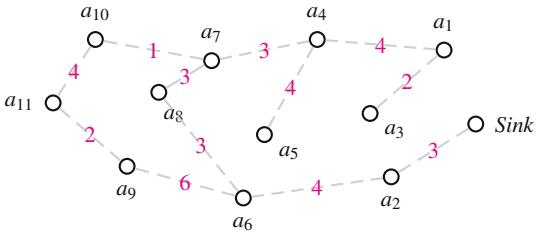

Fig. 11 The remaining cost graph after removing T_1

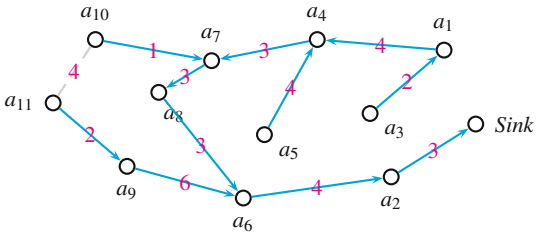

Fig. 12 The second tree T_2. If the paths to be employed are not jointly selected, we may fall into inefficient path constructions

in the original network, there exist edge-disjoint paths of costs 2, 4 and 5, respectively, that connect a_3 to the sink.

4.1.3 Braided Paths

The idea is to first construct a "primary" minimum-cost path from each node to the sink (as described in Sect. 3), then build a "braid" of alternative paths as follows: for each node on the primary path, build a path that does not include that node [5, 6].

Example 3 Consider the braided path choices depicted in Fig. 13. The minimum-cost path from node a_{11} to the sink involves hops through nodes a_8, a_5 or a_3. If node a_8 fails, it can be replaced by the path segment through nodes a_{10} and a_7. Similarly, node a_5 can be replaced through nodes a_6 and a_2, while node a_3 can be replaced through nodes a_4 and a_1.

Intuitively, braided paths are more energy-efficient than disjoint paths, simply because they are more likely to be physically close to the primary (minimum-cost) path. One would think that this higher energy efficiency would come at the cost

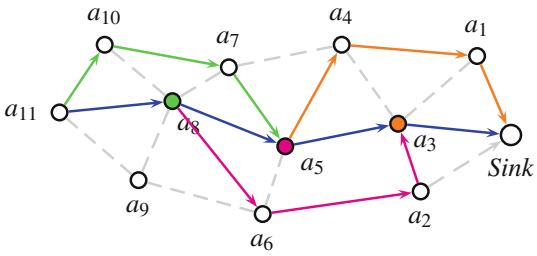

Fig. 13 Braided paths: in the path from node a_{11} to the sink, if any of the nodes a_8, a_5 or a_3 fails, there exists an alternative available path. For example, node a_5 can be replaced by using the path segment through nodes a_6 and a_2

of lower reliability (the reason being that braided paths would be less resilient to multiple-node failures than disjoint paths). Interestingly, early simulation results suggest that this may not be the case: it turns out that, in practical scenarios, braided paths can be as failure-resilient as 2-disjoint and 3-disjoint paths; the intuition is that, in practice, disjoint paths are not physically separated from one another enough to survive multiple-node failures [5].

4.2 Topology Usage

There are several choices on how to use multiple paths from a node to the sink; which one is better depends on the specific application requirements. We group the proposed uses in five broad categories; the first three are relevant for disjoint paths, while the others for braided ones.

4.2.1 Replicate Transmissions

Each source (i.e., each sensor that has a message to convey to the sink) sends its message through all available paths to the sink. This approach is sometimes also described as flooding.

Flooding is typically proposed in conjunction with disjoint paths, where each path carries a replicate of the same message to the sink. Indeed, to flood a braided mesh of paths, we would need the intermediate nodes to suppress incoming packets if a node has more incoming edges than outgoing edges, as is the case for node a_3 in Fig. 13. Similarly, if a node has more outgoing than incoming edges, it would need to create copies of a received packet and implement multicasting.

Flooding favors reliability over energy efficiency: it sends each message over multiple paths, hence, increases the probability of the message reaching the sink in case of node and channel failures; yet, when no such failures occur, flooding uses on the order of m times the number of necessary transmissions to get the message to the sink (assuming m paths are being used). Moreover, when multiple sources send out messages at the same time, flooding can result in congestion, i.e., collisions and queue build-up.

So, flooding is an appropriate choice for applications in which sensors send messages to the sink infrequently and, when they do, reliability is more important than energy efficiency.

4.2.2 Independent Transmissions

In this approach, sources use each path to send a different message to the sink.

This approach favors information rate over reliability: each source can send multiple messages to the sink at the same time; yet, messages sent over paths with node or channel failures may not be delivered. In practice, this approach is not energy-efficient either, in the sense that some messages are sent over more-than-minimum-cost paths. Moreover, similarly to the previous case, when multiple sources send out messages at the same time, it can result in congestion.

Thus, this approach is suitable for applications where sensors send out messages infrequently and, when they do, speed of dissemination is of critical importance. Such an application would be for example, if a sensor wants to send an image of an intruder to the sink; in such a situation, the sensor needs to transmit multiple packets to the sink as fast as possible.

4.2.3 Erasure Coding

Consider a configuration where a source is connected to the sink through m disjoint paths; sending the same message over m paths corresponds to a repetition code of rate $1/m$; sending a different message over each path corresponds to a rate-1 code. Between these two extremes, the source can use an erasure code of rate k/m: the source sends m coded packets; if the sink receives any k out of the m coded packets, it can retrieve the original information.

