Network Coding-Based Protection of Many-to-One Wireless Flows

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

***Abstract*—This paper addresses the problem of survivability of many-to-one flows in wireless networks, such as wireless mesh networks (WMNs) and wireless sensor networks (WSNs). Traditional protection schemes are either resource-hungry like the $(1+1)$ protection scheme, or introduce a delay and interrupt the network operation like the $(1:N)$ protection scheme. In this paper, we present a network coding-based protection technique that overcomes the deficiencies of the traditional schemes. We derive and prove the necessary and sufficient conditions for our solution on a restricted network topology. Then we relax these connectivity requirements and show how to generalize the sufficient and necessary conditions to work with any other topology. We also show how to perform deterministic coding with $\{0,1\}$ coefficients to achieve linear independence. Moreover, we discuss some of the practical considerations related to our approach. Specifically, we show how to adapt our solution when the network has a limited min-cut; we therefore define a more general problem that takes this constraint into account, which prove to be NP-complete. Furthermore, we discuss the decoding process at the sink, and show how to make use of our solution in the upstream communication (from sink to sources). We also study the effect of the proposed scheme on network performance. Finally, we consider the implementation of our approach when all network nodes have single transceivers, and we solve the problem through a greedy algorithm that constructs a feasible schedule for the transmissions from the sources.**

I. Introduction

The many-to-one communication mode is used in a number of networks including two of the newer types of networks. The first is Wireless Mesh Networks (WMNs), which are usually deployed to provide last-mile service to end users. WMNs are composed of Wireless Mesh Routers that form an infrastructure, which in turn is used to serve the Wireless Mesh Clients. In a WMN a router is called a gateway if it is connected to the wired network, where gateways provide Internet access to other routers through wireless multihop communication. The traffic in a WMN is either many-to-one from the wireless mesh clients to the gateway, or one-to-many from the gateway to the wireless mesh clients. Fig. 1 shows a WMN with a single gateway.

The second is Wireless Sensor Networks (WSNs), which are composed of a large number of sensing nodes that are deployed in a specific area of interest to monitor a certain phenomenon or to report the occurrence of certain events. Sensors in WSN can work in two modes; 1) they can continuously

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

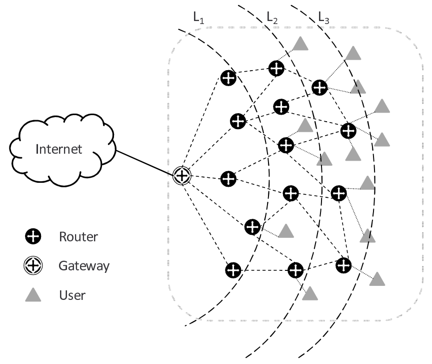


Fig. 1. Wireless Mesh Network

(e.g., periodically) acquire and send information from their surrounding environment to the base station (BS), or 2) they can be event-driven, where only upon detecting an event (e.g., an intruder) the sensing nodes in the surrounding region send the acquired data to the BS. In the continuous mode all the sensors send data to the BS, while in the event-driven mode only the sensors in the vicinity of the event send data to the BS.

Wireless mesh clients or sensor nodes are information sources that need to send their data to the destination (the gateway in a WMN or the BS in a WSN). However, the wireless communication medium is prone to various types of interference causing a wireless-link status to dynamically change according to the channel conditions, hence resulting in information loss. ARQ (Automatic Repeat reQuest) and FEC (Forward Error Correction) may help in such scenarios, but they will not be of any benefit if the channel was down permanently (or for a considerable amount of time). If the channel's Bit Error Rate (BER) is high (due to interference), there will be too many errors in the received packet for the FEC to correct, and retransmission (by ARQ) will be only wasting the energy of the transmitting node. To mitigate the effects of such a problem, various schemes to enhance the survivability of wireless networks against link failures were presented. Most of the survivability schemes in literature address the unicast [1] and multicast/broadcast [2] communication modes. In general, survivability mechanisms are divided into three main classes, 1) protection schemes [3][4][5][6][7][8]. 2) restoration schemes [9][10], and 3) hybrid schemes [11][12].

The main difference between protection and restoration schemes is that protection schemes reserve backup resources in advance (as a precaution), before the occurrence of a failure. In contrast, restoration schemes wait until a failure is detected, and then they start discovering the available resources, which introduces a delay in the recovery process. Hybrid schemes resort to restoration when protection fails. We only focus on protection since it is faster than restoration. The following classification of protection mechanisms is borrowed from the optical networks domain [13], where protection schemes are divided into two main categories:

1) Proactive protection, viz., 1+1 protection. In this scheme each source uses two edge-disjoint paths, each of which carries a copy of the data unit to the sink. Thus, if a link on one of the paths fail, the sink receives another copy on the other path. This is a very resource demanding solution, because we need at least twice as many resources, which makes this approach hard to realize.

