

Chapter 2

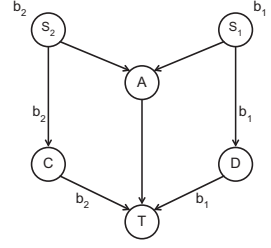
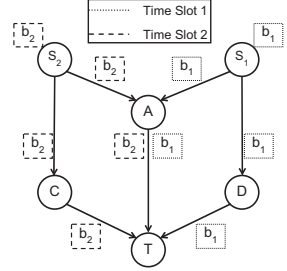
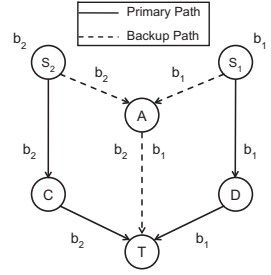
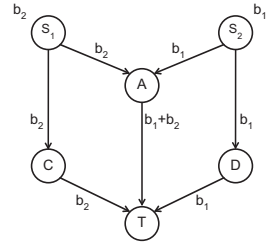
Network Coding-Based Resilient WSNs: The Centralized Approach

Link failures in Wireless Sensor Networks (WSNs) may occur due to severe channel fading, high levels of interference caused by other devices using the ISM band, or even physical damage to the network nodes or their antennas caused by the harsh surrounding environments. These problems may last for a considerable amount of time and they cannot be relieved using channel coding techniques or retransmissions. This chapter describes how to efficiently provide protection against such failures in a proactive manner while minimizing the used resources.

A centralized network coding-based scheme to provide proactive protection in a WSN will be introduced. This is done under the assumption that global network information is available, which may be the case in small size WSNs. With this assumption, this chapter makes it possible to focus on the theoretical aspects of the problem.

2.1 Problem Description

Let us consider the following motivating example shown in Figs. 2.1, 2.2, 2.3 and 2.4. In this network there are two sources, S_1 and S_2 that need to send two data units, b_1 and b_2 , respectively to a sink node T . To provide proactive protection against a single link failure each source must use the network in a different time slot to have two edge-disjoint paths to the sink as shown in Fig. 2.2. This is because the minimum-cut between the sources and the sink is 3. However, unlike proactive protection, if reactive protection is to be provided, the two sources can use the network in the same time slot as shown in Fig. 2.3. If a failure takes place on one of the primary paths the affected source will have to detect the failure first, and then reroute its data to use the backup path through node A , which introduces delay and interrupts network operation. Now suppose that node A in the network shown in Fig. 2.1 is allowed to combine b_1 and b_2 (bitwise XOR), and send the

Fig. 2.1 Original graph**Fig. 2.2** 1 + 1 protection**Fig. 2.3** 1:2 protection**Fig. 2.4** Network coding

resulting symbol to the sink on the link (A, T) , as illustrated in Fig. 2.4. This way, the two sources can use the network in the same time slot and still achieve proactive protection. If any of the three symbols sent to the sink is lost due to a link failure, the sink will still be able to recover the original data units. For example, assume that link (S_2, C) fails, the sink will receive $b_1 \oplus b_2$ on link (A, T) and b_1 on link (D, T) , and it can recover b_2 by performing the bitwise XOR operation on the received symbols.

For the purpose of the development in this chapter, two types of nodes, namely, source nodes (the ones producing data), and coding nodes (the ones that may

combine data) will be considered. From this perspective, the set of sources are referred to as U , and the set of coding nodes are referred to as L . To simplify the analysis, consider a WSN in which there is only one base station, and in which the nodes can be organized in t levels, where the minimum hop count between the base station and the nodes in level i is i (similar to [1, 2]). In general, it is better to combine the data units as early and as close as possible to the sources since this will reduce the used network resources. Therefore, we assume that U contains the nodes in some level j , and that L contains their direct neighbors in level $j - 1$. This assumption greatly simplifies the analysis, but does not compromise the theoretical results in any way as will be seen later.

In this chapter, the following questions are answered:

- How can network coding be used to provide protection against link (or path) failures in such many-to-one flow networks, while using the minimum possible number of paths?
- What are the necessary and sufficient conditions for such a solution to exist?
- If such a solution exists, how does it affect the network performance?

2.2 Network Coding-Based Resilience Approach

This section starts by developing the solution on a restricted network topology, which assumes the satisfaction of some connectivity and topology requirements (as will be stated below). Then, it is shown how to relax each of these requirements, and provide an appropriate generalization in Sect. 2.3.

2.2.1 Assumptions, Definitions and Notation

Since we are interested in the many-to-one flow from the sources in U to the base station, we can adopt the directed graph model in which a graph $G = (V, E)$ is used to represent the network. The set of vertices V represents the network nodes, and the set of edges E represents the available wireless links between network nodes, such that the edges are always directed from levels with higher indices to levels with lower indices. Considering the set of sources U and the set of coding nodes L , the edges will be directed from U to L since U contains the nodes in the next higher level to L . Taking that into account, we use the following assumptions and definitions:

1. Let $|U| = n$, and $|L| \geq n + 1$. In practice, this assumption is not always true. The reason to make such an assumption will become clear shortly. This assumption is relaxed in Sect. 2.3.4.