This approach enables a trade-off between reliability and speed of dissemination. However, similar to the previous case, it is not energy-efficient, as it uses suboptimal paths; moreover, similar to both previous cases, it can cause congestion.

4.2.4 Path-Selective Routing

In this approach, each message is routed along a single path from the corresponding source to the sink. The path is determined dynamically in the following way: each node that receives the message selects the best next-hop to the sink according to some local criterion such as:

- what is currently the minimum-cost next hop towards the sink;
- what is currently the minimum-cost path towards the sink;
- which is the next node that has received less traffic, and
- which is the next node that has most remaining energy.

This approach is well suited for braided-mesh topologies, where each node has multiple choices on how to forward a packet to the sink [5].

4.3 Room for Improvement

Multipath routing gracefully extends routing over trees to routing over larger topologies. There is, however, room for improvement regarding the following issues:

- *Control traffic*. Constructing and maintaining a larger topology requires (significantly) more control traffic. There is evidence that maintaining more than two paths per sensor (or more than two incoming and outgoing links per sensor node) requires control traffic that outweighs the benefits of multipath, as the required transmissions for control information consume a significant portion of the network resources (energy, wireless bandwidth, processing time) [6]. This problem becomes more pronounced when the network topology changes fast – in fact, for mobile sensor networks, constructing and maintaining multiple paths becomes practically infeasible.
- *Algorithmic complexity*. Identifying an optimal multipath topology is a computationally hard problem, even in the case where all topological information (channel quality between all pairs of nodes) is available at all nodes. Using suboptimal topologies can reduce the potential benefits of multipath.
- *Broadcasting*. Existing proposals do not exploit the inherent broadcasting capability of the wireless medium. For instance, consider an approach where each node that overhears a packet forwards it to the sink – even though it was not the packet's intended receiver. Such an "opportunistic" approach has the potential to increase reliability (because each packet is forwarded through multiple paths), but also waste network resources, including battery life (when there are no node/channel failures). In networks with tens or hundreds of nodes, controlling the number of redundant packets in the network is practically infeasible.

Ideally, we would like to have a multipath collection protocol that requires insignificant control traffic (compared to the actual data traffic), is transparent to the underlying topology changes, and exploits broadcasting. New ideas that have recently emerged from the area of network coding hold the potential to help towards this direction; we will next review the basic ideas in network coding and discuss how they fit in the context of sensor networks.

5 What Is Network Coding

Network coding is a new area that promises to revolutionize the way we treat information in a network, and have a deep impact in all network functionalities, such as routing, network storage, and network design [7–11]. The novel paradigm in network operation that network coding brings is that, instead of having individual source packets traversing a network, we instead have combinations of packets, each bringing some type of "evidence" about the source packets. These packet combinations are created throughout the network: we allow intermediate network nodes to

process their incoming information packets, and in particular, combine them to create new packets. A receiver collecting a sufficient number of such combined packets can use them to retrieve the original information sent by the sources. The area of network coding is centered around the application of this basic idea, of dealing with "evidence" instead of individual packets. The following "classical" example in the network coding literature illustrates the potential benefits over a wireless medium.

Example 4 Consider a wireless ad-hoc network, where devices A and C would like to exchange the binary files x_1 and x_2 using device B as a relay. We assume that time is slotted, and that a device can either transmit or receive a file during a time slot (half-duplex communication). Figure 14 depicts on the left the standard approach: nodes A and C send their files to the relay B, who in turn forwards each file to the corresponding destination.

The network coding approach takes advantage of the natural capability of wireless channels for broadcasting to give benefits in terms of resource utilization, as illustrated in Fig. 14.

In particular, node C receives both files x_1 and x_2, and bit-wise `xors` them to create the file $x_1 + x_2$, which it then broadcasts to both receivers using a common transmission. Node A has x_1 and can thus decode x_2. Node C has x_2 and can thus decode x_1.

This approach offers benefits in terms of energy efficiency (node B transmits once instead of twice), delay (the transmission is concluded after three instead of four time slots), wireless bandwidth (the wireless channel is occupied for a smaller amount of time) and interference (if there are other wireless nodes attempting to communicate in the neighborhood). The benefits in the previous example arise from that broadcast transmissions are made maximally useful to all their receivers.

Note that $x_1 + x_2$ is nothing but some type of binning or hashing for the pair (x_1, x_2) that the relay needs to transmit. Binning is not a new idea in wireless communications. The new element is that we can efficiently implement such ideas in practice, using simple algebraic operations.

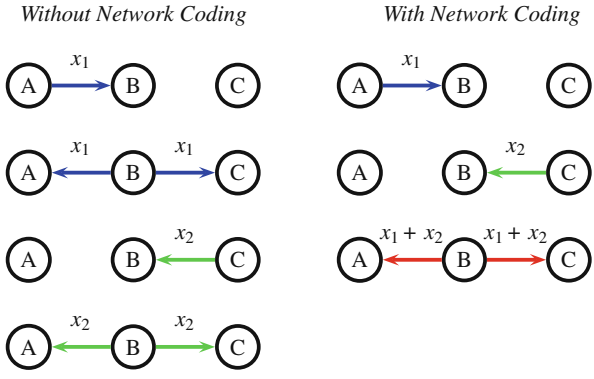

Fig. 14 Nodes A and B exchange information via relay B. The network coding approach uses one broadcast transmission less

5.1 Network Coding in Practice

In Example 3, we implicitly assumed that each node performs fixed encoding operations. The receivers know these operations, and use this knowledge to decode. In particular, nodes A and B know in advance that they will receive the linear combination $x_1 + x_2$. In a practical network, where the network structure, delays and synchronization varies, the selection and knowledge of the linear combination coefficients needs to be distributed in a decentralized manner.