2)A less demanding solution is to use reactive protection, viz., 1:N protection. In this scheme each source uses a primary path to the sink that is edge-disjoint from the primary paths used by other sources. In addition, an extra backup set of links (backup circuit) is reserved to be used by any of the sources if a failure occurs. Unlike proactive protection, where each source has its own backup path, sources are allowed to share the backup circuit, which makes better use of the network resources and makes this scheme more efficient.

In this paper, we utilize network coding [14][15][16] to provide protection in a proactive manner to many-to-one flows. The main advantage of using network coding is in reducing the needed resources to provide such protection. This is a new application of network coding in wireless networks, and to the best of our knowledge, using network coding in this direction has not been explored. The main result in the paper (Theorem 1) is summarized as follows:

In a many-to-one flow network with n sources, the single destination (sink) will be able to recover the n data units (from the n sources) in the case of a single link failure, if and only if, any subset of the n sources of size k can reach the sink through a set of edge-disjoint paths of size at least $k+1$, for all values of k such that $1 \leq k \leq n$.

The rest of this paper is organized as follows. Section II, describes the problem, and states the paper contributions. In Section III, we discuss the sufficient and necessary conditions for our solution to exist. Three generalizations of the original problem are discussed in Section IV. In Section V we show how to perform network coding using $\{0, 1\}$ coefficients. Section VI discusses some of the practical issues related to our approach. The network performance is studied in Section VII. In Section VIII we show how to schedule the sources transmissions if all the network nodes have a single transceiver, and we compare the performance of our approach with both the 1:N and the 1+1 protection schemes in terms of the required number of time slots. Finally, we conclude the paper in Section IX.

II. Problem Description

Link failures 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 weather conditions. These problems may last for a considerable amount of time and they cannot be relieved using FEC or ARQ. Our objective in this paper is to efficiently provide protection against such link failures in a proactive manner, but at the cost of reactive protection, i.e., using the minimum number of paths. We accomplish this by using network coding.

Let us consider the following motivating example shown in Figures 2(a), 2(b), 2(c) and 2(d). 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(b). This is because the minimum-cut between the sources and the sink is 3. However, unlike proactive protection, if we want to provide reactive protection the two sources can use the network in the same time slot as shown in Fig. 2(c). However, 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 we allow node A in the network shown in Fig. 2(a) to combine b_1 and b_2 (bitwise XOR), and send the resulting symbol to the sink on the link (A, T), as illustrated in Fig. 2(d). 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.

Although the main result in this paper is applicable to both WMNs and WSNs (and any other network that supports the many-to-one flow structure), we focus our discussion only on WMNs. In addition, we focus our discussion on link failures although the theorems and lemmas can be easily extended for the node failure case.

We consider a WMN, in which there is only one gateway, and in which the routers can be organized in t levels, where the routers in level i are i hops away from the gateway (e.g., the network in Fig. 1 has 3 levels). In addition each level of routers has a set of associated users that communicate with each other through the routers only. From now on, we refer to the routers and users in level i by L_i and U_i respectively. We assume that the router nodes work on two frequency channels, one for the communication between the routers themselves to construct and use the underlying infrastructure, and the other to communicate with users so that users do not interfere with routers. In addition, we assume that the t levels access the wireless medium in a TDMA manner, where each level of routers is assigned a different time slot, that is used to send data units from those routers, i.e. we study one level at a time. Actually, we assume that in each time slot the users transmit first (according to a schedule that will be discussed in Section VIII), and routers can start their transmissions afterwards.

In general, since each level of routers and their associated users are active alone in their assigned time slot, we assume that there are n source nodes in the network, which represent the users in the active level. We assume that each user generates a single data unit. Therefore, there are n data units from the n users that should be forwarded to the gateway router. Fig. 2(a) shows a network with two source nodes.

Our contribution in this paper lies in answering the following questions:

- 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?
- How does this scheme perform compared to the 1+1 and the 1:N schemes in practice?

III. Proposed approach

In this section we start by developing our solution on a restricted network topology, which assumes the satisfaction of some connectivity and topology requirements (as will be stated below). Then we show how to relax each of these requirements, and provide an appropriate generalization in Section IV.

A. Assumptions, Definitions and Notation

Since we are interested in the many-to-one flow from the users in U_i to the gateway, 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 (users and routers), 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 and from users to routers in a certain level. Taking that into account we define the following:

