Scalable Redundancy for Sensors-to-Sink Communication

Osameh M. Al-Kofahi Ahmed E. Kamal
Department of Electrical and Computer Engineering, Iowa State University
e-mails:{osameh, kamal}@iastate.edu

***Abstract*—In this paper, we present a new technique that uses deterministic binary network coding in a distributed manner to enhance the resiliency of Sensor-to-Base information flow against packet loss. First, we show how to use network coding to tolerate a single packet loss by combining the data units from k sensor nodes to produce $k+1$ combinations such that any k of them are solvable. After that, we extend the solution to tolerate multiple losses. Moreover, we study the coding efficiency issue and introduce the idea of relative indexing to reduce the coding coefficients overhead. To tolerate node or link failures, we introduce a simple routing protocol that can find maximally disjoint paths from the k sensor nodes to the base station. We study the relationship between the probability of successful recovery of all data units at the BS, and the number of sources protected together taking into consideration their hop distance from the BS. From this study we can decide on the appropriate number of sources to be protected together, so that the probability of successful recovery is higher than a certain threshold. Finally, we show through a simulation study that our approach is highly scalable and performs better as the network size increases.**

I. Introduction

Wireless Sensor Networks (WSNs) [1] are composed of a large number of sensing nodes, which are deployed in a certain area to monitor a phenomenon of interest. WSNs are usually deployed in harsh environments where packet loss and node failure is common, which in turn degrades the quality of the monitoring process. Survivability techniques are proposed to mitigate the effect of these problems, where in all these techniques the improved reliability comes at the price of increased redundancy [2]. Usually this is done by duplicating information on multiple paths, which either can be totally disjoint or braided [3], [4], [5]. Sufficient connectivity is a prerequisite for multipath routing, for example in [6], [7] sensor deployment and topology control algorithms were proposed to guarantee a k-connected network. In [8] the authors studied conditions that will enable multipath or braided routing in large wireless networks. They showed that if the shortest path between two terminal nodes is n hops then there exists (with high probability) $C \log n$ disjoint paths in a strip of width $a(C,p)\log n$ hops, where C is a constant and p is the availability of a wireless link.

In [4], information is sent on multiple paths to enhance the network reliability. A forwarding mesh that is composed of a set of interleaving paths was used in [3] to relay information from a source node to the base station (BS), where each data unit is given a budget that controls the degree of redundancy or the width of the mesh. On the other hand, in [5], the author introduced an efficient algorithm to give each node a set of node-disjoint paths to choose from in the case of a failure on the primary path. The concept of event-to-sink reliability in WSNs was introduced in [9], where a certain event is considered reported if the number of reports reaches a certain threshold. In [10] the authors proposed a recovery mechanism for WSNs with lossy links based on caching packets at selected nodes. The mechanism is called active caching because the caching decision is made per hop according to the accumulated success probability, which is piggybacked on every transmitted packet. If a packet is lost, a NACK is sent upstream until it reaches a node that has the lost packet in cache or until it reaches the source.

It was shown that the overhead from duplicating the data units can be reduced by using erasure codes in [11] and [12], where a packet is encoded into n smaller sub-packets such that only k of them, where $k < n$, are necessary to recover the original data unit. These sub-packets can be sent on the same path as in [11] or on node-disjoint paths as in [12]. In [13] rateless codes are used to reliably broadcast messages in a WSN. The problem in using rateless codes in general (and especially for broadcast) is that the sender cannot know when to stop sending redundant messages, unless it receives feedback from the receiver(s) to acknowledge that the message is received correctly. The paper utilizes Extreme Value Theorem (EVT) to estimate the largest number of required packets that enables all nodes to successfully recover the broadcast message, and thus, considerably reducing feedback from receivers.

In this paper, we introduce a new technique that will dramatically reduce the redundancy overhead compared with that in duplication-based multipath mechanisms, and yet still achieves the same level of survivability. We accomplish this by using network coding [14], [15]. In [16] we showed that network coding can be used to tolerate e failures in many-to-one flows, by having each set of n sources send at least $n+e$ combinations to the sink, such that any n of them are linearly independent. The authors in [17] used a coding scheme similar to that introduced by the author in [16] to protect against relay node failures and evaluated some QoS metrics for this protection scheme. It was shown that at high data rates (> 512 kbps) the delay for the coding-based scheme approaches the delay induced in the no protection case. The authors introduced a survivability mechanism in [18] that enables the max-flow between a pair of communicating nodes to be achieved by only protecting non-cutting links, network

This work was supported in part by the National Science Foundation under grants CNS-0626822, CNS-0626741, CNS-0721453 and ECS-0601570 and by a gift from Cisco Systems.

coding was utilized to maximize the number of protected non-cutting links. The work in this paper considers the many-to-one flow paradigm as in [16]. However, the scheme in [16] requires global information (in the first run at least) and assumes a fixed set of sources at a certain time. In contrast, in this work the scheme is fully distributed (each node operates using its local information), and there is no prior knowledge on the sources participating in the coding process. We start by showing how to use deterministic network coding in a distributed manner to protect the data units from k sources against a single packet loss, which is discussed in Section II. Our solution is extended in Section III to tolerate multiple packet losses. Moreover, we discuss some coding-related issues in Section IV, where we introduce the ideas of relative indexing and best effort decoding. A simple routing protocol is introduced in Section V, which can be used with our scheme to enhance the survivability of the information flow against node or link failures. In Section VI we study the relationship between the probability of successful recovery of all data units at the base station and the number of sources taking into account their hop distance from the base station. Simulation results are presented in Section VII. Finally, we conclude the paper in Section VIII.

A preliminary version of this paper was published by the authors in [19]. However, this paper makes two extensions to this work, which are the contents of Sections V and VI.

A. Comparison with previous work

Our scheme aims to reduce the redundancy overhead produced in duplication-based multipath routing schemes like [3], [4]. The simulations in Section VII show that our scheme succeeds at reducing the duplication overhead by 70%. The schemes in [11], [12], and [13] use erasure codes to send data from a single source to single/multiple destination(s) in Wireless Sensor Network (WSN). Our work differs in two aspects: first we use network coding instead of erasure codes, which should provide better performance even in a unicast connection as shown in [2]. Moreover, our coding scheme uses a binary field which simplifies the coding and decoding process even in the case of tolerating multiple losses. Second, unlike [11] and [12] that focus on unicast connections (or [13] that focuses on broadcast), our scheme is tailored for a many-to-one (convergcast) flow from a set of sensors to the base station, which is the dominant type of flow in a WSN.

In [16] we proposed a coding scheme for many-to-one flows in Wireless Mesh Networks (WMNs), where the network is structured and the nodes are less constrained. The scheme in [16] is centralized as mentioned above. This scheme was used and applied in [17] to tolerate relay node failures. The work in this paper is different in its nature since it targets a WSN, which is composed of highly constrained nodes that cannot waste energy to collect global information about the network. Therefore, the scheme in this paper is fully distributed, where each node can operate using its local information only.

II. Operation

We consider a dense and uniformly distributed WSN, in which packetes loss may occur. Assuming that at most one packet can be lost, our objective is to practically provide proactive protection to the information flow from the sensors to the BS using as few resources as possible. Before discussing the details of node operation, we need to clarify our assumptions and notations. Specifically, we assume the following:

1) We assume a wireless sensor network in which the sensor nodes can be organized into levels or rings around the base station, such that, the minimum hop count between the sensor nodes in ring i and the BS is i hops. An example is shown in Figure 1, where nodes d, e and f are in the first ring, nodes c and b are in the second ring and finally node a is in the third ring around the BS. We use the terms ring and level interchangeably throughout the paper.
2) We assume that there are $\mathcal{R}$ levels in the sensor network, and we denote the level of node u by l_u.
3) Routing a packet ensures its progress towards the sink in each transmission, i.e. a packet gets closer to the sink by one hop after each transmission.
4) Sensors generate data periodically and at the same rate.
5) The data units for all sensor nodes are equal in size, and the data unit for sensor node u is denoted by d_u.
6) Throughout the paper we use p to denote a certain packet, and we reserve d_p to only represent the information symbol carried in packet p.