2. Let T represent the base station. Also, in the remainder of this chapter base station and sink may be used interchangeably since it is assumed that all nodes have the base station as their only destination.
3. All the links in the original graph G are of unit capacity unless stated otherwise, and there are no parallel links.
4. The minimum link cut between the nodes in L and the base station T is equal to $|L|$. Networks that do not have this property are discussed in Sect. 2.3.
5. The sub-graph induced by the nodes in U and L is bipartite. The general cases will be considered later in this chapter. This assumption can be naturally achieved in hierarchical or gradient-based routing protocols.
6. Only one link fails at a time.
7. All data packets have the same length.
8. G^T is the graph formed by: the nodes in U and L and all the links between them, a hypothetical sink node T' and hypothetical links from all the nodes in L to T' . G^T for the graph in Fig. 2.5 is shown in Fig. 2.6.
9. G^{ST} is the graph formed by: the nodes in U and L , and all the links between them, with a capacity of n assigned to each of these links, a hypothetical sink node T' , hypothetical links with capacity of n from all nodes in L to T' , a hypothetical source node S' and hypothetical links with capacity of $n + 1$ from S' to the nodes in U . G^{ST} for the graph in Fig. 2.5 is shown in Fig. 2.7.

Fig. 2.5 Graph G

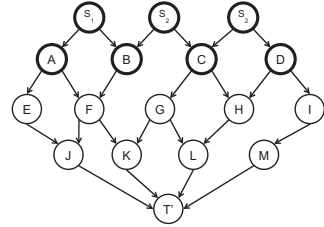


Fig. 2.6 Graph G^T

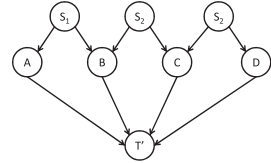
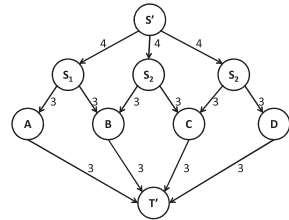


Fig. 2.7 Graph G^{ST}



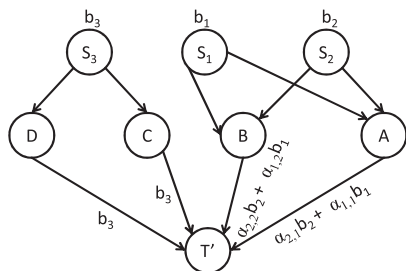
2.2.2 Sufficient and Necessary Conditions

To make the information flow from the n source nodes in U tolerant to a single link failure, we need to send $n + 1$ linear combinations (or equations) to the sink on $n + 1$ edge-disjoint paths. Moreover, to guarantee the successful recovery of the data, any n combinations from the $n + 1$ must be linearly independent (solvable). This way, the sink can recover the original n data units by solving any n from the $n + 1$ linear combinations, and since the $n + 1$ paths are edge-disjoint, a single-link failure will affect at most one path and only one combination will be lost. Note that similar to $1 + 1$ protection, there is no need to detect the failure. Note also, that similar to $1 : N$ protection only $n + 1$ paths are used.

The problem is divided into two sub-problems. The first deals with the needed information content in the linear combinations, i.e., how should the data units be incorporated in the combinations to guarantee the successful recovery of the original n data units in the case of a failure. The second is the coding problem to guarantee the linear independence of any n from the $n + 1$ linear combinations. This section focuses on the former and leaves the latter to Sect. 2.4. Thus, it is always assumed that the created combinations are linearly independent in this section.

Since we want to tolerate a single failure let us assume that $|L| = n + 1$ for now. From the assumptions above, each of the nodes in L has its own path to the sink that is edge-disjoint from the paths used by other nodes. Therefore, for simplicity, the original graph G can be replaced with G^T , where a path from a node in L to the sink is represented by a direct link. Taking this into account, the condition that will enable the L nodes to construct the $n + 1$ combinations that can tolerate a single link failure is that *any k nodes in U , must be connected to at least $k + 1$ nodes in L , for all values of k , where $1 \leq k \leq n$* . Consider the network in Fig. 2.8. The condition is satisfied for $k = 1$ since each node has two neighbors in L (or equivalently, has two paths to the sink). Also, the condition is satisfied for $k = 3$. However, it is not satisfied for $k = 2$ since S_1 and S_2 are only connected to two nodes in L . Therefore, if either of the links AT or BT fails, the sink will not be able to recover all three data units, because the other link that did not fail will be carrying the only combination of the two data units b_1 and b_2 , while the sink needs at least two.

Fig. 2.8 Condition not satisfied for $k = 2$. S_1 and S_2 are only connected to two nodes in L



We now continue with proving that this condition is necessary and sufficient for the nodes in L to be able to construct the $n + 1$ combinations that can tolerate a single link (or path) failure.

Proposition 2.1. *The sink will recover the n data units even if one of the $n + 1$ combinations is lost, if and only if, any subset of nodes in U of size k is connected to a subset in L of size at least $k + 1$, for all values of k , where $1 \leq k \leq n$.*

Proof. In the previous scenario we viewed the data units from sources as variables, and the $n + 1$ nodes in L as combinations (or equations), and a variable is present in an equation if the corresponding source is connected to the node representing that equation.