Fortunately, three ideas, that appeared successively in time, give us an elegant and flexible way to perform network coding in a completely decentralized manner. These are:

1. Randomly chose the linear combinations at each network node [12].
2. Append "coding vectors" at the header of each packet to allow the receivers to decode without need of synchronization [13].
3. Use subspace coding to achieve the same goal as in (2) more efficiently [14].

The first idea determines what intermediate nodes in the network do. The second and third offer two alternative approaches for the encoding of the data at the sources and corresponding decoding at the receivers.

5.2 Randomized Network Coding

Assume we have n source packets $\{x_1, \ldots, x_n\}$ that contain symbols over a field $\mathbb{F}_q$ and we want to convey them to multiple destinations over a network using network coding. Throughout the network, intermediate nodes perform linear combining of the source packets. Thus, a destination receives combinations of the form

$$c_1 x_1 + c_2 x_2 + \cdots + c_n x_n,$$

where $c_i \in \mathbb{F}_q$. In the network coding literature, the vector of coefficients

$$c = [c_1,\ c_2,\ \ldots,\ c_n]$$

is called a *coding vector*. Each destination can retrieve the data, if it receives n linearly independent combinations of the source packets, or, n linearly independent coding vectors. For example, let $\{\rho_i\}$ be the combined packets a destination collects, we can write in a matrix form:

$$\begin{bmatrix} \rho_1 \\ \rho_2 \\ \vdots \\ \rho_n \end{bmatrix} = \underbrace{\begin{bmatrix} c_{11} & c_{21} & \ldots & c_{n1} \\ c_{12} & c_{22} & \ldots & c_{n2} \\ & \cdots & & \\ c_{1n} & c_{2n} & \ldots & c_{nn} \end{bmatrix}}_{\mathbf{A}} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}. \tag{1}$$

If the linear combinations are independent, i.e., matrix $\mathbf{A}$ is full rank, we can solve the above equations and retrieve the source packets. The task of network code design amounts to deciding what linear combinations to form throughout the network so that each receiver gets a full rank set of equations.

Randomized network coding is based on the simple idea that, for a field size q large enough, there exist so many valid solutions, that even random choices of the coefficients allow us to find a valid solution with high probability. Thus we can simply ask each intermediate node in the network to create and send uniform at random linear combinations of the packets it has received. The associated probability of error can be made arbitrarily small by selecting a suitably large alphabet size [12]. For example, if we could choose the coefficients $\{c_{ij}\}$ of matrix $\mathbf{A}$ in (1) uniformly at random, the matrix $\mathbf{A}$ would be full rank with probability at least $(1 - \frac{1}{q})^n$. In practice, simulation results indicate that even for small field sizes (for example, using $m = 8$ bits per symbol, i.e., $q = 2^8$) the probability of error becomes negligible [15].

Randomized network coding requires no centralized or local information, is scalable and yields to a very simple implementation. Thus, it is very well suited to a number of practical applications, such as sensor networks and more generally dynamically changing networks.

5.2.1 Generations and Coding Vectors

The next question to answer is, even if we randomly select what linear combinations to perform, how do we convey to the destinations what are the linear combinations they have received so that they can decode. Moreover, in a network where information gets generated at a constant rate, we need to decide what packets to combine and how often do we decode. To achieve these, we cannot rely on synchronization, since packets are subject to random delays, may get dropped, and follow different routes.

The approach in [13] first groups the packets into *generations*. Packets are combined only with other packets in the same generation. A generation number is appended to the packet headers to make this possible (one byte is sufficient for this purpose). The size of a generation can be thought of as the number of source packets n in synchronized networks: it determines the size of matrices the receivers need to invert to decode the information. Since inverting an $n \times n$ matrix requires $\mathcal{O}(n^3)$ operations, and also since waiting to collect n packets affects the delay, it is desirable to keep the generation size small. On the other hand, the size of the generation affects how well packets are "mixed", and thus it is desirable to have a fairly large generation size. Indeed, if we use a large number of small-size generations, intermediate nodes may receive packets destined to the same receivers but belonging to different generations. Characterizing this trade-off is an open research problem.

As a second step, the approach in [13] appends within *each* packet header a vector of length n that describes which linear combination of the source packets

$\{x_1, \ldots, x_n\}$ it contains. These vectors are what we called coding vectors. The encoded data is called the *information vector*. For example, the coding vector $\mathbf{e}_i = (0, \ldots, 0, 1, 0, \ldots 0)$, where the 1 is at the ith position, means that the information vector is equal to x_i (i.e., is not encoded). A packet that contains the linear combination $\rho = c_1 x_1 + c_2 x_2 + \cdots + c_n x_n$ has the coding vector $(c_1, \ldots, c_n)$ and the information vector ρ.