1) Let U_s be the set of users in the level being considered, where $|U_s| = n$.
2) Let T be the only sink node in the network.
3) Let L_s be the set of routers in level s, where $|L_s| \geq n{+}1$. In practice, this assumption is not always true and the reason to make such an assumption will become clear shortly. We relax this assumption in Section VI.
4) All the links in the original graph G are of unit capacity, and there are no parallel links.
5) The minimum link cut (or the minimum node cut, if we are concerned with node failures) between the nodes in L_s and the sink node T is $\geq n + 1$. Networks that do not have this property are discussed in Section VI.
6) The sub-graph induced by the nodes in U_s and L_s is bipartite. We will consider general cases later in this paper. In reality, this assumption is half true since users in a WMN communicate only through routers, i.e., in terms of edges on graphs, no two user nodes have an edge in between. However, routers in the same level may communicate with each other, and the graph therefore may have edges between router nodes; we call such a graph a semi-bipartite graph. We show in the appendix a simple procedure to find an equivalent bipartite graph for any semi-bipartite graph that has edges between router nodes.
7) Only one link fails at a time.
8) A node (either a user or a router) can receive from, or transmit to, multiple nodes simultaneously. This can be done by using multiple transceivers at each node. In Section VIII we show how to handle nodes with single transceivers.
9) All packets have the same length.
10) G^T is the graph formed by: the nodes in U_s and L_s and all the links between them, a hypothetical sink node T' and hypothetical links from all the nodes in L_s to T'.
11) G^{ST} is the graph formed by: the nodes in U_s and L_s, 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_s to T', a hypothetical source node S' and hypothetical links with capacity of n+1 from S' to the nodes in U_s.

As an illustration of points 10 and 11 above, a simple graph is shown in Fig. 3, and its corresponding G^T and G^{ST} are given in Fig. 4 and Fig. 5 respectively. In these figures S_1, S_2 and S_3 are the nodes in U_s, and A, B, C and D are the nodes in L_s.

B. Sufficient and necessary conditions

Suppose we can deliver to the sink $n{+}1$ linear combinations (or equations) of the original n data units on $n{+}1$ edge-disjoint paths, such that, any n combinations are linearly independent (solvable). The sink can recover the original n data units by solving any n from the $n + 1$ linear combinations. Since the $n + 1$ paths are edge-disjoint, a single-link failure will affect at most one path. That is, the sink will still receive n linearly independent combinations and will be able to recover the original n data units. As in the $1{+}1$ protection scheme the recovery can be done without the need to detect the failure, and compared to the $1 : N$ protection scheme, this approach requires the same number of paths ($n{+}1$), but does not impose a delay or interrupt the network operation. This clarifies the basic idea of our approach. An example is shown in Fig. 2(d).

We divide the problem 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 combinations from the $n + 1$ linear combinations. In this section we focus on the former and leave the latter to Section V. Thus, we always assume that the created combinations are linearly independent in this section.

As mentioned earlier, only one level of users, U_s, is active at a certain time, and our goal is to use deterministic network coding to provide proactive protection for the n users in that level. Under assumption 6, coding cannot begin in sources, since each of which only knows its own data unit and does not have any knowledge about the other data units in other sources. Therefore, creating the $n + 1$ combinations is the responsibility of the intermediate network nodes that connect the sources to the sink.

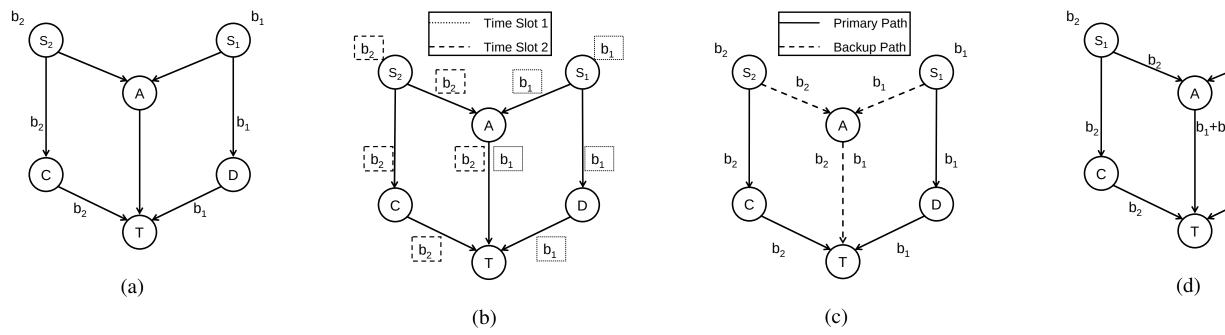


Fig. 2. (a) Original graph (b) 1+1 Protection, (c) 1:2 Protection, (d) Network coding Protection