The implication is proved by contradiction. Assume that the sink is able to recover the n data units, even if one of the $n + 1$ combinations is lost. But let there be a subset of U nodes of size k , that is connected to a subset of L nodes of the same size k . Then, the sink cannot randomly choose n combinations from the $n + 1$, because it **MUST** pick all the k combinations that were formed by the subset of L nodes mentioned above; otherwise, the k variables from the corresponding k nodes in U will only be present in $k - 1$ equations, i.e., they cannot be recovered. This contradicts the assumption that the sink is able to recover the original n data units if **ANY** of the combinations was lost, which concludes the proof of the implication.

To prove the converse, contradiction will also be used. Assume that any subset of nodes in U of size k is connected to another subset of nodes in L that is of size at least $k + 1$. But, there is a mandatory combination, which cannot be lost for the sink to be able to recover the original n data units. A combination is essential and can not be lost, if it leaves a set of equations of size say l with $l + 1$ unknowns, which are impossible to solve without that combination. But for this case to happen, there must have been some $l + 1$ nodes in U that are only connected to $l + 1$ nodes in L , which contradicts the original assumption, of having any k nodes in U connected to at least $k + 1$ nodes in L , for all values of k , where $1 \leq k \leq n$. \square

Since all groups of nodes of all the sizes from 1 to n need to be checked, it may seem that checking the satisfiability of this condition is of order $O(2^n)$. It will now be shown how to check the condition above in polynomial-time with respect to the number of sources using a max-flow algorithm. The graph G^{ST} is used in the next proposition.

Proposition 2.2. *An S - T maximum-flow of at least $n(n + 1)$ is achievable in G^{ST} , if and only if, any subset of U of size k is connected to a subset in L of size at least $k + 1$, for all values of k , where $1 \leq k \leq n$.*

Proof. The implication is proved by contradiction. Assume the max-flow value is indeed $n(n + 1)$, then all the links from S to the n original sources are saturated (i.e., each one carries a flow equal to $n + 1$). And assume that there are some k nodes in U that are **only** connected to some k other nodes in L . The incoming flow to this component equals $k(n + 1) = kn + k$ while the outgoing capacity equals kn , which means that there are k units of flow that will be blocked from the sink, that is the max-flow = $n(n + 1) - k$ which contradicts the original assumption of the max-flow.

The converse is proved by contradiction. Suppose that any k nodes in U are connected to at least $k + 1$ nodes in L , but the maximum achievable flow was less than $n(n + 1)$, then there are some of the links from S to the nodes in U that could not be saturated. Assume only one of those links carried n units of flow to a node in U say node u . Then node u either has a single outgoing link, or is one of k nodes in U , that are connected to another set of k nodes in L (otherwise, it would have been able to forward this remaining unit of flow to the sink through an augmenting path on the residual network [3]). In both cases node u will violate the connectivity assumptions. Therefore, the max-flow must be $n(n + 1)$. This concludes the proof. \square

2.3 Generalizations and Practical Considerations

In this section we generalize the conditions in order to cover any network topology, and to tolerate multiple failures. Then we consider the cases when $|L| > n + 1$ and when the network has a limited min-cut.

2.3.1 General Network Topology

So far, we have assumed that the graph induced by the nodes in U and L is bipartite. This assumption may not always apply. Therefore, we now generalize the above conditions for general network topology.

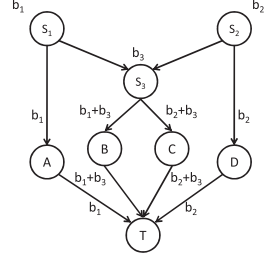
By carefully inspecting the condition of Proposition 2.1, one can see that the essence of the solution lies in the number of edge-disjoint paths from a group of sources to the sink. In the special case considered previously, each node in L represented one such path. Hence, Proposition 2.1 can be generalized in the following Theorem.

Theorem 2.1. *The sink will be able to recover the n data units even if **ANY** one from the $n + 1$ combinations is lost, if and only if, any subset of nodes in U of size k is connected to the sink through a set of edge-disjoint paths of size at least $k + 1$, for all values of k , where $1 \leq k \leq n$.*

Proof. The proof follows directly the same reasoning used in proving Proposition 2.1. \square

As an example to illustrate Theorem 2.1, consider a less restricted network topology, where links between sources are allowed, as shown in Fig. 2.9. It can be easily verified that the network in Fig. 2.9 satisfies the condition in Theorem 2.1. Moreover, this condition can be checked using the same idea in Proposition 2.2.

Fig. 2.9 Non-bipartite topology



2.3.2 Multiple Failures

The necessary and sufficient conditions for the case of multiple failures can be derived from the above discussion, and are summarized in the following theorem:

Theorem 2.2. *The sink will be able to recover the n data units even if e link failures occur (i.e., at most e combinations are lost), if and only if, any subset of U of size k is connected to the sink through a set of edge-disjoint paths of size at least $k + e$, for all values of k , where $1 \leq k \leq n$.*

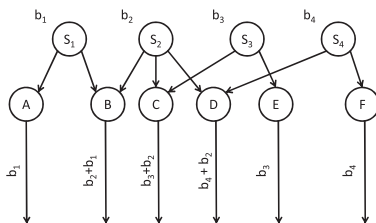
Proof. The proof follows directly the same reasoning used in proving Proposition 2.1. \square

Although two generalizations have been discussed in the rest of the chapter the analysis of the baseline case that satisfies the assumptions in Sect. 2.2.1 will continue.