The coding vectors are updated locally at each node that performs linear combining, to reflect the new linear combination of the source packets that the new packet carries. For example, if a node receives two packets with coding vectors $\mathbf{e}_i = (0, \ldots, 0, 1, 0, \ldots 0)$ and $(c_1, \ldots, c_n)$, with corresponding information vectors x_i and ρ, it can create the new information vector $\alpha x_i + \rho$ for some value $\alpha \in \mathbb{F}_q$. To send this new information vector, it will use the coding vector $(c_1, \ldots c_{i-1}, c_i + \alpha, c_{i+1}, \ldots, c_n)$. Combining can occur recursively and several times inside the network.

Each receiver examines the coding vectors of the packets it receives, to learn what are the linear combinations it has received. In particular, the coding vectors it receives are nothing but the rows of the matrix $\mathbf{A}$ in (1) that determine the linear equations it needs to solve.

Appending coding vectors to packets incurs an additional overhead. For example, for a packet that contains 1400 bytes, where every byte is treated as a symbol over $\mathbb{F}_{2^8}$, if we have $h = 50$ sources, then the overhead is approximately $50/1400 \approx 3.6\%$.

5.2.2 Subspace Coding

The approach based on appending coding vectors in order to be able to decode at the receiver is well suited for large packets where the overhead is small. In wireless sensor networks, and generally, wireless networks, the situation is quite opposite: it is quite often the case that packets consist of a few bits. In such cases, using coding vectors can add a significant overhead.

A new approach recently proposed in [14, 16] promises to be helpful in the case of very short packet lengths. This approach is again designed to work with use of randomized network coding, and is based on using subspaces as "codewords" to convey the information from the sources to the receivers. For simplicity we will here consider a single source transmitting n independent packets to receivers, but the same approach can easily be extended to multiple sources [17]. We note that recent work [18–22] has established that subspace coding only offers benefits as compared to the coding vectors approach for very short block lengths.

Consider a source that would like to convey n independent source packets to receivers over a network that employs randomized network coding. Assume that each packet has length λ over $\mathbb{F}_q$. The n packets can take in total $M = q^{n\lambda}$ values. Thus the source, for each set of packets, has one of these values to convey.

The source can achieve this as follows. First, it selects to operate over an nL dimensional vector space V over $\mathbb{F}_q$, i.e., a vector space, where vectors have length

nL and have elements in $\mathbb{F}_q$. A basis of this space consists of nL linearly independent vectors. For example, the space $\mathbb{F}_2^3$ has the basis

$$\{\mathbf{e}_1 = [1\ 0\ 0], \quad \mathbf{e}_2 = [0\ 1\ 0], \quad \mathbf{e}_3 = [0\ 0\ 1]\}.$$

A subspace π is a subset of the vector space V that is a vector space itself. We can think of subspaces as "planes" that contain the origin. For example, the space $\mathbb{F}_2^3$ contains 7 two-dimensional subspaces. One such subspace is $\pi_1 = <\mathbf{e}_1, \mathbf{e}_2>$. Another is $\pi_2 = <\mathbf{e}_2 + \mathbf{e}_3, \mathbf{e}_1>$. It also contains 7 one-dimensional subspaces, one corresponding to each non-zero vector. Moreover, the subspace (plane) $\pi_1 = <\mathbf{e}_1, \mathbf{e}_2>$ contains the three "line" (one-dimensional) sub-subspaces $\pi_3 = <\mathbf{e}_1>$, $\pi_4 = <\mathbf{e}_2>$, $\pi_5 = <\mathbf{e}_1 + \mathbf{e}_2>$. Therefore, we can define subspaces of lower dimension as sub-spaces of higher dimensional subspaces. In the above example, we can see that $\pi_3 \subset \pi_1 \subset \mathbb{F}_2^3 = V$. We say that two subspaces are *distinct* if they differ in at least one dimension. For example, $\pi_1 = <\mathbf{e}_1, \mathbf{e}_2>$ and $\pi_2 = <\mathbf{e}_2 + \mathbf{e}_3, \mathbf{e}_1>$ are distinct.

The source selects a codebook of M distinct subspaces, and each set of n packets is mapped to a different such subspace. The receivers learn this codebook. To convey the value of the source packets, the source needs to convey what is the particular subspace these packets are mapped to. To do so, it inserts in the network a set of basis vectors (packets) that span the subspace. Assume for example it sends the vectors $\{b_1, \ldots, b_k\}$ that span a subspace π. The critical observation is that, the mixing through randomized network coding intermediate nodes perform, preserves the subspaces. Indeed, linear operations, no matter what these operations are, can only create vectors that are in the span of the basis $\{b_1, \ldots, b_k\}$ and thus within π. As a result, every node that receives k linearly independent vectors will be able to identify which is the subspace π that the source has sent. The source has then transmitted information through the choice of the subspace that it sends. This property makes the use of subspaces for encoding robust to the topology of the network and to arbitrary linear operations performed at the intermediate nodes.