Note that the first three assumptions are made to help in streamlining the discussion.

Assume that sensor node u has a data unit d_u that must be forwarded to the BS. Node u can send two copies of d_u to the sink to tolerate a single packet loss as a proactive alternative to retransmission. However, if there is a large number of sensor nodes that need to forward information to the sink, this solution is not efficient anymore. This is because 50% of the forwarded information is redundant, and thus, at least 50% of the network resources (bandwidth and energy) will be wasted to provide this redundancy.

Assume that node u has sent two copies of d_u to the sink. Naturally, this is done through multi-hop communication for each of these packets. Suppose that as the two packets are forwarded, $k-1$ of the forwarding nodes had data units of their own that also need to be sent to the BS. We show how to protect d_u and the $k-1$ other data units by forwarding only $k+1$ packets, through the use of network coding. In general, the nodes in a wireless sensor network can be divided into the following three types:

1) **Type 1.** A source with no data to relay.
2) **Type 2.** A source with data to relay.
3) **Type 3.** Just a relay with no data of its own.

To tolerate a single loss using network coding in a distributed manner, we need to specify the way a node operates given its local information. We define the following operation for each class of nodes:

1) **Type 1.** Assume node u only has its own data unit d_u, and has no packets from other sensors to relay. Then node u just sends two copies of d_u to be able to tolerate a single loss e.g, node a in Figure 1.
2) **Type 2.** Assume node u has its own data unit d_u, and in

addition has received another packet, say p, which needs to be relayed (e.g., node b in Figure 1). Then node u produces the following two packets:

a) **Packet 1.** Contains the bitwise XOR of d_u and d_p.

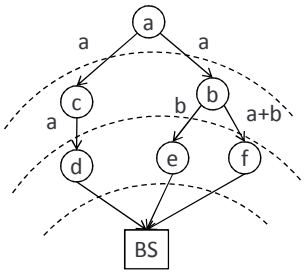


Fig. 1. Protecting data from two sources against a single packet loss

The type of a node changes according to its status, i.e., if node u is of Type 3 at a certain time instant t_i, it might become a Type 1 or Type 2 node at t_{i+1} if a data unit is generated locally. Now recall the scenario of node u and the $k-1$ other sources that forward the data units of u. In this scenario node u is the only node of Type 1, and the $k-1$ sources are of Type 2. Note also, that each of the $k-1$ sources increases the number of total packets by 1, which means that the total number of generated packets is $k+1$. Let us call this operation a *forwarding process initiated by node* u, or for short, an *F-process* initiated by a Type 1 node, u.

We now prove that by following the rules of operation, if k sources are involved in a certain F-process then they will produce $k+1$ packets such that any k of them carry a solvable set of combinations.

Claim 1. *If s is the k^{th} source to participate in an F-process initiated by node u, in which the number of involved sources until now is $k-1$ (including u), then by following the rules of operation described above, s will increase the number of packets to $k+1$, such that any k packets from them carry a solvable set of combinations.*

Proof. We prove this claim by induction. We need to show that if any $k-1$ from the current k combinations (produced by the $k-1$ sources) are solvable, then the participation of the k^{th} source will produce $k+1$ combinations such that any k of them are solvable.

The *basis step* is when the F-process is first initiated at node u. By following the rules of operation, node u will send two copies of d_u. Thus we have $k=1$ source, and $k+1=2$ combinations (trivial combinations in this case), such that any one (k) of them is solvable, which is obvious since each combination contains only one data unit, so if one packet is lost the other is sufficient to recover d_u.

To prove the *inductive step*, assume that $k-1$ sources have transmitted and any $k-1$ from the current k combinations are solvable. Node s can participate only if it is of Type 2, i.e., it received one packet (say p) from the current k packets and it has its own data unit d_s that must be sent to the BS. By following the rules, s will produce two packets: 1) a packet containing the XOR of d_s with d_p (note that the number of total packets is still k since p and d_u are merged into one packet), and 2) a packet containing d_s (i.e., the number of packets is increased by 1). We now need to show that losing either one of the newly created packets will leave us with k solvable combinations. If we lose the first packet, then the second packet will be sufficient to recover d_s, and from our assumptions the remaining $k-1$ combinations will be sufficient to recover the remaining $k-1$ data units. If we lose the second packet, then from our assumptions we can recover all the data units in d_p (and thus d_p itself) using the $k-1$ combinations other than $d_s \oplus d_p$, and then recover d_s by $d_p \oplus (d_s \oplus d_p)$. Finally, if we lose a packet other than those produced by s, we can recover d_s from the second packet produced by s, which leaves us with $k-1$ solvable combinations in $k-1$ unknowns. ■

A node of Type 1, will initiate a process that will involve a large number of nodes of the other two types. To distinguish the packets belonging to the process initiated by a node of Type 1, we add a new field in all the packets generated by this process. Let us call this field the *"initiator ID"* or *IID* for short. A node of Type 1 will put its ID and a timestamp in this field, and a node of Type 2 (say u) that XORs d_p with its data unit will copy the IID from p and put it in both generated packets. If a Type 2 node receives 2 or more packets with the same IID it must not combine its data with more than one of them, since this may produce dependent combinations.

Let P_i^j be the set of packets leaving level i (i.e., generated by level i, or just forwarded by it) that belong to the process initiated by node j, then $|P_i^j|$ is equal to:

$$|P_i^j| = 1 + \sum_{k=i}^{l_j} S_k^j = 1 + N_i^{l_j} \tag{1}$$

where S_k^j is the number of source nodes (Type 2) in level k that are involved in the process initiated by j, and $N_i^{l_j}$ is the number of source nodes in the levels from i to l_j that are involved in the process initiated by node j. Of course, there is only one node of Type 1 in the process initiated by node j, and that is j itself.

There should be a limit on the number of combinations that can be produced by an F-process. Let h denote this limit. For now we choose h to be the min-node cut in the network under consideration, and we elaborate more on the selection of h later. In a sufficiently dense and uniformly distributed WSN, the minimum node cut is usually the number of one hop neighbors of the BS. Therefore, no more than $h-1$ sources can be involved in any process. To insure this, we add a new field in the packet format; Let us call it *"Number of Remaining Combinations"* or *"NRC"* for short. We now redefine the operation of Type 1 and Type 2 nodes as follows (the operation of Type 3 nodes do not change):

1) **Type 1.** Assume node u only has its own data unit d_u, and has no data units from other sensors to relay. Then node u just sends two copies of d_u, such that in both

packets the values of $IID = u$ and $NRC =$

2) **Type 2.** Assume node u has its own data un addition has received another packet, say p, to relay. The operation of u depends on the of p as follows:

   a) If $NRC(p) \geq 2$ node u produces the fc packets, such that, in both packets IID and $NRC = \lfloor NRC(p)/2 \rfloor$.

      i) **Packet 1.** Contains $d_u \oplus d_p$.
      ii) **Packet 2.** Contains only d_u.

   b) However, if $NRC(p) = 1$, node u ac of Type 3, i.e., just a relay.