2.3.3 The Case of $|L| > n + 1$

Until now, it has been assumed that the number of nodes in L is exactly $n + 1$. The number of L nodes could be larger than $n + 1$. This however, does not invalidate our conditions and the above requirements will still apply. However, there are two issues that should be noted. First, since $|L| > n + 1$ the number of produced combinations may be more than $n + 1$ depending on the connectivity between U and L . One extreme case is when each node in L is connected to only one node in U , which will cause the number of combinations to be $2n$. The second issue is that when the number of combinations is larger than $n + 1$, we cannot randomly select any n combinations to recover the original data units. The network shown in Fig. 2.10 gives a good example, where the minimum number of combinations is $n + 2$. In this network, selecting four combinations randomly may not cover all the data units. For example, if the combinations created in nodes C , D , E and F were chosen by the sink to calculate the original four data units, b_1 cannot be recovered. However, if any of the above mentioned four combinations was replaced by either

Fig. 2.10 Minimum combinations = $n + 2$



of the combinations from A or B , the sink will be able to recover all the original data units.

2.3.4 Networks with Limited Minimum Cuts

So far, it has been assumed that the number of edge-disjoint paths from L to T is larger than or equal to $n + 1$. Obviously, this may not be the case always. From Menger's theorem [4], the maximum number of edge-disjoint paths between the nodes in L and the sink is equal to the minimum edge cut (min-cut for short) between L and T . Let h denote the min-cut. If $h \geq n + 1$, then the developed approach can be applied directly, and the combinations formed in L can be forwarded to the sink. On the other hand, if h is less than $n + 1$, then the formed combinations cannot be forwarded as is, and the above approach must be modified.

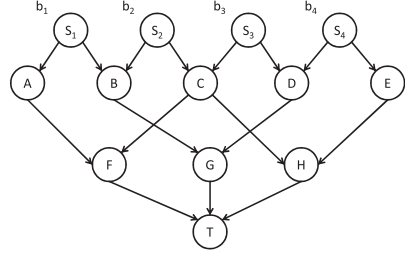
Let us assume that $h < n + 1$, then the sink cannot receive more than h combinations. That is, to protect against a single failure, the combinations should not contain data from more than $h - 1$ sources. Therefore, we divide the n sources into groups of size $h - 1$ sources each, and then choose a set of feasible groups that covers all the sources. It is assumed that the groups are time multiplexed, and a group of sources is defined to be feasible if it satisfies the condition in Proposition 2.1. A set of groups is defined to cover all the sources, if each source is present in at least one of the groups in the chosen set.

The way to choose the covering set of feasible groups must take the following into consideration:

1. The degree of disjointedness between groups, which affects the fairness and the rate at which the sources transmit, as will be seen in Sect. 2.5.
2. The used network resources.

As an illustration consider the network in Fig. 2.11, which contains four sources. The maximum number of edge-disjoint paths (h) from L to T in this network is 3. Therefore, the largest possible group of sources that can be protected together, will be of size at most 2. In this example all the groups of size two are feasible according to Proposition 2.1. Let us now compare the following three sets of groups:

Fig. 2.11 A network with $h = 3$



1. $Set_1 = \{\{S_1S_2\}, \{S_1S_3\}, \{S_1S_4\}\}$: In this set, the rate of S_1 is 1 *symbol/time slot* since it is present in all groups. While the rate of other sources is $\frac{1}{3}$ *symbol/time slot*, which is unfair.
2. $Set_2 = \{\{S_1S_3\}, \{S_2S_4\}\}$: this choice of groups achieves better fairness, where the bandwidth is equally divided and all the sources transmit at a rate of $\frac{1}{2}$ *symbol/time slot*.
3. $Set_3 = \{\{S_1S_2\}, \{S_3S_4\}\}$: this is the best solution, because not only we achieve better fairness than Set_1 , but also we use fewer network resources than Set_2 since each of the groups uses only three links to forward data to the minimum cut edges, compared to four links in the group of Set_2 .

The problem of choosing the smallest set of feasible groups to cover all sources can be proved to be NP-complete through a reduction from the **K-Set Cover** problem [5].

2.3.5 MILP Formulation

In this section the problem of source grouping is formulated as a mixed integer linear program. First, we define the following:

1. $s(k)$ source number k , where $1 \leq k \leq n$.
2. M the maximum number of groups which equals $n - h + 2$.
3. f_{ij}^{kc} the flow of source k in group c on edge (i, j) .
4. z_{ij}^c a binary variable which is equal to 1 if the edge (i, j) carried flow for group c and 0 otherwise.
5. g_c^k a binary variable which is equal to 1 only if source k was in group c and 0 otherwise.