6 Network Coding for Sensor Networks

Network-coding ideas and techniques have already been successfully applied in the context of wireless mesh and/or ad-hoc networks (see [23–26] for some examples). A natural question to ask is, are sensor networks any different? Apart from the fact that both types of networks are wireless, they differ in almost everything else that matters when designing a network protocol: topology, traffic patterns, performance metrics. For example, in wireless mesh/ad-hoc networks, network coding offers throughput benefits when certain multicast or concurrent-unicast traffic patterns occur, such as the well known pattern in Fig. 14. These patterns do not naturally occur in sensor networks, which typically carry many-to-one traffic from the sensors to the sink; for this traffic pattern, network coding does not offer throughput benefits. In many sensor-network applications, throughput is not even relevant

as a metric – as discussed, the two predominant metrics are energy efficiency and reliability.

There is evidence that network coding can help achieve a better energy efficiency/reliability trade-off than traditional single-path or multipath collection protocols. Intuitively, by allowing nodes to mix incoming packets, network coding introduces each piece of information into multiple paths – i.e., network coding offers a new way to implement multipath communication. However, before we examine the potential benefits, we need to solve a crucial practical problem: as the following examples illustrate, in a sensor network where a large number of nodes communicate short messages to the sink, it does not make sense to use neither coding vectors nor subspace coding.

Example 5 Consider a sensor network consisting of 100 sensors, where each sensor periodically communicates a 1-byte message to the sink. Assume that we want to implement network coding using coding vectors and operations over a field of size $q = 2^4$. In this case, we would need 50 bytes of coding-vector data *per packet* to convey just 1 byte of information. Such overhead rules out coding vectors as a practical design option.

Example 6 Subspace coding offers increased efficiency as it removes the need for coding vectors. However, designing subspace code for the case where the sources are not collocated is challenging. Consider for example the case where two sensor nodes use codebooks consist of subspaces of a vector space $\mathbb{F}_q^\ell$, i.e.,

$$\mathcal{C}_i = \{\pi_j^{(i)} : 1 \leq j \leq |\mathcal{M}_i|\}, \quad i = 1, \ldots, n.$$

To transmit information to the sink, source i maps a measured value to one such subspace π and inserts in the network d vectors that span π. In relaying information towards the sink, the sensor linearly combines all packets it has received (including that generated by itself) and transmits the combined packet to the next relays towards to the sink. As a result, the sink will observe vectors from the union of subspaces inserted by all the sources. In particular, if source i inserts the subspace π_i, the sink will observe vectors from the subspace $\pi_1 + \pi_2 + \cdots + \pi_n$. Using the knowledge of the codebooks $\{\mathcal{C}_i\}$, it needs to decode the sensor data.

To be able to correctly decode at the sink, we need to ensure that every combination of sensor data results in a *distinct* union subspace. Assume for simplicity we have two source nodes, S_1 using the codebook $\mathcal{C}_1 = \{\pi_1,\ \pi_2,\ \pi_3\}$, while S_2 the codebook $\mathcal{C}_2 = \{\pi_4,\ \pi_5,\ \pi_6\}$. Table 1 summarizes all outcomes. For this code to be identifiable, we want all (or some) entries in Table 1 to correspond to distinct subspaces. For example, $\pi_1 + \pi_4$ should be a distinct subspace from $\pi_2 + \pi_5$.

This problem is hard to solve even for the case of two sources, and a very small codebook (in our example each node transmits only 3 values). Designing such a code for 100 sources would be an admirable feat, let alone designing a code that can also be efficiently decoded.

Table 1 Coding for two sources

$\mathcal{C}_2/\mathcal{C}_1$	π_1	π_2	π_3
π_4	$\pi_1+\pi_4$	$\pi_2+\pi_4$	$\pi_3+\pi_4$
π_5	$\pi_1+\pi_5$	$\pi_2+\pi_5$	$\pi_3+\pi_5$
π_6	$\pi_1+\pi_6$	$\pi_2+\pi_6$	$\pi_3+\pi_6$

6.1 Code Design

We now describe a network-coding approach for sensor networks. The main intuition in this approach is to attempt to reduce the length of the coding vectors, by restricting the freedom in the coding operations that intermediate network nodes have. Indeed, the classic design of coding vectors, allows potentially *all* source packets to get combined together; our approach is to employ coding vectors that allow at most m packets get combined [18]. This is motivated through three observations:

1. Coding vectors allow us to decode the data even if all the sources' packets get mixed. However, packets get combined only if their paths to the sink overlap. For a network symmetrically deployed around a sink, it is unlikely that all paths will overlap.
2. Not all sensors may be actively measuring during every round. For example, when sensing anomalies, we expect a small fraction of all potential sources to send information.
3. We can artificially restrict the number of sources that get combined, by appending to each packet a few bits to count the number of combined packets it contains.

Our design problem can now be stated as follows. Given n sources, the sink is going to observe packets that contain linear combinations of *at most* m sources. We want to design codes that allow us, by receiving each combined vector, to determine which linear combination of the source packets it contains. Our goal is to utilize vectors of length ℓ that is much smaller than the number of sensor nodes n.

Our construction utilizes properties of erasure correction codes, and proceeds as follows. Select a linear code of length n, minimum distance $2m+1$, and redundancy ℓ, with ℓ as small as possible. Consider the $\ell \times n$ parity check matrix $\mathbb{H}$. Assign to each source as coding vector the 1-dimensional subspace spanned by one column of $\mathbb{H}$. The source, includes this column in front of the data packet it sends, exactly in the same way as in the case of the usual coding vectors.