Note that by taking the floor when updating the h is made equivalent to the largest power of 2 tha or equal to h. Note also how the resulting tree resembles a binary tree, where each leaf is a combination in our case. However, our tree is restricted in depth, where it can have at most $d = \log_2 h$ levels. Therefore, the maximum number of combinations (or leaves) is $2^d = 2^{\log_2 h} = h$, which is the case only when we have a complete binary tree of d levels. By this we insure that the maximum number of combinations is less than or equal to h and the maximum number of participating sources in less than or equal to $h - 1$. For example, consider the network in Figure 2 where $h = 7$. Node a is the initiator node (Type 1 node), by following the rules of operation a will send two packets carrying d_a to b and c with $IID = a$ and $NRC = \lfloor \frac{7}{2} \rfloor = 3$. Node c is a Type 3 node, i.e., it will relay only and will not change the value of the IID or the NRC. Node b is a Type 2 and according to the rules of operation it will produce two packets with $IID = a$ and $NRC = \lfloor \frac{3}{2} \rfloor = 1$. It is obvious that starting with $h = 7$ we can have at most 4 packets produced, which is equivalent to starting with $h = 4$ (i.e., the closest power of 2 less than 7). Therefore, in the remainder of this paper we will always assume that h is a power of 2.

It is worth mentioning that assigning the NRC value as described in the rules of operation is not optimal in terms of maximizing the number of participating sources in a certain F-process. For example in Figure 2, assume that similar to node c node d is also a Type 3 node and the packet forwarded from d will reach the BS on a path composed only of Type 3 nodes. However, assume that node f has a data unit of its own that it wants to combine with the packet received from b. It will not be able to participate since the NRC value in the received packet is 1. Therefore, if node a knew beforehand the source nodes that will relay its packets, it can set the NRC to 1 in the packet sent to c, and to 6 (which is equivalent to 4) in the packet sent to b. Nevertheless, since we assume that a node operates using only its local information, and the deployed network is dense and uniformly distributed, a realistic assumption that an initiator node can make is that both packets will pass through the same number of sources approximately, and thus divide h by 2. Using simulation, in Section VII we verify that using this assumption results in a performance that is not much worse than the optimal.

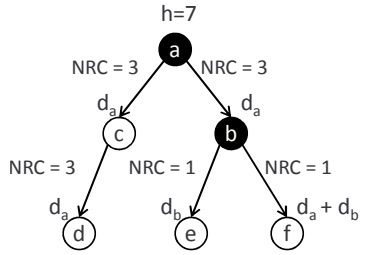


Fig. 2. NRC update for h=7

III. Tolerating Multiple Losses

The operation described in Section II proactively protects each generated data unit against a single loss. Suppose we want to protect each data unit against e losses. We can do this using the same operation described in Section II by allowing a Type 1 (Type 2) node, say u, to initiate (participate in) multiple F-processes for the same data unit. Assuming that at most e packets (i.e., combinations) can be lost, d_u must be inserted by u in at least $e+1$ combinations. Since each time u participates in a process two combinations are produced, node u must participate in (or initiate) $\frac{e+1}{2}$ F-processes. Therefore, if e is even (i.e., $e + 1$ is odd) either u participates in $\lceil \frac{e+1}{2} \rceil$ F-processes, or in $\lfloor \frac{e+1}{2} \rfloor$ processes and produces a packet containing d_u only with $NRC = 1$ to complete the $e + 1$ insertions ($NRC = 1$ to prevent other sources from combining their data with this combination).

Specifically, we assume that each node of Type 1 or Type 2 keeps a set of counters that are initialized to $e + 1$, each of which is associated with a certain data unit that is generated by the sensor node itself. Each counter keeps track of the remaining number of participations needed to provide protection against e failures for a certain data unit. A counter is decreased by 2 each time the node participates in (or initiates) a certain F-process until it reaches 0, where no further participations are needed. An example is shown in Fig.3, where we want to tolerate 3 packet losses for each of the data units, i.e., $e = 3$. There are three Type 1 nodes (A, B and C), and three Type 2 nodes (D, E and F), each of which has a single data unit to be sent. We only focus on the packets going through nodes D, E and F, where each of these nodes forward traffic from more than one F-process, and we assume that the paths for the remaining packets do no not intersect. For every data unit a counter is initialized to 4, i.e., each Type 1 node can *initiate* 2 F-processes and each Type 2 node can *participate* in 2 F-processes. For example consider node A, the two packets sent from A to nodes D and E represent one process initiated by A, and the remaining two packets represent the other process initiated by A after which the counter will be 0 for d_a. Nodes D and E receive packets from all the three Type 1 nodes, but since they can only participate in 2 processes, they choose to participate in the processes initiated by nodes A and B. Therefore, when nodes D and E receive the packet from C they act as Type 3 nodes and just forwards the packet as is. It should be pointed out that a node participating in a process refers to coding its own data with data from another node. Type 3 nodes are just relays, and they do not participate or change anything in the packets they receive. As shown in Figure 3,

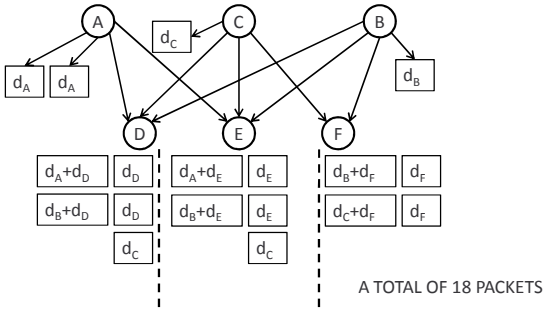


Fig. 3. Protecting against 3 losses

IV. Coding/Decoding Issues

The efficiency of network coding is an important issue that **arises** in most network coding-based applications. In this section we show that the overhead, which results from the need to send the coding vectors along with the combinations to the BS, can be significantly reduced by using relative indexing. Moreover, we introduce the idea of best effort decoding, which allows us to make use of the combinations received at the BS even if more than one packet was lost.

A. Relative Indexing for Efficient Encoding

For the sink to be able to decode the linear combinations, it needs to have the chosen coefficients or the coding vectors for each one of the combinations. In binary network coding for multicast connections, the length of the coding vector is determined by the minimum max-flow h between the single source and any of the terminals [14], where position i in the vector is reserved for d_i, i.e., if index i = 1 then d_i is present in the combination, and if index i = 0 then d_i is not. This is possible in multicast connections because all the data units originate from the same source, which means that this single source can assign for each data unit a unique index to identify it with. However, in our case the coding process is distributed, and thus the coding vector should be of size N, where N is the total number of nodes in the network, where each node is a possible source at some time. In a sensor network N can be very large, where it can be in the hundreds or even thousands of nodes. It is obvious that this is a waste of bandwidth since at most $h-1$ sources can participate in a certain F-process.

We now show how to enable the source nodes in an F-process to use an $(h-1)$-bit coding vector by relatively indexing the sources in that process. To do this, we use the NRC value in addition to a new control bit that we will add to the packet header, which we refer to as I. Assume that both packets generated by an initiator node A will be combined with data units from two other sources B and C. The question is if A is given index 1 in the $(h-1)$-bit coding vector, how can B and C (and any other Type 2 node in the process) choose distinct indices using the values of NRC and control bit I?

In an F-process there is only one node of Type 1, which will be always assigned index 1 (index 0 will not be used). This leaves us with (h-2) indices, which will be recursively divided into halves. Let us consider the pair of Type 2 nodes in the first coding level after A (i.e., B and C), where each of them receives a packet with $NRC = \frac{h}{2}$, but B receives a packet in which $I = 0$ and C receives a packet in which $I = 1$. We give B the first index in the first half, and C the first index in the second half of the remaining h-2 indices. Half of h-2 is $NRC - 1$ (since $h = 2 * NRC$), which means that relative to index 1 the first half begins at the next position **after** *position 1* (i.e., 1+(1)), and the second half begins **after** the next $NRC-1$ positions after *position 1* (i.e., 1+($NRC-1$)+1). This is shown in Figure 4, where h is 8, the data unit from node B will be given index $1+(1) = 2$, and the data unit from node C will be given the index $1+(NRC-1+1) = 1+NRC = 5$. In the second iteration the process continues but with reference to position $1 + (1)$ for the nodes descending from B, and with reference to position $1 + (NRC - 1) + 1$ for the nodes descending from C.