To guarantee the feasibility of each group, we use the same method presented in Proposition 2.2. That is, we assume each source has a flow of h units, and that the capacity of all edges is $h - 1$. This way, a group will be feasible if a flow of $h(h - 1)$ can be forwarded to T . Taking this into account, the linear integer program is as follows:

Objective function:

$$\text{Minimize } \sum_{c=1}^M \sum_{\forall (i,j) \in E} z_{ij}^c \quad (2.1)$$

Subject to:

$$\sum_{\forall j: (s(k),j) \in E} f_{s(k)j}^{kc} = g_c^k \cdot h, \quad \forall k, c \quad (2.2)$$

$$z_{ij}^c - \frac{\sum_{k=1}^n f_{ij}^{kc}}{h-1} \geq 0, \quad \forall c, (i,j) \in E. \quad (2.3)$$

$$\sum_{\forall i: (i,j) \in E} \sum_{k=1}^n f_{ij}^{kc} = \sum_{\forall i: (j,i) \in E} \sum_{k=1}^n f_{ji}^{kc} \quad (2.4)$$

$$\forall c, j \notin \{T, s(1), \dots, s(n)\}$$

$$0 \leq \sum_{k=1}^n f_{ij}^{kc} \leq h-1, \quad \forall c, (i,j) \in E \quad (2.5)$$

$$\sum_{c=1}^M g_c^k = 1, \quad \forall k \quad (2.6)$$

$$\sum_{k=1}^n g_c^k \leq h-1, \quad \forall c \quad (2.7)$$

The objective in (2.1) is to minimize the number of used links for each group. Constraint (2.2) says that if source k was participating in group c the outgoing flow from it must be equal to h in the time slot for that group. Constraint (2.3) forces z_{ij}^c to be equal to 1 if the flow on edge (i,j) was not 0. Constraint (2.4) says that in a certain group (i.e. at a certain time slot) the amount of flow (of all sources) entering a node equals the amount of flow leaving that node. Constraint (2.5) says that the sum of flow of all sources in a certain group cannot exceed the capacity of any link which is equal to $h-1$. Constraint (2.6) ensures that each source participates in one group only, and (2.7) guarantees that a group contains no more than $h-1$ sources.

This MILP guarantees fair bandwidth sharing, i.e., a source cannot transmit again unless all other sources have transmitted. This is ensured by constraint (2.6) that forces each source to participate in one group only. As will be shown later in Sect. 2.5, a source might have the opportunity to transmit more than once without affecting the throughput of other sources; let us call this opportunistic transmission. The MILP can be modified for opportunistic transmissions as follows:

The objective function should be:

$$\text{Minimize } \sum_{c=1}^M \left(\sum_{k=1}^n g_c^k + \sum_{\forall (i,j) \in E} z_{ij}^c \right) \quad (2.8)$$

with the following modifications on constraints (2.6) and (2.7):

$$\sum_{c=1}^M g_c^k \geq 1, \quad \forall k \quad (2.9)$$

$$\sum_{k=1}^n g_c^k = h - 1, \quad \forall c \quad (2.10)$$

Now the bandwidth is utilized by constraint (2.10) that sets the size of all groups to its maximum size $h - 1$, and a source is allowed to participate in more than one group by constraint (2.9).

2.4 Coding

In the previous sections, linear independence between the linear combinations produced in L was assumed. In this section, it will be shown how to achieve this independence between combinations through using $\{0, 1\}$ coefficients. This reduces all operations to bit-wise XOR operations, and simplifies the coding and decoding processes.

Achieving independence using $\{0, 1\}$ coefficients depends solely on how one composes each combination from only the data units, i.e., a data unit is present in a combination if its coefficient is 1, and a data unit is not present if its coefficient is 0. For instance, in Fig. 2.4, the three combinations that were sent to the sink are, $C_1 = b_1$ (coefficient of b_2 is 0), $C_2 = b_2$ (coefficient of b_1 is 0) and $C_3 = b_1 + b_2$. In the following sections we assume that the connectivity condition of Proposition 2.1 is satisfied, and it will be shown how to decide on the data units composing each of the linear combinations, through finding simple paths and trees.

2.4.1 Path Coding and Tree Coding

Consider the bipartite graph induced by the nodes in U and L . Assume there exists a undirected path that has both ends in L . Note that since the graph is bipartite, the path will alternate between the nodes in U and the nodes in L . Assume also, that the path includes all the nodes in U . For such path, it is clear that any k nodes from U

on that path have at least $k + 1$ neighboring nodes from L that are also on the same path. In addition, note that the number of L nodes on such a path is the minimum number of nodes that satisfies the connectivity conditions, because each source has only two neighboring nodes in L .

Such a path not only finds a set of nodes in L that satisfy the connectivity conditions, but also helps in the assignment of the coding coefficients to create the needed linearly independent combinations. To illustrate the benefits of coding according to the connectivity on a path, consider the example in Fig. 2.12, where there are 4 source nodes in U , and 5 nodes in L . If every node in L XORs all the data units it receives, we will have dependent combinations like $\{b_4 + b_3\}$ and $\{b_4 + b_3\}$ (or $\{b_1 + b_2\}$ and $\{b_1 + b_2\}$), thus invalidating the linear independence requirements.

However, consider the simple undirected path $\{A, S_1, B, S_2, C, S_3, D, S_4, E\}$ that is represented by the solid edges in Fig. 2.13. If a coding coefficient of 1 is assigned to the links on the path (solid), and a coding coefficient of 0 is assigned to the links not on the path (dashed), then linear independence will be guaranteed, since any two combinations cannot have more than one element (i.e., data unit) in common.