Intermediate nodes combine up to m source packets to create a coded packet, using randomly selected coefficients. The sink, when receiving each encoded packet can determine by examining the coding vectors:

- which columns of the matrix $\mathbb{H}$ contribute to the resulting linearly combined vector.
- using this knowledge, the exact coefficients that have been used for the combining.

The reason our construction allows to perform these operations, stems from a well known property the columns of matrix $\mathbb{H}$ satisfy. Let $\mathcal{A}$ denote the set of these vectors. Then, if the code has minimum distance $2m+1$, *any set of* $2m$ *vectors in* $\mathcal{A}$ *are linearly independent* [27]. This directly implies the following properties. For vectors in $\mathcal{A}$:

(P1) Any set of $0 < \beta < 2m$ vectors span a distinct β-dimensional subspace.
(P2) The distance between two subspaces π_1 and π_2, where each subspace is spanned by a different set of $0 < \beta < 2m$ vectors, equals $\min\{\beta, 2m-\beta\}$.

From property (P1) we see that the sink receives distinct subspace for *every* distinct set of m sources, and therefore is able to decode the identities and the information. Thus, even though there are $n = |\mathcal{A}|$ possible sources, we do not need to use vectors of length proportional to n, but instead, of length $\ell \geq 2m$.

The following example illustrates this procedure.

Example 7 Consider a code with minimum distance $2m+1 = 5$ and a parity check matrix $\mathbf{H}$ of dimension $\ell \times n$ with columns $h_1, \ldots, h_n$:

$$\mathbf{H} = [h_1 \; h_2 \; \ldots \; h_n] \tag{2}$$

Each source appends a different h_i vector in the header of its information packet, and sends the packet through the network where intermediate nodes combine up to two packets. The sink receives n packets, and extracts from their header n vectors of the form $y_k = \alpha_{k1} h_{k1} + \alpha_{k2} h_{k2}$, for $k = 1, \ldots n$. For each vector y_k, the unknowns are the indices $k1$ and $k2$, i.e., which column vectors are combined, as well as the coefficients α_{k1} and α_{k2}. From property (P1), the sink can uniquely determine the column vectors h_{k1} and h_{k2} in whose span lies y_k. Using this knowledge, it can then uniquely identify α_{k1} and α_{k2}. This implies that the k-th packet received at the sink, contains the linear combination of the packets send by the sources $k1$ and $k2$ with coefficients α_{k1} and α_{k2}.

Thus to decode, the sink can create an $n \times n$ transfer matrix, where row k will contain exactly two nonzero entries: α_{k1} at position $k1$ and α_{k2} at position $k2$, and then solve the system a system of linear equations to retrieve the data.

In order to quantify our savings, we need to examine how ℓ scales with m. This is related to a well-studied problem in coding theory, namely, for a given code length $n \triangleq |\mathcal{A}|$, and a given minimum distance $2m+1$, what are upper and lower bounds on the number of codewords $A(n, m)$ this code can have [27]. Using the Gilbert-Varshamov lower bound and the sphere packing upper bound [27], it can be shown that for large values of n,

$$nH_2\left(\frac{2m+1}{2n}\right) \leq \ell \leq nH_2\left(\frac{2m+1}{n}\right), \tag{3}$$

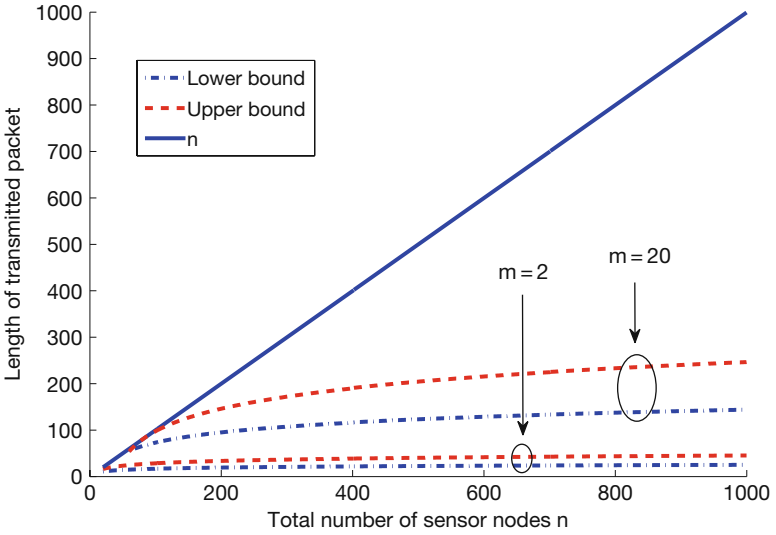

Fig. 15 Bounds on the length ℓ of the coding vectors when $m = 2$ and $m = 20$ sources get combined, as a function of the number of sensor nodes n

where H_2 is the binary entropy function, namely, $H_2(p) = -p \log p - (1-p) \log(1-p)$. Figure 15 plots the bounds in (3) as a function of n, for $m = 2$ and for $m = 20$. We conclude that our proposed code design results in using a fraction of the length n, that goes to zero as the ratio $\frac{2m+1}{n}$ goes to zero.

Example 8 Using a table of the best codes known [27] we can see for example that, there exist binary linear codes of length $n = 512$ with redundancy $\ell = 18$ and minimum distance $2m + 1 = 5$. Thus in a sensor network with 512 nodes if at most $m = 2$ source vectors get combined, we need to use vectors of length $\ell = 18$.