To generalize for other levels, let $X(p)$ be the coding vector of the combination carried in p, and let X_i be an $(h-1)$-bit vector with 1 in the i^{th} position and zeros otherwise. In addition, let $I(p)$ and $NRC(p)$ denote the value of bit I and the NRC field in p respectively. If a node u decides to participate in the F-process of some packet p, it calculates its index according to the following equation:

$$Index(u) = \max_{i:X_i \wedge X(p)=X_i} i + I(p) * (NRC(p) - 1) + 1 \quad (2)$$

where the values for $I(p)$ and $NRC(p)$ are set according to following operation (we only redefine the operation for Type 1 and Type 2 nodes):

1) **Type 1.** Assume node u has its own data unit d_u, and has no data units from other sensors to relay. Then node u sends two packets carrying d_u, such that IID = u and NRC = $\lfloor h/2 \rfloor$ in both packets. In addition to the following settings:
   - **Packet 1.** $I(Packet1) = 0$, $X(Packet1) = X_1$.
   - **Packet 2.** $I(Packet2) = 1$, $X(Packet2) = X_1$.
2) **Type 2.** Assume node u has its own data unit d_u and has received another packet p. The operation of u depends on the NRC value of p as follows:
   a) If $NRC(p) \geq 2$ node u calculates its index using equation 2. Then it produces the following two packets, such that, in both packets IID = $IID(p)$ and NRC = $\lfloor NRC(p)/2 \rfloor$.
      i) **Packet 1.** Contains $d_u \oplus d_p$, and has $I(Packet1) = 0$, and $X(Packet1) = X(p) \oplus X_{Index(u)}$.
      ii) **Packet 2.** Contains only d_u, and has $I(Packet2) = 1$, and $X(Packet2) = X_{Index(u)}$.
   b) However, if $NRC(p) < 2$ node u acts as a node of type 3, i.e., just a relay.

For the sake of illustration, we went through one more iteration for the descendents of B, where the NRC is set to 2 in the packets received by nodes F and G, resulting in

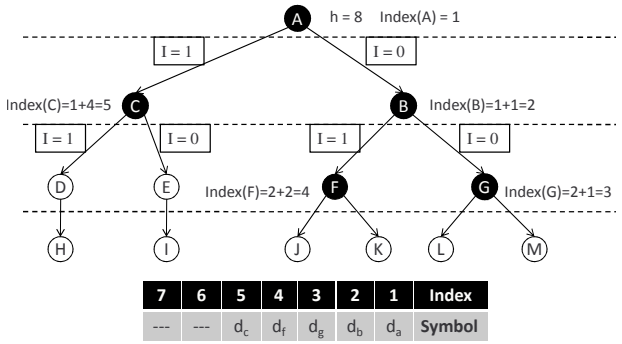


Fig. 4. Relative indexing, for $h = 8$

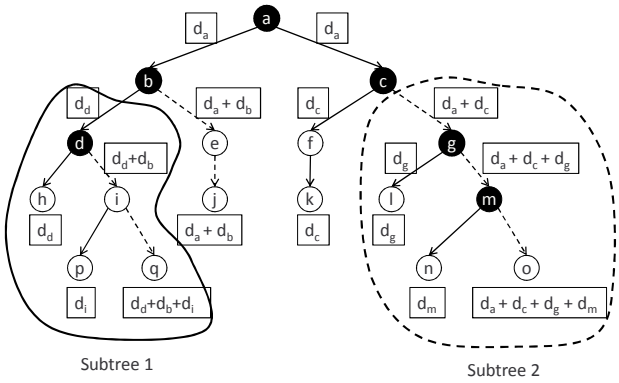


Fig. 5. Best Effort Decoding

G taking index 2+(1)=3 and F taking index 2+(2-1 + 1)=4. Lastly, we want to emphasize that relative indexing is valid only for a set of combinations having the same IID.

B. Best Effort Decoding

For the operation described in Section II, if e combinations were lost from the resulting $1 + N_1^j$ combinations in a certain F-process, the sink should not discard the remaining combinations because it may still be able to recover a subset of the encoded symbols. Actually, the remaining combinations received at the BS will still be solvable, if the lost combinations remove $e - 1$ data units, i.e., $1 + N_1^j - e$ equations remain in $N_1^j - (e - 1) = 1 + N_1^j - e$ unknowns. From the operation described in Section II this condition is satisfied for any subtree rooted at a Type 2 node u that combines its data unit with a trivial combination (i.e., a single data unit combination).

Consider a subtree rooted at a Type 2 node u, which encodes its data unit with a single data unit combination containing only d_v. The number of combinations this subtree will produce is equal to the NRC value in the packet carrying d_v. In addition, the number of new data units that will be added in these combinations by the sources in the subtree (including the root node u) is $NRC - 1$. Thus we will have NRC combinations in NRC unknowns, which can be solved. However, if the Type 2 node u combined its data unit with a non-trivial combination, the number of unknowns will be larger than the number of combinations. An example is shown in Figure 5, where Type 2 nodes that receive trivial combinations are the heads of all solid links, and Type 2 nodes that receive non-trivial combinations are the heads of all dashed links. For instance the combinations from Subtree 1 are solvable since node d combines its data unit with a trivial combination, while those from Subtree 2 are not.

In reality the BS does not need to search for solvable subtrees to decode them. Rather, it can use the decoding method in algorithm 1 for a set of combinations having the same IID, where RCV is the set that will contain the recovered data units, and Z is a set for temporary use in decoding. This way the BS will recover data units as much as it can, and will stop when no new single data units are found, hence the name best effort decoding.

Algorithm 1 Best Effort Decoding

1: $RCV = \phi$
2: Z= All single data unit combinations.
3: **while** $Z \neq \phi$ **do**
4: For each element in Z remove it from all combinations containing it, using bitwise XOR.
5: $RCV = RCV \cup Z$
6: Z= All new single data unit combinations.
7: **end while**

V. Routing for maximally disjoint paths

In all of our examples in the previous sections, we assumed that the data units or combinations will select routes to the BS in a way that constructs a binary tree. If we are considering packet loss only, then we do not need to restrict the routing of the combinations to the BS to be a tree. However, if we want to use our coding scheme to tolerate actual node or link failures, then we need a routing protocol that selects for each combination a path that is maximally disjoint from the paths of all other data units or combinations. Note that if all the paths are totally disjoint, then the routes from the sources in the different coding levels will form a tree. Therefore, to be able to tolerate node or link failures, we present a simple routing protocol that guarantees maximally disjoint paths in this section.

What we mean by maximally disjoint here is that the used paths from the sources in an F-process should be node-disjoint when possible (and thus, edge-disjoint too). But sometimes achieving node-disjointedness may not be possible, e.g., when two nodes that are forwarding packets with same IID have one downstream neighbor, say node X. In this case we should try to achieve edge-disjointedness if possible. The paths will be edge-disjoint if node X forwards the two packets with the same IID (i.e., from the same F-process) to two different downstream neighbors (if available). However, it may happen that even edge-disjoint paths cannot be established if the number of downstream neighbors to node X is not enough. For example, if node X has a single downstream neighbor then it will forward the two packets to the same downstream node, and in this case the paths will not even be edge-disjoint. The proposed routing protocol aims for node-disjoint paths first, then if not possible, it tries to achieve edge-disjoint paths, and it will not create overlapping paths unless it is forced to do so when the number of nodes is not enough to guarantee node

or edge-disjointedness.