Of course, it may not be always possible to find such a simple path that covers all the nodes in U . However, a path is only but a tree with two leaf nodes, or a tree with a root and one leaf node (depends on where the root of the tree is). Therefore, path coding can be generalized to tree coding, where we need to find a tree that covers all the nodes in U , with all of its leaf nodes in L . Note that requiring the leaves to be in L is only to guarantee that each node in U has two neighbors in L . Assigning tree links a coefficient of 1, and non-tree links a coefficient of 0 guarantees linear independence as stated in the next theorem and as illustrated in Fig. 2.14 (Fig. 2.15 clarifies the underlying tree structure).

Fig. 2.12 Bad coding: linear independence of any n combinations is not satisfied

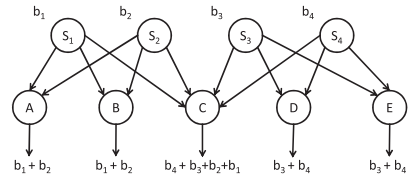


Fig. 2.13 Path coding: linear independence is satisfied in any n combinations

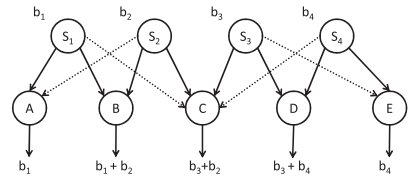


Fig. 2.14 Tree coding: linear independence is satisfied in any n linear combinations

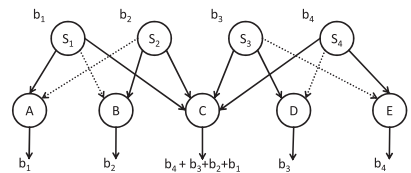
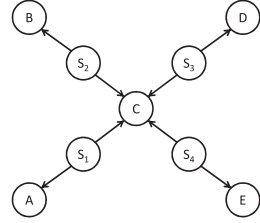


Fig. 2.15 Underlying tree

Theorem 2.3. *If the linear combinations at the nodes in L are created according to their connectivity with the source nodes in U on the coding tree, i.e., a link on the tree is assigned a coefficient of 1 and a link not on the tree is assigned a coefficient of 0, then any n combinations from the resulting $n + 1$ combinations are linearly independent.*

Proof. A direct proof is used to prove this implication. We prove that any n combinations from the $n + 1$ are linearly independent by proving that they are solvable by constructing an algorithm to solve for the n data units. In the algorithm the term “leaf combination” refers to a combination created at a leaf node in the tree, which will be a trivial combination that consists of a single data unit. The algorithm works as follows:

1. Put all the data units from the leaf combinations in a set; let us call it the *Recovered Data Units Set*, or the *RDU set* for short.
2. Remove all data units in the RDU set from the remaining combinations. This is done through XORing a data unit with all the combinations that it participates in.
3. After the previous step, a new set of data units will be recovered. These compose the new RDU set. The data units in the old RDU set will not be used further since they are removed from all combinations.
4. Repeat Steps 2 and 3 until all the data units are recovered.

To see how any n combinations are solvable, remove one of the combinations created at the nodes in L . There are two possibilities for this combination:

1. It is a leaf combination: in this case we are guaranteed to have at least one other leaf combination (for path coding), and the algorithm can start from it.
2. It is a non-leaf combination: in this case the coding tree is divided into two smaller trees, each of which will have at least one leaf combination (for path coding).

Note that the running time for this algorithm is $O(n)$, since in each step at least one data unit is recovered. The worst case occurs when the coding tree is a path and one of the leaf combinations is lost. \square

Algorithm 2.1 Construct a Coding Tree

```

1: while there are leaves from  $U$  do
2:   Pick a leaf node from  $U$  in the tree, say  $u$ 
3:   Find one of  $u$ 's neighbors in  $L$  say  $x$  //other than  $u$ 's parent
4:   Call ModTree( $u, x$ )
5: end while

```

Algorithm 2.2 **ModTree**(u, x)

```

1: Connect  $u$  to  $x$ , this will create a cycle, say  $C$ 
2: Traverse the nodes on  $C$ , until we reach a node in  $U$ , say node  $v$ , that has a neighbor  $w$  not on the cycle.
3: if  $w$  is already connected to  $v$  then
4:   Cut the cycle directly before or after  $v$ 
5:   return
6: else
7:   Cut the cycle before or after  $v$ 
8:   Call ModTree( $v, w$ )
9: end if

```

2.4.2 Constructing a Coding Tree

A coding tree can be constructed using the following three steps:

1. Build a tree rooted at a node in L . This can be a depth-first search or a breadth-first search tree (DFS or BFS). This step is of order $O(|E|)$.
2. Modify the tree to guarantee that all leaf nodes are in L .
3. If $|L| > n + 1$, trim extra leaves. This operation is of order $O(|V|)$.

The involved step is step two since steps one and three can be easily accomplished. Algorithms 2.1 and 2.2 describe a procedure to do the modification in the second step.