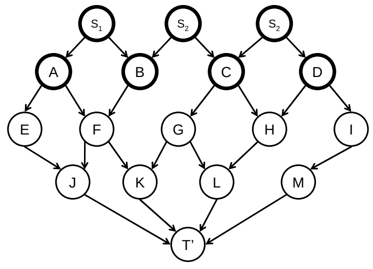


Fig. 3. Graph G

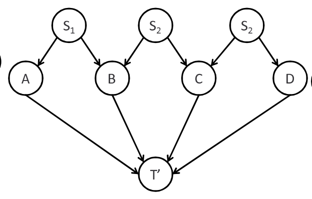


Fig. 4. Graph G^T

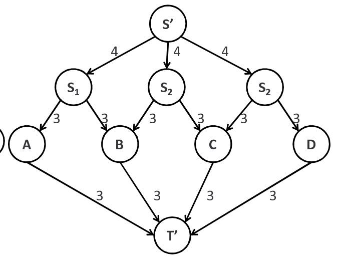


Fig. 5. Graph G^{ST}

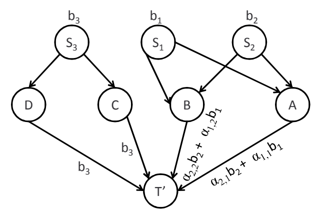


Fig. 6. Condition not satisfied

It is better to do the coding as close as possible to the sources, since this will reduce the used network resources as we will show in Section VI. Thus, we consider the closest nodes in the intermediate network to U_s that can perform coding on the data units, i.e., the nodes of L_s. We assume the case when $|L_s| = n + 1$, i.e., each one of the nodes in L_s is responsible for producing one combination and forwarding it to the sink. From assumption 5, each of the nodes in L_s has its own path to the sink that is edge-disjoint from the paths used by other nodes. Therefore, for simplicity, our original graph G can be replaced with G^T, where a path from a node in L_s to the sink is represented by a direct link. Taking this into account, the condition that will enable the L_s nodes to construct the $n+1$ combinations that can tolerate a single link failure is:

Condition: *Any k nodes in U_s, must be connected to at least $k + 1$ nodes in L_s:* Consider the network in Fig. 6, 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. Consider the linear combination created in a node v in L_s. There are n possible participants that can contribute to creating this linear combination. Let us assume that the combination consisted of two data units (i.e. v is connected to two source nodes in U_s). Then the sink can recover the two data units upon the failure of the path from node v if these two symbols were present in at least two other equations such that there are n independent equations in n unknowns. That is, if node v is connected to k nodes in U_s, then for the sink to be able to recover all the original data units if the combination created in v is lost, the neighboring set in L_s which encode data units from the k nodes in U_s must be of size at least k, or $k + 1$ if we include v. In general we can say: *Any group of nodes in U_s of size k must be connected to at least $k + 1$ nodes in L_s.*

Using the concept of matching in graph theory, an equivalent statement would be: $n + 1$ *perfect matchings between the nodes in U_s and those in L_s must exist, such that each matching, M_i, corresponds to the case when one of the nodes in L_s, v_i, is removed, where* $1 \leq i \leq n + 1$. This guarantees the existence of a dedicated alternate path from every source node to the sink upon a single link failure. This condition implies that the Max-flow is greater than or equal to 2 from every source to the sink.

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

Lemma 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_s of size k is connected to a subset in L_s of size at least $k + 1$, for all values of k, where $1 \leq k \leq n$.*

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

We prove the implication 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_s nodes of size k, that is connected to a subset of L_s 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_s nodes mentioned above; otherwise, the k variables from the corresponding k nodes in

U_s 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, we also use contradiction. Assume that any subset of nodes in U_s of size k is connected to another subset of nodes in L_s 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_s that are only connected to $l+1$ nodes in L_s, which contradicts our original assumption, of having any k nodes in U_s connected to at least $k+1$ nodes in L_s, for all values of k, where $1 \leq k \leq n$.■

It can be seen that a naive check for the above condition takes time of order $O(2^n)$ since we must consider all values of k. We will now show 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 Lemma. (see Section III-A, bullet 11 for definition).

Lemma 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_s of size k is connected to a subset in L_s of size at least $k+1$, for all values of k, where $1 \leq k \leq n$.*

Proof. We prove the implication 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_s that are **only** connected to some k other nodes in L_s. 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.

We prove the converse by contradiction. Suppose that any k nodes in U_s are connected to at least $k+1$ nodes in L_s, 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_s that could not be saturated. Assume only one of those links carried n units of flow to a node in U_s say node u. Then node u either has a single outgoing link, or is one of k nodes in U_s, that are connected to another set of k nodes in L_s (otherwise, it would have been able to forward this remaining unit of flow to the sink through an augmenting path on the residual network [17]). In both cases node u will violate the connectivity assumptions. Therefore, the max-flow must be $n(n+1)$. This concludes the proof. ■

IV. Generalizations

In this section we introduce three generalizations. First we discuss the case when $|L_s|$ is larger then $n+1$, and introduce a mixed integer linear program (MILP) to compute the minimum number of L_s that can be used to tolerate a single failure. Then we generalize the sufficient and necessary conditions presented in the previous section to be suitable for any network topology (not necessarily bipartite), and any number of failures.

A. The case of $|L_s| > n+1$

Until now, we have assumed that the number of nodes in L_s is exactly $n+1$. The number of L_s nodes could be larger than $n+1$. This however, does not invalidate our conditions and the above requirements will still apply.