For our routing protocol to work we need the underlying WSN to be organized into levels or rings around the BS, as assumed at the beginning of the paper. Initially, a node (say v) has its level l_v set to a very large value. Arranging the network into levels can be done through a simple process initiated by the BS. The BS starts by broadcasting a control packet containing a field called hop_count, which is initialized to 1. After that, if a node v receives this control packet with $hop\_count \geq l_v$, it discards the packet. Otherwise, it makes $l_v = hop\_count$, increases the hop_count by 1 and rebroadcasts the control packet.

WSNs are usually densely deployed. Therefore, during the process of organizing the network into levels, a node, v, will most likely receive more than one packet with the same minimum hop_count from nodes in $l_v - 1$. Each node stores the IDs of these neighbors in a list called $next\_hop\_list$. If a node needs to forward a data packet to the BS, it needs to only send it to one of the nodes stored in the $next\_hop\_list$ as we will clarify now. In a network where the nodes are organized into levels as we have shown, a very simple routing protocol can be used, in which a node just picks one of its neighbors in the $next\_hop\_list$ and broadcasts its data packet. When a node receives a packet it just checks the level of the source, if the source is farther from the BS (in a higher level), then it rebroadcasts the packet. Otherwise, if the source is closer to the BS, the heard packet is discarded. This protocol would suffice if we are concerned with packet loss only: a Type 1 node can include in the packet two nodes from its $next\_hop\_list$ to generate the needed redundancy; a Type 2 node includes the id of a different neighbor for each packet; and a Type 3 node just picks one neighbor from the $next\_hop\_list$. However, to tolerate node failures we need a way to limit each forwarding node (in the lower levels closer to the BS) to forward only one packet at most from each F-process (if possible, i.e., there are enough nodes in lower levels), so that the failure of a node can reduce the number of packets in any F-process by at most one. To eliminate extra routing overhead, we proposed to exploit the protocol used in MAC layer. So the proposed routing protocol does not deal with channel access, we only assume that there is some CSMA/CA MAC layer protocol that does this for us, and we are just tweaking it to eliminate any extra routing overhead.

We now define the operation for Type 3 nodes to produce maximally disjoint paths (the operation for Type 2 nodes is similar, but takes into account two packets instead of one). To establish maximally node-disjoint paths from the multiple sources in the same F-process, a Type 3 node must not forward more than one packet from the same F-process (i.e., with the same IID). Note that to establish maximally edge-disjoint paths, a Type 3 node can forward multiple packets with the same IID, but not to the same next hop (unless it is forced to do so). To do this, each node needs to know for which F-processes it had forwarded packets before. Therefore, each node stores the IID of each packet it forwards in a list called the IID_list. If node v is forwarding a packet p with $IID(p)$, it must select a neighbor in the $next\_hop\_list$ that does not have $IID(p)$ in its IID_list. This can be accomplished during the RTS/CTS negotiation, where the IID (say $IID(p)$) of packet p (which needs to be transmitted) is piggybacked in the RTS message, and a next-hop node in $l_v - 1$ can reply with a CTS if it does not have $IID(p)$ in its IID_list. Node v may not be able to find a neighboring node in $l_v - 1$ that does not have $IID(p)$ in its list. Therefore, to solve this problem, we add a new field to the RTS message, which is basically a flag (bit) called *"Force forward"* or simply FF. A node can force forward a packet, if it did not receive a CTS, by setting $FF = 1$ in the RTS and sending it again. Note that force forwarding does not force a node to forward data if it did not receive a CTS. Since a node does not send a packet unless it hears a CTS, then to force nodes in lower levels to send CTS messages (when they have already forwarded a packet from the F-process of the packet requesting to be sent) the FF flag can be used. If a neighboring node in $l_v - 1$ hears an RTS with $FF = 1$ it replies with a CTS regardless if it has $IID(p)$ or not. The operation of a node upon receiving a packet is described in Algorithm 2, and its operation when sending a packet is described in Algorithm 3. Although Algorithms 2 and 3 do not show it, but we assume that some back off mechanism is used before a packet is sent.

Algorithm 2 Operation of node v: Receiving packet p

```
1: if (p is a control packet) then
2:     if (hop_count > l_v) then
3:         Discard p
4:     else if (hop_count < l_v) then
5:         l_v = hop_count
6:         Clear next_hop_list
7:         Rebroadcast p
8:     else
9:         //hop_count = l_v
10:        Add sender of p to next_hop_list
11:        Rebroadcast p
12:    end if
13: else if (p is RTS received from level l_v + 1) then
14:    if (if IID(p) ∉ IID_list) then
15:        Send CTS
16:    else if (FF(RTS) == 1) then
17:        Send CTS
18:    end if
19: else
20:    //p is a data packet
21:    Store IID(p) in IID_list
22:    Send p downstream
23: end if
```

Algorithm 3 Operation of node v: Sending packet p

```
1: if (p is a control packet) then
2:     Broadcast p
3: else
4:     //p is a data packet
5:     Piggyback IID(p) on RTS
6:     if (CTS is received from node u before timeout) then
7:         Send p to node u
8:     else
9:         FF(RTS) = 1
10:        Wait for CTS
11:        Upon receiving CTS from node u, send p to u
12:    end if
13: end if
```

It is easy to see that by using the described routing protocol, all paths from all sources will be node-disjoint unless some node force forwards a packet. Also, the paths will be link-disjoint unless a node forwards two packet to the same next hop (one packet is normally forwarded, but the other is force forwarded to the same next hop). By using such a simple routing protocol, we can say that the F-process can tolerate at most one node (or link) failure if the paths from all sources are node (or link) disjoint. An example is shown in Figure 6, where an F-process is initiated by node A, and node B is the only downstream Type 2 node. In the example, node C force forwards the packet carrying data unit a because there are only two nodes in $l_C - 1$. After that, no force forwarding occurs, which produces edge-disjoint paths. Note that if another neighbor to node C is found in $l_C - 1$, then the paths would be node-disjoint. However, note that if there were only two nodes in $l_E - 1$, then E will force forward one of the data units a or b, and the resulting paths will not even be edge-disjoint. It should be noted that if the paths are totally disjoint, then the failure of a node can cause at most one combination to be lost even if it is a Type 2 node. Let v be a Type 2 node that receives packet p from some F-process, then if v fails before sending its two packets containing $d_v \oplus d_p$ and d_v, the failure of v will only cause d_p to be lost.

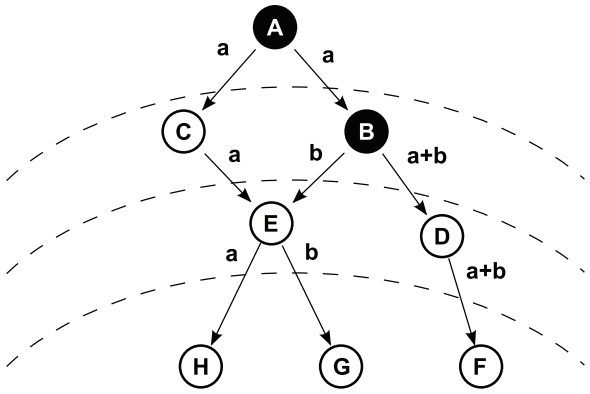


Fig. 6. The figure shows how the combinations created in an F-process initiated by node A are forwarded. Node C cannot find a downstream neighbor that did not forward a packet with IID = A. Therefore, it force forwards data unit a to node E, which results in paths that are not node-disjoint. Note that if another neighbor to node C is found the paths would be node-disjoint. Note also that if there were only two nodes in $l_E - 1$, then E will force forward one of the data units (either a or b), and the resulting paths will be neither link nor node-disjoint

At the beginning we assumed a dense WSN, as it is usually the case in most WSNs, and we therefore do not deal with sparse networks. However, in a worst case scenario where there is only one path from a node to the BS and all the nodes on the path are Type 2 nodes, then all the packets in the F-process will be sent (force forwarded) on the same path.