In Algorithm 2.1, we search the tree for a leaf node that falls in U . Upon finding such a leaf node u , we look for a neighbor x in L for node u that is different from the parent of u . It is guaranteed to find such a neighbor x , because each single node in U is connected to at least 2 nodes in L (Proposition 2.1). After finding u and x , the procedure **ModTree** adds the link between them to the tree creating a cycle C . Then, it traverses the nodes on C to find a node v in U that has a neighbor w in L not on C . Again, we are guaranteed to find such a node v that has such a neighbor w , because any k nodes in U are connected to at least $k + 1$ nodes in L , and since the cycle is composed of equal numbers of nodes from both U and L , then there must be a U node on this cycle that has a neighbor in L not on the cycle. If v was connected to w on the tree, we cut the cycle before or after v , to make the graph a tree again. On the other hand, if v and w are not connected on the tree, we recursively call **ModTree** (shown in Algorithm 2.2). As an illustration consider the network in Fig. 2.16, the resulting DFS-Tree (Depth First Tree) rooted at node C is shown in Fig. 2.17. The nodes that will be found when running Algorithm 2.1 and two

Fig. 2.16 Network with $n = 3$

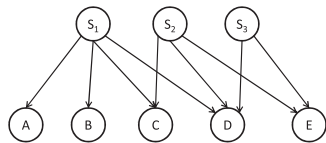


Fig. 2.17 DFS-Tree

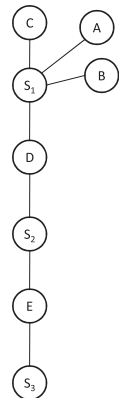
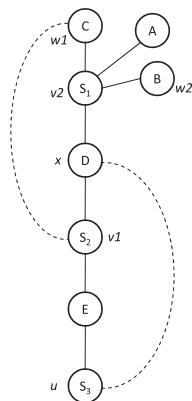


Fig. 2.18 Two iterations



iterations of **ModTree** are shown in Fig. 2.18, the cycles and the edges that are marked to be cut are shown in Fig. 2.19, and the final result after trimming the extra leaf nodes is shown in Fig. 2.20.

There can be at most $n - 1$ leaf nodes in U , and the recursive call to **ModTree** can be done at most n times. Hence, the running time of Algorithm 2.1 is of order $O(n^2)$.

Fig. 2.19 Making the modification

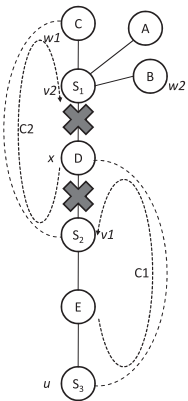
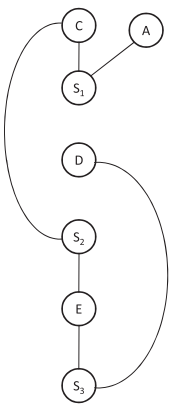


Fig. 2.20 Result



2.5 Network Performance

Sending redundant data reduces the effective data rate. This section studies the effect of network coding-based protection on the effective data rate in the following cases:

- **Case 1:** Fair bandwidth sharing, with no protection.
- **Case 2:** Fair bandwidth sharing, with protection.
- **Case 3:** Opportunistic transmission, with no protection.
- **Case 4:** Opportunistic transmission, with protection.

Fair bandwidth sharing means that a source does not transmit again until all other sources have transmitted, and opportunistic transmission means that a source transmits whenever it has an opportunity to do so. In the discussion, the rate \mathcal{R} is defined as the number of data units that can be received by the sink per unit time. Also the following is assumed:

1. If $h < n + 1$, sources will be divided into groups, as in Sect. 2.3.4.
2. There are two edge-disjoint paths from every source to the sink, i.e., minimum number of paths is used. This assumption will simplify the analysis, and will not affect its validity. This is because the rate is affected by the number of unique data symbols that can be recovered from the combinations, but not by the number of occurrences of the data symbols in the combinations.
3. It is assumed that the selected groups are feasible as defined in Sect. 2.3.4.

Case 1a: When $h \geq n$, the sink can receive all the n data units at the same time, which means that:

$$\mathcal{R} = n$$

Case 1b: When $h < n$, the sources should be grouped, which will cause the rate to vary at the sink depending on the way the sources were divided:

1. The best grouping scenario is when the sources are sorted in disjoint groups, giving rise to $\lceil \frac{n}{h} \rceil$ groups, i.e., $\lceil \frac{n}{h} \rceil$ time units are needed for all the sources to be covered, which means that:

$$\mathcal{R} = \frac{n}{\lceil \frac{n}{h} \rceil} \leq \frac{n}{\frac{n}{h}} = h$$

2. Since every source has two-edge disjoint paths to the sink, then any source can reach two different edges in the min-cut. The worst grouping scenario occurs when there are $h - 2$ min-cut edges that can be only reached by $h - 2$ sources, forcing the remaining $n - h + 2$ sources to use just the remaining two min-cut links. Because we assume no protection in this case, each source can use one edge in the min-cut, hence, the $n - h + 2$ sources can be divided into $\lceil \frac{n-h+2}{2} \rceil$ pairs, each pair when combined with the other $h - 2$ sources will form a group. Which produces $\lceil \frac{n-h+2}{2} \rceil$ similar groups that only differ in two elements. The network in Fig. 2.21 shows an example, where the sources S_1, S_2, \dots, S_{h-2} are present in all the selected groups (although we assume that they transmit in only one time slot out of the $\lceil \frac{n-h+2}{2} \rceil$), and the sources from S_{h-1} to S_n can connect to the sink only through the last two min-cut links. In this case the $n - h + 2$ sources will share these two links in $\lceil \frac{n-h+2}{2} \rceil$ time slots. Which means that:

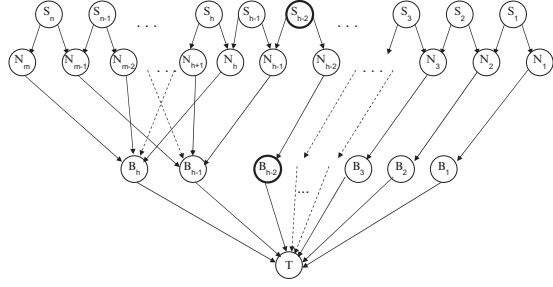
$$\mathcal{R} = \frac{n}{\lceil \frac{n-h+2}{2} \rceil} \leq \frac{n}{\frac{n-h+2}{2}} = \frac{2n}{n-h+2}$$

It can be seen that the last two cases are equivalent when $h = 2$, and they both give $\mathcal{R} = 2$.