Example 9 An alternative approach to using shortened coding vectors, is to use smaller generations. In particular, we can assume that we divide the source nodes into groups, where each group contains m sources, and allow only the packets that originate from the same set of sources to be combined. A special counter attached to the header of the packet can specify the generation in which the packet belongs. This approach differs from our proposed approach in that, it may happen for a node to have multiple uncoded packets, but not be able to code them together as they belong in different generations.

6.2 Opportunistic Broadcasting with Network Coding

The idea is the following: when a node sends out a packet, apart from the intended receiver, some of the node's neighbors opportunistically overhear and receive the transmitted packet; each node forwards towards the sink a mix of the packets it

receives and the ones it opportunistically overhears. Thus, each piece of information may appear, linearly combined, in a large number of packets across the network. By propagating multiple packets that contain different linear combinations of the original data, we can create multiple opportunities for decoding the original data and achieve increased robustness to path failures.

We now outline a collection protocol that relies on this idea: it constructs a tree and implements opportunistic broadcasting on top. We first describe how the tree is used, then how it is constructed.

Consider a sensor network where nodes have formed a tree rooted at the sink. The network operates in rounds. During each round, each node stores the packets it receives from its children as well as any packets it opportunistically overhears from its other neighbors. It starts transmitting as soon as one of the following conditions is met:

- It has received packets from all its children.
- It has received a packet from at least one child and a time window has expired.
- It has overheard new information that exceeds a predetermined threshold.

A node continues to transmit until its parent signals that it has successfully received one of its packets, or an upper limit on the number of per-node transmissions is reached. Each transmission carries a different linear combination of the node's received/overheard packets – such that, if a neighbor happens to successfully overhear more than one transmissions, each one will bring new information to it.

Example 10 Consider the tree shown in Fig. 5 and assume we use it as described above. Recall that, in this tree, a node's parent is not necessarily its minimum-cost neighbor (it is the next hop on the node's minimum-cost path to the sink). For example, consider node a_{10}; to reach its parent, a_8, it must transmit on average twice; however, it can reach a_7 by transmitting on average once. Hence, if node a_{10} transmits until a_8 acknowledges receipt of a packet, it is likely to transmit twice, and a_7 is likely to successfully overhear both transmissions, i.e., collect two different linear combinations of the packets previously received and overheard by a_{10}. Similarly, if node a_2 transmits until its parent a_3 acknowledges receipt of a packet, its transmissions are likely to be successfully overheard by the sink.

We now discuss how to construct the tree. Assume that we want to ensure that each transmission be received by at least two neighbors of the transmitting node. We can implement this requirement in two ways: (i) *Hard*, where we explicitly mandate that two neighbors acknowledge reception. This effectively amounts to constructing two trees, which, as discussed in Example 2, is computationally hard. (ii) *Soft*, where we require that on average at least two neighbors receive each transmitted packet. We choose the latter and implement it as follows.

Consider a node that has multiple neighbors. In the Collection Tree Protocol (described in Sect. 3), each node chooses its parent so as to minimize the overall cost of communicating to the sink. We now revise this as follows: each node chooses its parent so as to minimize the overall cost of communicating to the sink, subject to the following *redundancy constraint*: a node is not allowed to choose as

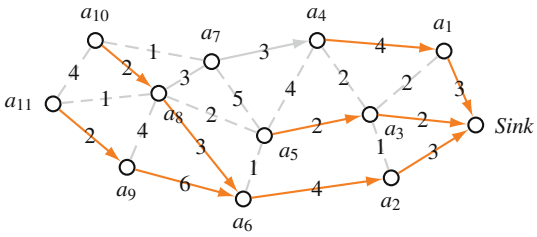

Fig. 16 A tree that satisfies the redundancy constraint: each node is connected to the tree through a link that is not its minimum-cost link, with the exception of the nodes that are connected directly to the sink

its parent its lowest-cost neighbor, unless it has no other choice. This protocol can be implemented as easily as CTP, by simply adding the redundancy constraint.

When a node a transmits a packet and its parent b successfully receives it, it is likely that so do a's neighbors that are connected to it through better links than b. The redundancy constraint ensures that, whenever possible, at least one such neighbor exists for each node. The following example illustrates this property.

Example 11 Figure 16 depicts a tree that satisfies the redundancy constraint. In this tree, node a_4 chooses node a_1 as its parent, whereas its lowest-cost neighbor is a_3. On average, the packets received by a_1 are also received by a_3. The cost of this tree is 34. The cost of the tree constructed by CTP (shown in Fig. 5) was 25. So, we ensure that on average each packet is received by at least two neighbors at the cost of increasing transmissions by approximately 35%.

7 Conclusions

In this chapter, we looked at the problem of designing a protocol that performs the simplest possible sensor-network task: communicate measurements from the sensors to the sink, without any in-network information processing. We described protocols that balance the competing goals of energy efficiency and reliability, ranging from a simple tree-based protocol to multipath protocols, and discussed their basic benefits and drawbacks; we also outlined an emerging protocol based on network coding. We close by noting that a rigorous study would be necessary to evaluate the relative value of each approach and how well each is matched to specific application needs.