VI. Selecting parameter h

The min-cut is not the only factor that limits the number of combinations h. Any network link may be operational with a certain probability. As the value of h gets larger more sources will be able to participate in an F-process, which in turn increases the number of used network links, and thus increases the chance of having multiple link failures that can affect the ability of the sink to successfully decode the received combinations in a certain F-process. In this section we study the relationship between h and the probability of successful recovery $P(rcv)$. We assume that all network links have the same success probability, which we denote by q. In the following discussion, we focus on a single F-process and we show how $P(rcv)$ changes with respect to the value of h.

Since we need sources to participate in an F-process to further reduce the redundancy overhead, we need to know how the participation of these sources will affect $P(rcv)$. The probability of successful recovery depends on the number of used links, which in turn depends on three factors: 1) L, the level of the initiator node, 2) $h-1$, the number of sources that will participate in the F-process, and 3) the way the sources are distributed in the levels below the initiator node, or equivalently the resulting topology of the tree. The probability of successful recovery given some known topology, Tp_1, with k_1 links, is equal to the probability that at most one loss occurs in Tp_1, i.e.,:

$$\begin{aligned} P(rcv|Tp_1) &= P(\text{no loss in } Tp_1) + P(1 \text{ loss in } Tp_1) \\ &= q^{k_1} + k_1(1-q)q^{k_1-1} \end{aligned}$$

Therefore, we can compute $P(rcv)$ as follows:

$$P(rcv) = \sum_{\forall Tp_i} P(rcv|Tp_i)P(Tp_i)$$

Since we assume that all network nodes generate data units at the same rate, i.e., the probability of being a source is the same for all nodes, then all possible topologies can occur with the same probability. Therefore, we can write the probability of successful recovery as a function of L and $h-1$ as follows:

$$P(rcv) = P_{rcv}(L, h-1) = \frac{\sum_{\forall Tp_i} P(rcv|Tp_i)}{\text{\#possible topologies}}$$

$$= \frac{\sum_{\forall Tp_i} (P(\text{no loss in } Tp_i) + P(1 \text{ loss in } Tp_i))}{\text{\#possible topologies}}$$

$$= \frac{P_0(L, h-1) + P_1(L, h-1)}{\text{\#possible topologies}}$$

$P_0(L, h-1)$ is the sum of probabilities of no loss in all the possible topologies starting at level L with $h-1$ sources, and $P_1(L, h-1)$ is the sum of probabilities of exactly 1 loss in all the possible topologies starting at level L with $h-1$ sources. $P_0(.,.)$ and $P_1(.,.)$ are evaluated for trees starting at nodes of Types 1 or 2.

Following the rules of operation, the initiator node will send two packets containing copies of its data unit to the BS. As these packets are forwarded, a Type 2 node may participate and code its data with the data unit received from one of these packets, which also produces two packets according to

the rules of operation that may also trigger other downstream Type 2 nodes to participate. For the purpose of computing $P(rcv)$, we can ignore the contents of the packets forwarded from Type 1 or Type 2 nodes, and just focus on the number of transmissions or packets produced, we can view an F-process (initiated by a Type 1 node) as a recursive process that repeats itself (in some lower level, and with a fewer number of sources) on every downstream source (Type 2 node). This allows us to calculate $P_0(L, h-1)$ and $P_1(L, h-1)$ using recursive formulas, but first we need the following definitions:

- let $\pi(\lambda, \alpha)$ be the probability that there is no loss in the branch starting at level λ, with α sources downstream.
- let $\bar{\pi}(\lambda, \alpha)$ be the probability that there is a loss in the branch starting at level λ, with α sources downstream.
- let $\mu(\lambda, \alpha)$ be the probability that a loss will occur downstream, given that we are starting at level λ with α sources and no loss has occurred yet.
- let $\nu(\lambda, \alpha)$ be the probability that no loss will occur downstream, given that we are starting at level λ with α sources and a loss has already occurred.
- let $\rho(\lambda, \alpha)$ be the probability that no loss will occur downstream, and the root is not necessarily a source. This is similar to $P_0(\lambda, \alpha)$, but branching may not happen at level λ.

Notice that $\pi(.,.)$ and $\bar{\pi}(.,.)$ are evaluated at nodes of Type 3, since they start with branches, or links. Also note that $\mu(.,.)$ and $\nu(.,.)$ start at nodes on the tree which may be of any type, i.e., they have degrees 1 or 2.

Using these definitions we can write P_0 and P_1 as follows:

$$P_0(L, h-1) = \sum_{\alpha_1=0}^{\lfloor \frac{h-1}{2} \rfloor} \sum_{\forall\{\alpha_2 : \alpha_1+\alpha_2=h-2,\ 0\le\alpha_2\le\lfloor \frac{h-1}{2} \rfloor\}} \pi(L, \alpha_1)\pi(L, \alpha_2) \quad (3)$$

$$P_1(L, h-1) = \sum_{\alpha_1=0}^{\lfloor \frac{h-1}{2} \rfloor} \sum_{\forall\{\alpha_2 : \alpha_1+\alpha_2=h-2,\ 0\le\alpha_2\le\lfloor \frac{h-1}{2} \rfloor\}} \pi(L, \alpha_1)\bar{\pi}(L, \alpha_2) + \sum_{\alpha_1=0}^{\lfloor \frac{h-1}{2} \rfloor} \sum_{\forall\{\alpha_2 : \alpha_1+\alpha_2=h-2,\ 0\le\alpha_2\le\lfloor \frac{h-1}{2} \rfloor\}} \bar{\pi}(L, \alpha_1)\pi(L, \alpha_2) \quad (4)$$

where,

$$\pi(L, \alpha) = q.\rho(L-1, \alpha) \quad (5)$$

with $\pi(1, 0) = q$, and $\pi(1, \alpha \ge 1) = 0$ because there are no sources below level 1.

$$\bar{\pi}(L, \alpha) = q.\mu(L-1, \alpha) + (1-q).\nu(L-1, \alpha) \quad (6)$$

with $\bar{\pi}(1, 0) = 1-q$, and $\bar{\pi}(1, \alpha \ge 1) = 0$.

$$\mu(L, \alpha) = q.\mu(L-1, \alpha) + (1-q).\nu(L-1, \alpha) + P_1(L, \alpha) \quad (7)$$

with $\mu(1, 0) = 1-q$, $\mu(1, 1) = 2q(1-q)$, and $\mu(1, \alpha > 1) = 0$.

$$\nu(L, \alpha) = \nu(L-1, \alpha) + P_0(L, \alpha) \quad (8)$$

with $\nu(1, 0) = q$, $\nu(1, 1) = q^2$, and $\nu(1, \alpha > 1) = 0$.

$$\rho(L, \alpha) = P_0(L, \alpha) + q.\rho(L-1, \alpha) \quad (9)$$

with $\rho(1, 0) = q$, $\rho(1, 1) = q^2$, and $\rho(1, \alpha > 1) = 0$.

Finally, we need to compute the total possible number of topologies to be able to calculate $P_{rcv}(L, h-1)$. To do this we can use $P_0(L, h-1)$. Since P_0 computes the sum of the probability of no loss in all possible topologies setting $q = 1$ will make the probability of no loss equals 1 for each topology, and P_0 will result in counting the total number of possible topologies. That is, $P_{rcv}(L, h-1)$ can be calculated as follows:

$$P_{rcv}(L, h-1) = \frac{P_0(L, h-1) + P_1(L, h-1)}{P_0(L, h-1)|_{q=1}} \quad (10)$$

The probability of successful recovery is evaluated in the next section.