The only difference is that the minimum number of combinations may be more than $n+1$, depending on the topology. One extreme case, is when each of the n sources is connected to exactly two nodes in L_s, and each of the L_s nodes has exactly one neighbor in U_s, i.e., $|L_s| = 2n$. Assuming that the max-flow from L_s to the sink is $2n$, the minimum number of combinations that tolerate a single link failure in this case is $2n$, which is equivalent to 1+1 protection, where coding is not needed and routing can be used.

Another issue that arises in this general case is selecting the appropriate linear combinations that will enable the sink to recover all the original data units. The network shown in Fig. 7 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 of the combinations from A or B, the sink will be able to recover all the original data units. We conjecture that the problem of finding the minimum number of L_s nodes that satisfies the connectivity conditions, and hence, the problem of finding the minimum number of linear combinations that can tolerate a single link failure is NP-Complete. We therefore formulate a solution to this problem as a mixed integer linear program.

We use the graph G^{ST} to formulate an MILP to calculate the minimum number of paths that are needed from nodes in L_s to the sink, in order to forward $n(n+1)$ units of flow. Each node in L_s is traversed by a single path to the sink, and thus calculating the minimum number of paths will result in the minimum number of nodes in L_s that satisfies the connectivity conditions.

1) Notations:

- Let m be the number of nodes in L_s.
- $N_a^b(u)$ is the set of neighbors in a of node u, such that u is in b, where b and $a \in \{U_s, L_s\}$ and if $b \in U_s$, then $a \in L_s$ and vice versa.
- f_{uv} and c_{uv} corresponds to the flow and capacity of edge (u,v) respectively, where $0 \leq f_{uv} \leq c_{uv}$.
- $c_{S'u} = n+1$, $\forall u$ where S' is the hypothetical source and $u \in U_s$. The capacity of all other edges is n.
- y_v is defined for every node v in L_s and it equals the sum of all flows going into that node.
- z_v is a binary variable that is defined for every node v in L_s, which is equal to 1 if the outgoing link from v to the sink carries at least one unit of flow.

2) MILP: We begin with the assumption that the maximum achievable flow from S' to T' is $n(n+1)$. All the nodes in

L_s that participate in forwarding the flow are selected to be the coding nodes. We calculate the minimum number of L_s nodes that satisfies the connectivity conditions by calculating the minimum number of used paths. The objective function is:

$$Minimize \sum_{v=1}^{m} z_v \tag{1}$$

Subject to:

$$z_v - \frac{y_v}{n} \geq 0, \forall v \in L_s \tag{2}$$

This constraint is defined for every node v in L_s, and it sets the binary variable z_v to 1 if there is an outgoing flow from node v to the sink T', i.e., the path from v is used.

$$\sum_{\forall v: v \epsilon N_{L_s}^{U_s}(u)} f_{uv} = n+1, \forall u \in U_s \tag{3}$$

$$y_v - \sum_{\forall u: v \epsilon N_{L_s}^{U_s}(u)} f_{uv} = 0, \forall v \in L_s \tag{4}$$

These two constraints represent the conservation of flow constraints, where the constraint in equation 3 beside conserving the flow assures that the links from the hypothetical source to all the nodes in U_s are saturated to guarantee a max-flow of $n(n+1)$. And, the constraint in equation 4 (combined with the bounds on y_v below) restricts the sum of all incoming flows to a certain node v in L_s not to exceed the capacity of the single outgoing link to the sink. Finally the following two bounds are needed:

$$0 \leq f_{uv} \leq c_{uv}, \forall (u,v) \tag{5}$$

$$0 \leq y_v \leq n, \forall v \tag{6}$$

For $u \ \epsilon \ U_s$ and $v \in L_s$.

B. General Network Topology

Assumption 6 in Section III-A states that the graph induced by the nodes in U_s and L_s is bipartite. This is a restricted topology that may not be present in a real network. Therefore, we address the problem of adapting to other network topologies in this subsection.

By carefully inspecting the condition of Lemma 1, one can see that the essence of the solution lies in the number of edge-disjoint paths (or node-disjoint paths if we are concerned with node-failures) from a group of sources to the sink. In the special case considered previously, each node in L_s represented one such path. Hence, Lemma 1 can be generalized in the following Theorem.

Theorem 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_s 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 Lemma 1. ■

As an example to illustrate Theorem 1, consider a less restricted network topology, where we allow links between sources, as shown in Fig. 8. Table I lists a collection of edge-disjoint paths between every possible combination of sources and the sink. It can be easily verified that the network in Fig. 8 satisfies the condition in Theorem 1. Moreover, this condition can be checked using the same idea in Lemma 2. Specifically, by assuming 1) that each source node is connected to a hypothetical source S' through a link with a capacity of n+1, and 2) that all other links have a capacity of n. Then, checking if the max-flow to the sink is at least $n(n+1)$.