Case 2a: When $h \geq n + 1$ the sink can receive the $n + 1$ combinations at the same time and recover all the n data units using any n combinations, that is:

$$\mathcal{R} = n$$

Fig. 2.21 Worst case grouping



Case 2b: When $h < n + 1$ the sources should be divided into groups, and the rate at the sink will depend on how the grouping was accomplished:

1. As before, the best grouping is when the sources can be divided into disjoint groups, but in this case, since protection is assumed to be provided, the groups will be of size $h - 1$, thus producing at most $\lceil \frac{n}{h-1} \rceil$ groups. Hence, the rate will be:

$$\mathcal{R} = \frac{n}{\lceil \frac{n}{h-1} \rceil} \leq \frac{n}{\frac{n}{h-1}} = h - 1$$

2. The worst case grouping is similar to that discussed in Case 1b, but in this case since protection is assumed, the $n - h + 2$ sources mentioned earlier will not be divided into pairs, rather, each of which will be active alone with the other $h - 2$ sources, thus giving rise to $n - h + 2$ groups, each of size $h - 1$. The network in Fig. 2.21 still gives a valid example. The rate in this case is:

$$\mathcal{R} = \frac{n}{n - h + 2}$$

Note that the last two cases are also equivalent when $h = 2$, and give $\mathcal{R} = 1$, since each source must use two edge-disjoint paths to forward data to the sink.

Case 3: The calculations from Case 1 are still valid for this case, except when $h < n$ with worst case grouping, in which the rate at the sink will equal h , since the $h - 2$ sources are allowed to transmit in every one of the $\lceil \frac{n-h+2}{2} \rceil$ time slots. To capture the difference, we should consider the rate at the sources, where it was $\frac{1}{\lceil \frac{n-h+2}{2} \rceil}$ for each of the n sources in Case 1. However, in this case, $\mathcal{R} = 1$ for each of the $h - 2$ sources that are repeated in all groups and $\frac{1}{\lceil \frac{n-h+2}{2} \rceil}$ for each of the $n - h + 2$ sources that can only use two min-cut links.

Case 4: The only difference between this case and case 2, is when $h < n + 1$ with worst case grouping, in which the rate at the sink will always be equal to $h - 1$. Again, to see the difference we should consider the rate at the sources, where it will be 1 for each of the $h - 2$ common sources, and $\frac{1}{n-h+2}$ for each of the $n - h + 2$ sources that share only two min-cut edges.

2.6 Chapter Summary

This chapter presented a centralized approach for network coding-based protection. This mechanism provides protection to many-to-one flows at the speed of proactive protection but at the cost of reactive protection. Necessary and sufficient conditions to achieve this protection for n source nodes were derived. Basically, for a set of n sources that have a common destination, it was proven that to be able to send $n + 1$ linear combinations (from the n sources) such that any n combinations are enough to recover the original data units, any k source nodes must have at least $k + 1$ edge-disjoint paths to the common destination for all values of k , where $1 \leq k \leq n$. Note that sending $n + 1$ combinations on $n + 1$ disjoint paths guarantees protection against one failure. The study was generalized, and conditions to cover multiple failures were derived, and it was shown how to deal with networks with limited minimum cuts. In addition, a polynomial-time algorithm to perform binary network coding (i.e., using $\{0,1\}$ coefficients) was introduced. This simplifies the decoding process where only XOR operations are needed to recover the original data units. Finally, the effect of the network coding-based protection approach on the performance of the network was studied, and the data rate with and without protection was calculated.

References

1. C. Schurgers, S. Mani, "Energy efficient routing in wireless sensor networks.", In Military Communications Conference, 2001. MILCOM 2001. Communications for Network-Centric Operations: Creating the Information Force. IEEE, vol. 1, pp. 357–361. IEEE, 2001.
2. P. Quang and D. Kim, "Enhancing real-time delivery of gradient routing for industrial wireless sensor networks", Industrial Informatics, IEEE Transactions on 8, no. 1 (2012): 61–68.
3. T. Cormen, C. Leiserson, R. Rivest and C. Stein. *Introduction to Algorithms*. The MIT Press; 3rd edition (July 31, 2009).
4. J. Kleinberg and E. Tardos. *Algorithm Design*. Addison Wesley, 2005.
5. M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. WH Freeman New York, 1979.

Resilient Wireless Sensor Networks

The Case of Network Coding

Al-Kofahi, O.M.; Kamal, A.E.

2015, X, 68 p. 55 illus., 49 illus. in color., Softcover

ISBN: 978-3-319-23963-7