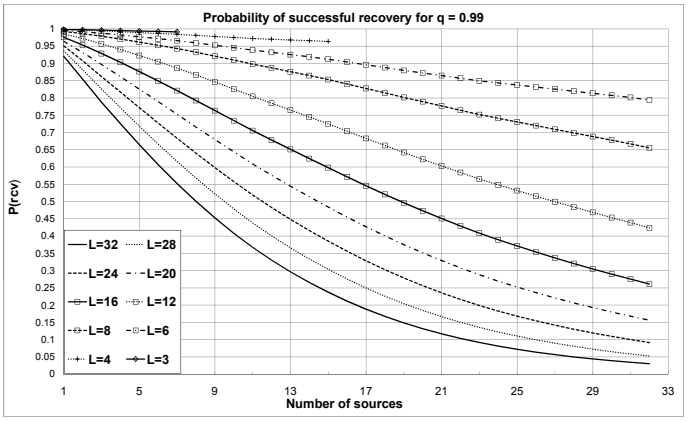
VII. Performance Evaluation

In this section we evaluate the probability of successful recovery, $P(rcv)$, based on the formula derived in the previous section. We also present simulation results from our implementation in TOSSIM, to illustrate the benefits of our scheme in reducing the protection overhead.

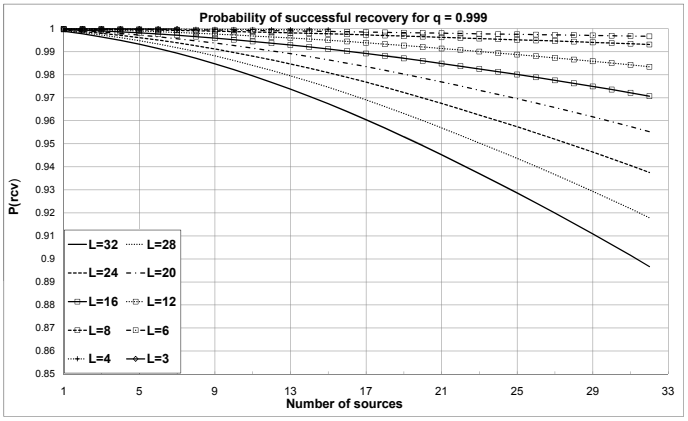
A. Evaluating $P(rcv)$

We plotted $P(rcv)$ against the total number of sources in an F-process starting at different levels, for $q = 0.99$, $q = 0.999$, and $q = 0.9999$. The results are shown in Figure 7. As expected, for some level L, increasing the number of participating sources in an F-process will reduce $P(rcv)$ because this increases the total number of used links. Therefore, depending on the desired probability of successful recovery, nodes in levels that are farther from the BS may want to just use duplication, or equivalently let $h = 2$ to prevent downstream Type 2 nodes from participating. Also, for a fixed number of sources, the figure shows that the probability of successful recovery increases as the initiator node gets closer to the BS (i.e., as L decreases).

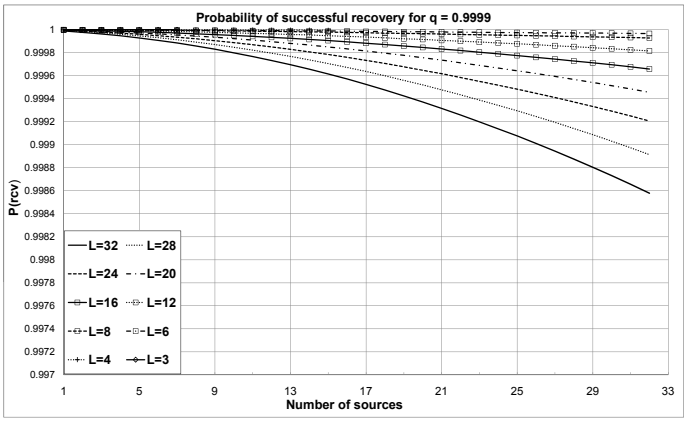
Note that the total number of combinations, h, produced by an F-process that has L levels is limited by 2^L, and that the total number of source nodes is limited by $2^L - 1$. We will see from the simulation results in the next section that we do not actually need a large number of Type 2 nodes to participate in an F-process to significantly reduce the duplication overhead. This is due to the fact that the reduction



(a) $q = 0.99$



(b) $q = 0.999$



(c) $q = 0.9999$

Fig. 7. Effect of the initiator node level (L) and the number of participating sources on the probability of recovery $P(rcv)$.

in the duplication overhead becomes negligible after the first few sources participate. Finally, it should be noted that all these calculations for $P(rcv)$ represent lower bounds, and are only valid if we are considering the recovery of the data units from all the sources participating in an F-process. This is because (as we have shown in Section IV-B) our coding scheme allows the partial recovery of a subset of the coded data units if they are from a subtree rooted at a Type 2 node that receives a native data unit. Therefore, the chances for recovery are better than what Figure 7 implies.

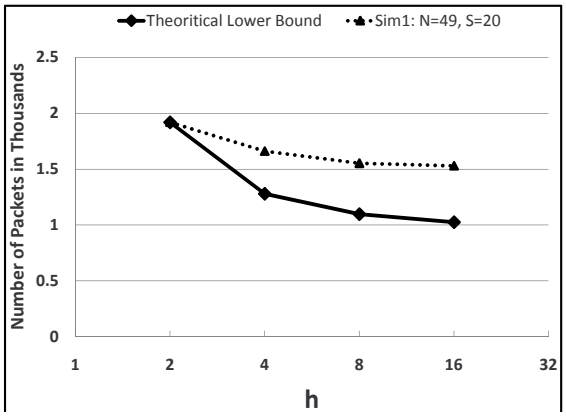
B. Evaluating the produced overhead

We evaluated the performance of our scheme in terms of produced overhead, using simulation on TOSSIM. The effects of the following three parameters were studied in our experiments: 1) S, which represents the number of packets generated by each sensor node, 2) the number of nodes in the network, denoted by N, and 3) h, the maximum number of produced combinations in an F-process, which controls the degree of coding. As a reference, we compared our results to a theoretical lower bound, which represents the best way in which packets can be combined. The best case occurs when every $h-1$ nodes belong to a certain F-process in which all of them are either Type 1 or Type 2 nodes. That is, we will have $\frac{N}{h-1}$ groups of nodes each of which produces a total of hS packets. This gives the following lower bound on the total number of packets produced:

$$(\frac{h}{h-1})N * S$$

We experimented with three scenarios, where in all of these scenarios h took the values $\{2, 4, 8, 16\}$. Note that when $h = 2$ no coding will take place and our scheme will be equivalent to traditional duplication based redundancy. The first scenario is a 7x7 grid network (i.e., N = 49) where each node produced 20 packets, the result is shown in Figure 8, where at $h = 16$ the duplication overhead is reduced by 42%. The second is also a 7x7 grid, but each node produced 50 packets to see how increasing the number of produced packets affects the performance. Figure 9 shows that by increasing the number generated packets our approach performs better and decreases the duplication overhead by 50% at $h = 16$. Finally, we increased the size to a 10x10 grid network (i.e., N = 100) and fixed the number of produced packets to 50, to see how the performance is affected by increasing the network size in addition to the number of packets. Again, the results assure that the performance gets better with increasing the network size and generated packets, where at $h = 16$ the duplication overhead is reduced by 72%. It is clear that the third scenario is the closest to the lower bound, since the chances for combining packets are enhanced as the network and/or number of generated packets grows larger. Although these scenarios are relatively small in size, they clearly illustrate that our coding-based approach is scalable and performs better as the network size increases.