TABLE I
EDGE-DISJOINT PATHS FROM COMBINATIONS OF SOURCES TO SINK IN FIG. 8

Sources	Paths
S_1	$\{S_1\text{-}A\text{-}T\}$,$\{S_1\text{-}S_3\text{-}B\text{-}T\}$
S_2	$\{S_2\text{-}D\text{-}T\}$,$\{S_2\text{-}S_3\text{-}C\text{-}T\}$
S_3	$\{S_3\text{-}B\text{-}T\}$,$\{S_3\text{-}C\text{-}T\}$
$\{S_1, S_2\}$	$\{S_1\text{-}A\text{-}T\}$,$\{S_1\text{-}S_3\text{-}B\text{-}T\}$,$\{S_2\text{-}D\text{-}T\}$
$\{S_1, S_3\}$	$\{S_1\text{-}A\text{-}T\}$,$\{S_1\text{-}S_3\text{-}B\text{-}T\}$,$\{S_3\text{-}C\text{-}T\}$
$\{S_2, S_3\}$	$\{S_2\text{-}D\text{-}T\}$,$\{S_2\text{-}S_3\text{-}C\text{-}T\}$,$\{S_3\text{-}B\text{-}T\}$
$\{S_1, S_2, S_3\}$	$\{S_1\text{-}A\text{-}T\}$,$\{S_2\text{-}D\text{-}T\}$,$\{S_3\text{-}B\text{-}T\}$,$\{S_3\text{-}C\text{-}T\}$

C. Multiple Failures

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

Theorem 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_s 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 Lemma 1. ■

Although we have discussed three generalizations in the previous subsections, in the rest of the paper we continue our analysis of the baseline case that satisfies the assumptions in Section III-A.

V. CODING

In the previous sections, we assumed linear independence between linearly combined data units. In this section, we will show 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.

A linear combination is a summation of data symbols (b_i's) each of which is multiplied by a coefficient (α_i) from a finite field $GF(q)$, as follows: $C = \sum_i \alpha_i . b_i$, where $b_i, \alpha_i \in GF(q)$. The independence of combinations relies on the b_i's and the choice of the α_i's. Therefore, achieving independence using $\{0, 1\}$ coefficients depends solely on how we compose 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(d), the three combinations that were sent to the sink are, $C_1 = b_1$, $C_2 = b_2$ and $C_3 = b_1 + b_2$, each of which is composed only from data units and no coefficients (other than 1 and 0) were used.

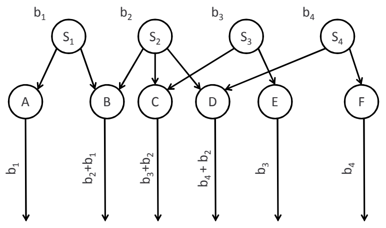


Fig. 7. Minimum combinations = n+2

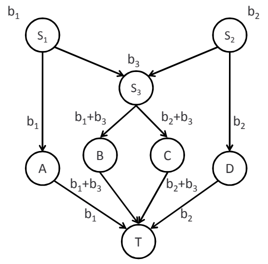


Fig. 8. Non-bipartite topology

In the following subsections we assume 1) that the connectivity condition of Lemma 1 is satisfied; 2) if the graph induced by the nodes in U_s and L_s is not bipartite the transformation in the appendix is used. We now show how to decide on the data units composing each of the linear combinations, through finding simple paths and trees.

A. The benefits of paths and trees

Consider a path in the bipartite graph that has both ends in L_s and in the middle it alternates between U_s and L_s until it includes all the nodes in U_s. It is clear that any k nodes from U_s on that path have at least $k+1$ neighboring nodes from L_s also on that path. Also, note that the number of L_s 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_s.

Such a path not only finds a set of nodes in L_s 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.9, where we have 4 source nodes in U_s, and 5 nodes in L_s. If we let all the sources use all their outgoing links to L_s as shown in the example, 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.

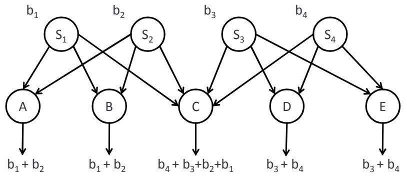


Fig. 9. Bad coding: linear Independence of any n combinations is not satisfied

However, consider the simple path {A, S_1, B, S_2, C, S_3, D, S_4, E} that is represented by the solid edges in Fig.10. If we compose the combinations at the L_s nodes according to their connectivity with the nodes in U_s on the path, i.e., a solid line correspond to a coefficient of 1 and a dashed line correspond to a coefficient of 0, then linear independence will be guaranteed, since any two combinations cannot have more than one element (i.e., data unit) in common.