References

1. H. Dubois-Ferrière, L. Fabre, R. Meier, and P. Metrailler, "Tinynode: a comprehensive platform for wireless sensor network applications," in *Proceedings of the IEEE International Symposium on Information Processing in Sensor Networks (IPSN)*, April 2006.
2. N. Burri, P. von Rickenbach, and R. Wattenhofer, "Dozer: ultra-low power data gathering in sensor networks," in *Proceedings of the IEEE International Symposium on Information Processing in Sensor Networks (IPSN)*, April 2007.

3. R. Fonseca, O. Gnawali, K. Jamieson, S. Kim, P. Levis, and A. Woo, "The collection tree protocol," http://www.tinyos.net/tinyos-2.x/doc/html/tep123.html, 2006.
4. Z. Li and B. Li, "Improving throughput in multihop wireless networks," *IEEE Transactions on Vehicular Technology*, vol. 55, pp. 762–773, 2006.
5. S. De, C. Qiao, and H. Wu, "Meshed multipath routing with selective forwarding: an efficient strategy in wireless sensor networks," *Computer Networks*, vol. 43, no. 4, pp. 481–497, 2003.
6. D. Ganesan, R. Govindan, S. Shenker, and D. Estrin, "Highly-resilient, energy-efficient multipath routing for wireless sensor networks," in *Proceedings of the ACM Symposium on Mobile Ad Hoc Networking and Computing (MOBIHOC)*, October 2001.
7. R. W. Yeung and N. Cai, "Network error correction, i: basic concepts and upper bounds," *Communications in Information and Systems (CIS)*, vol. 6, pp. 19–35, 2006.
8. N. Cai and R. W. Yeung, "Network error correction, ii: lower bounds," *Communications in Information and Systems (CIS)*, vol. 6, pp. 37–54, 2006.
9. C. Fragouli and E. Soljanin, *Network Coding: Fundamentals*, ser. Foundations and Trends in Networking. Now Publishers, Delft, 2007, vol. 2, pp. 1–133.
10. C. Fragouli and E. Soljanin, *Network Coding: Applications*, ser. Foundations and Trends in Networking. Now Publishers, Delft, 2008, vol. 2, pp. 135–269.
11. T. Ho and D. S. Lun, *Network Coding: An Introduction*. Cambridge University Press, Cambridge, UK, 2008.
12. T. Ho, M. Médard, R. Koetter, D. R. Karger, M. Effros, J. Shi, and B. Leong, "A random linear network coding approach to multicast," *IEEE Transactions on Information Theory*, vol. 52, pp. 4413–4430, 2006.
13. P. A. Chou, Y. Wu, and K. Jain, "Practical network coding," in *Proceedings of the Allerton*, October 2003.
14. R. Koetter and F. Kschischang, "Coding for errors and erasures in random network coding," in *Proceedings of the IEEE International Symposium on Information Theory*, June 2007.
15. Y. Wu, P. A. Chou, and K. Jain, "A comparison of network coding and tree packing," in *Proceedings of the IEEE International Symposium on Information Theory*, June 2004.
16. D. Silva and F. R. Kschischang, "Using rank-metric codes for error correction in random network coding," in *Proceedings of the IEEE International Symposium on Information Theory*, June 2007.
17. M. Jafari Siavoshani, C. Fragouli, and S. Diggavi, "Non-coherent network coding for multiple sources," in *Proceedings of IEEE International Symposium on Information Theory*, 2008.
18. M. Jafari Siavoshani, L. Keller, C. Fragouli, and K. Argyraki, "Compressed network coding vectors," in *Proceeding of the IEEE International Symposium on Information Theory*, June 2009.
19. M. Jafari Siavoshani, S. Mohajer, C. Fragouli, and S. Diggavi, "On the capacity of non-coherent network coding," in *Proceedings of the IEEE International Symposium on Information Theory*, June 2009.
20. S. Mohajer, M. Jafari Siavoshani, S. Diggavi, and C. Fragouli, "On the capacity of multisource non-coherent network coding," in *Proceeding of the IEEE Information Theory Workshop*, June 2009.
21. D. Silva, F. R. Kschischang, and R. Koetter, "Capacity of random network coding under a probabilistic error model," in *Proceeding of the 24th Biennial Symposium on Communications*, June 2008.
22. A. Montanari and R. Urbanke, "Coding for network coding," arXiv:cs.IN/0711.3935, 2007.
23. C. Fragouli, J. Widmer, and J.-Y. Le Boudec, "A network coding approach to energy efficient broadcasting: From theory to practice," in *Proceedings of the IEEE Conference on Computer Communications (INFOCOM)*, April 2006.
24. S. Katti, S. Gollakota, and D. Katabi, "Embracing wireless interference: analog network coding," in *Proceedings of the ACM SIGCOMM*, August 2007.
25. S. Katti, H. Rahul, W. Hu, D. Katabi, M. Medard, and J. Crowcroft, "Xors in the air: practical wireless network coding," in *Proceedings of the ACM SIGCOMM*, August 2006.

26. C. Gkantsidis, W. Hu, P. B. Key, B. Radunovic, P. Rodriguez, and S. Gheorghiu, "Multipath code casting for wireless mesh networks," in *Proceedings of the ACM CoNEXT*, December 2007.
27. F. J. MacWilliams and N. J. A. Sloane, *The theory of error correcting codes*, ser. North-Holland Mathematical Library. North Holland, Amsterdam, Netherlands, 1983.