To see how the probability of successful recovery ($P(rcv)$) affects the overhead, we modified the results in Figure 10



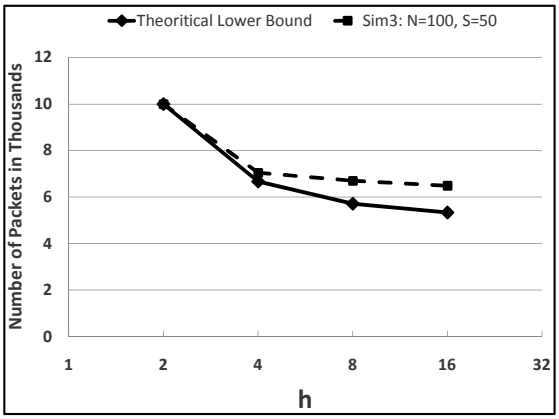


Fig. 10. Simulation results for N=100 and S=50

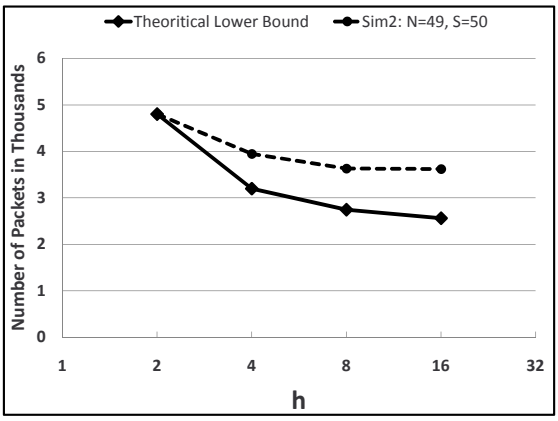


Fig. 9. Simulation results for N=49 and S=50

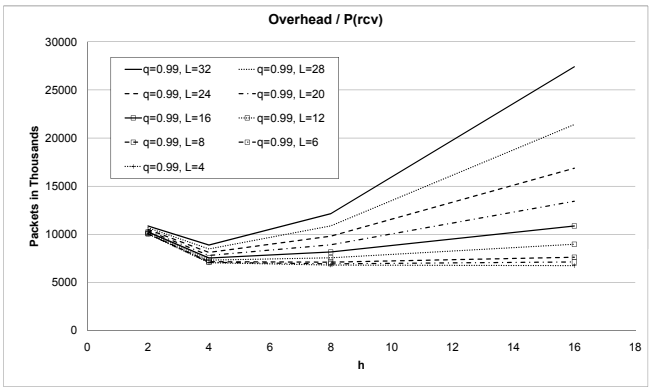


Fig. 11. The figure shows how the produced overhead changes with respect to the initiator node level (L), and the number of participating sources.

to reflect the change in $P(rcv)$. We divided the number of packets for each point in Figure 10 by the corresponding $P(rcv)$ (at $h-1$) from Figure 7(a). The result is shown in Figure 11. It is clear from the figure that network coding might not always be better than duplication especially if an F-process is initiated at a level that is far from the BS. The figure also shows that a good reduction in the overhead can be achieved by just letting a small number of sources participate in an F-process, e.g., 3 (at h = 4).

VIII. Summary

Deterministic binary network coding was utilized in a distributed manner to add lightweight redundancy to the information flow from the sensor nodes to the base station. Relative indexing was introduced to enable the use of an $(h-1)$-bit coding vector instead of an N-bit coding vector, where N is the number of sensor node (which can be in thousands), and $h << N$. In addition, we presented best effort decoding that allows us to make use of the combinations received at the BS even if more than one packet was lost. We enabled our scheme to tolerate link and node failures by presenting a simple routing protocol that guarantees maximally disjoint paths for the produced combinations in an F-process. We also studied the relationship between the probability of successful recovery, $P(rcv)$, and the number of sources participating in a certain F-process that is initiated from some level L. Finally, simulation results confirmed our theory and proved that our scheme is highly scalable, where it performs better as the network size and/or the number of sources increases.

References

[1] I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: a survey. Computer Networks 38 (2002) 393422.

[2] O. Al-Kofahi and A. Kamal. Survivability strategies in multihop wireless networks. Wireless Communications, IEEE. Vol. 17, Issue: 5, Oct 2010, pp. 71-80.

[3] F. Ye, G. Zhong, S. Lu, and L. Zhang. Gradient broadcast: A robust data delivery protocol for large scale sensor networks. ACM Wireless Networks (WINET), vol. 11, no. 2, March 2005.

[4] D. Ganesan, R. Govindan, S. Shenker, and D. Estrin. Highly-resilient, energy-efficient multipath routing in wireless sensor networks. ACM SIGMOBILE Mobile Computing and Communications Review archive Volume 5 , Issue 4 (October 2001).

[5] W. Lou. An efficient n-to-1 multipath routing protocol in wireless sensor networks. In Proc. 2nd IEEE International Conference on Mobile Ad-hoc and Sensor Systems (MASS 2005).

[6] N. Li and J.C. Hou. Flss: A fault-tolerant topology control algorithm for wireless networks. In Proc. 5th ACM International Symposium on Mobile Ad Hoc Networking and Computing (Mobihoc 2004).

[7] J.L. Bredin, E.D. Demaine, M. Hajiaghayi, and D. Rus. Deploying sensor networks with guaranteed capacity and fault tolerance. In Proc. 6th ACM International Symposium on Mobile Ad Hoc Networking and Computing (Mobihoc 2005).

[8] C. Chau, R. Gibbens, R. Hancock, and D. Towsley. Robust multipath routing in large wireless networks. In Proc. 30th IEEE International Conference on Computer Communications (IEEE INFOCOM 2011).

[9] Y. Sankarasubramaniam O.B. Akan I.F. Akyildiz. Esrt: Event-to-sink reliable transport in wireless sensor networks. In Proc. 4th ACM International Symposium on Mobile Ad Hoc Networking and Computing (Mobihoc 2003).

[10] Dae-Young Kim and Jinsung Cho. Active caching: A transmission method to guarantee desired communication reliability in wireless sensor networks. volume 13. IEEE COMMUNICATIONS LETTERS, 2009.
[11] S. Kim, R. Fonseca, and D. Culler. Reliable transfer on wireless sensor networks. In Proc. SECON 2004.
[12] S. Dulman, T. Nieberg, J. Wu, and P. Havinga. Trade-off between traffic overhead and reliability in multipath routing for wireless sensor networks. In Proc. WCNC 2003.
[13] W. Xiao, S. Agarwal, D. Starobinski, and A. Trachtenberg. Reliable wireless broadcasting with near-zero feedback. In Proc. 29th IEEE International Conference on Computer Communications (IEEE INFOCOM 2010).
[14] R. Ahlswede, N. Cai, S. R. Li, and R. Yeung. Network information flow. IEEE TRANSACTIONS ON INFORMATION THEORY, VOL. 46, NO. 4, JULY 2000.
[15] R. W. Yeung, S. R. Li, N. Cai, and Z. Zhang. *Network Coding Theory*. now Publishers Inc, 2006.
[16] O. Al-Kofahi and A. Kamal. Network coding-based protection of many-to-one wireless flows. IEEE Journal on Selected Areas in Communications special issue on Network Coding in Wireless Communications, Vol. 27, No. 5, June 2009,pp. 797–813.
[17] B. Saeed, P. Rengaraju, C. Lung, T. Kunz, and A. Srinivasan. Qos and protection of wireless relay nodes failure using network coding. In Proc. Network Coding (NetCod), 2011 International Symposium on.
[18] O. Al-Kofahi and A. Kamal. Max-flow protection using network coding. In Proc. IEEE International Conference on Communications 2011 (ICC 2011).
[19] O. Al-Kofahi and A. Kamal. Scalable redundancy for sensors-to-sink communication. In in the proceedings of the IEEE Globecom 2008.