Of course, we may not always find such a simple path. However, since a path is a special case of a tree with two leaves, then if we can find a tree that covers all the nodes in U_s, with all of its leaf nodes in L_s, we can construct independent

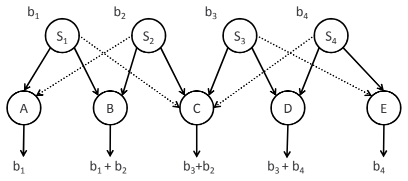
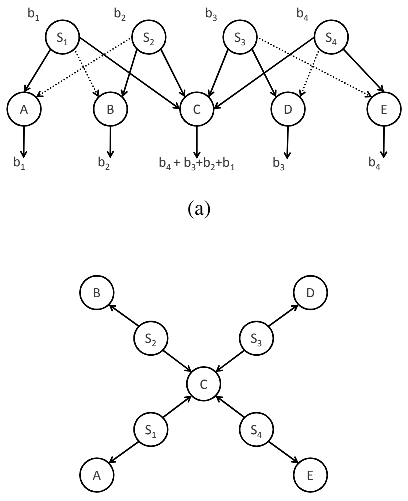


Fig. 10. Path coding: linear independence is satisfied in any n combinations

linear combinations according to the connectivity on the tree. This is shown in Fig.11(a), for the network presented in Fig.9, Fig.11(b) clarifies the underlying tree structure.


(a)

(b)

Fig. 11. (a) Tree coding: linear independence is satisfied in any n linear combinations, (b) Underlying tree

We now prove that constructing the linear combinations at the nodes in L_s according to their connectivity with the source nodes in U_s on the coding tree results in a set of $n+1$ linear combinations such that any n of them are linearly independent.

Theorem 3. *If the linear combinations at the nodes in L_s are created according to their connectivity with the source nodes in U_s 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 $n+1$ linear 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. We make 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_s. 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 (when it is a path), 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 (when it is a path).

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.■

B. Constructing a coding tree

We can construct a coding tree using the following three steps: first we begin by constructing a tree rooted at a node in L_s, then we modify its structure to guarantee that there are no leaves in U_s. Finally we trim the extra L_s leaves if any (when $|L_s| > n+1$).

A tree can be constructed in time of order $O(|E|)$, e.g., a depth-first search (DFS) tree or a breadth-first search (BFS) tree, and the trimming can be done in time of order $O(|V|)$. The non-trivial part is modifying the structure of the tree to guarantee that there are no U_s leaves. Algorithms 1 and 2 describes a procedure to do the modification.

Algorithm 1 Construct a coding tree

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

In Algorithm 1, we search the tree for a leaf node that falls in U_s. Upon finding such a leaf node u, we look for a neighbor x in L_s for node u that is different from the parent of u. We are guaranteed to find such a neighbor x, because each single node in U_s is connected to at least 2 nodes in L_s (Lemma 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_s that has a neighbor w in L_s not on C. Again, we are guaranteed to find such a node

Algorithm 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 $U_s$, 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
```

v that has such a neighbor w, because any k nodes in U_s are connected to at least $k+1$ nodes in L_s, and since the cycle is composed of equal numbers of nodes from both U_s and L_s, then there must be a U_s node on this cycle that has a neighbor in L_s 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**. As an illustration consider the network in Fig.12, the resulting DFS-Tree (Depth First Tree) rooted at node C is shown in Fig.13(a). The nodes that will be found when running Algorithm 1 and two iterations of **ModTree** are shown in Fig.13(b), the cycles and the edges that are marked to be cut are shown in Fig.13(c), and the final result after trimming the extra leaf nodes is shown in Fig.13(d).

There can be at most $n-1$ leaf nodes in U_s, and the recursive call to **ModTree** can be done at most n times. Hence, the running time of Algorithm 1 is of order $O(n^2)$. Note that trimming extra leaf neighbors does not guarantee the minimum number of nodes in L_s. Therefore, to see how well this algorithm performs, in terms of needed number of L_s nodes, we compared it to the MILP presented in Section IV-A, the results are shown in Fig. 14. Each point on the graph corresponds to the average number of needed L_s nodes over 100 random topologies with the same number of nodes in U_s and L_s, where we varied the number of U_s nodes from 2 to 10 while keeping the number of L_s nodes twice as many.

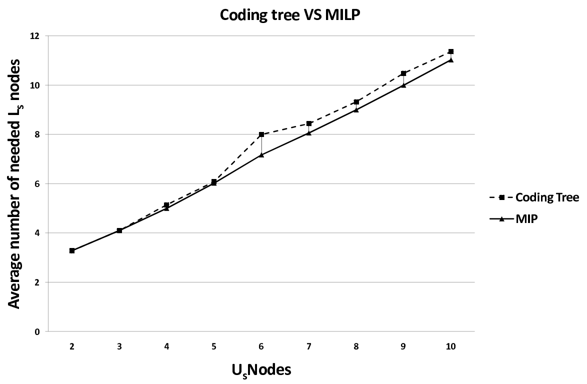


Fig. 14. Tree VS MILP

VI. Practical Considerations

In this section we start by considering networks with limited minimum cuts, where we show how to modify our solution to work even if the minimum cut between the sources and sink was less than $n+1$, and we formulate an MILP to solve this problem. In addition, we show that this problem is NP-

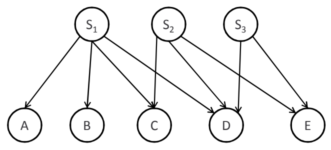


Fig. 12. Network with $n = 3$

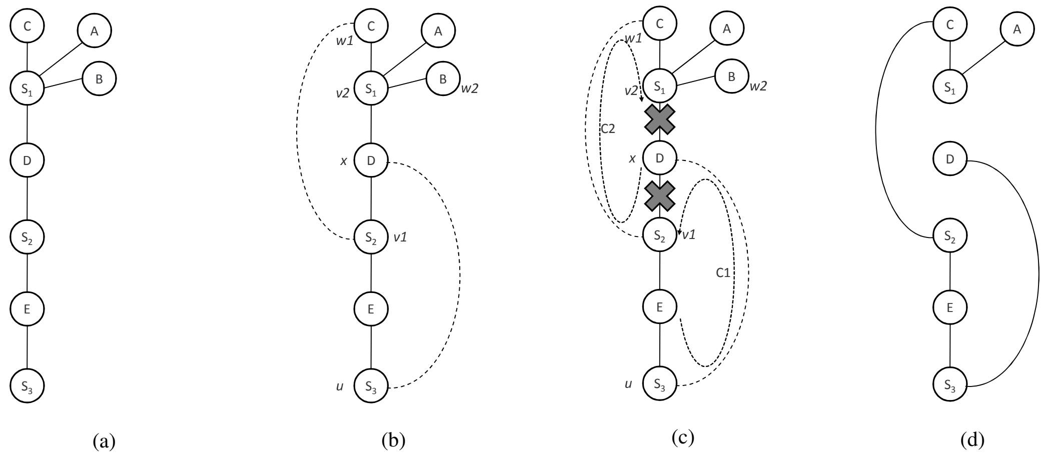


Fig. 13. (a) DFS-Tree, (b) Two iterations, (c) Making the modification, (d) Result

complete by a simple reduction from the K-set cover problem. After that, we discuss the decoding process at the sink and we show that designing the combinations in a optimal manner that allows their fast recovery (according to the decoding process discussed in Section V-A) is an NP-complete problem also. Finally, we show how to make use of our solution in upstream data transmission (from sink to sources).

A. Networks with limited minimum cuts

In our previous discussion, we always assumed that the min-cut between L_s and the sink is greater than or equal to $n+1$. However in practice this may not be the case, since in reality the network gets narrower as we approach the sink. Thus, in this section we study networks with limited min-cuts.

From Menger's theorem [17], the maximum number of edge-disjoint paths between the nodes in L_s and the sink is equal to the minimum edge cut. Let the number of these edge-disjoint paths be h. If h is greater than or equal to $n+1$, then our approach can be applied directly, and the combinations formed in L_s 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 must be modified.

Let us assume that $h < n+1$, then the sink cannot receive more than h combinations at a time. That is, there is no point in allowing more than $h-1$ sources to transmit at the same time if we want to achieve protection using our scheme. Therefore, we propose to 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. We assume that the groups are time multiplexed, and we define a group of sources to be feasible if it satisfies the condition in Theorem 1. We say a set of groups covers all the sources, if each source is present in at least one of the groups in the chosen set.

The way we 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 we will see in Section VII.
2) The used network resources.

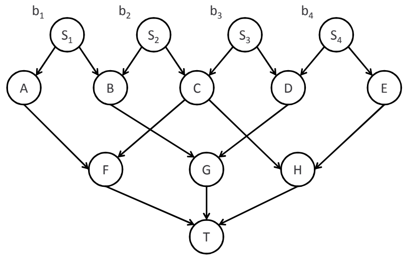


Fig. 15. A network with $h = 3$

As an illustration consider the network in Fig. 15, which contains four sources. The maximum number of edge-disjoint paths (h), from L_s to the sink in this network is equal to 3. Therefor, 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 Theorem 1. Let us now compare the following three sets of groups:

1) Set_1={$\{S_1S_2\}$, $\{S_1S_3\}$, $\{S_1S_4\}$}: In this set S_1 is common in all the groups, which is not a fair solution. This is because, if S_1 was allowed to transmit in all the three time slots, it will be transmitting at a rate of 1 *symbol/time slot*, while each of the remaining sources is allowed to transmit only in one of the three time slots, i.e, transmitting at a rate of $\frac{1}{3}$ *symbol/time slot*.
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 achieves better fairness than Set_1, but also we use less 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 groups 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 as will be shown in the next subsection.

B. Problem Complexity

In this section we show that the problem of dividing the sources into feasible groups, and choosing a covering set from the groups to cover all the source nodes is NP-complete. First, we start by presenting the decision version of the source grouping problem. We will call this problem the **Source-Grouping** problem, then we will consider a simplified version of **Source-Grouping** and prove that it is NP-complete by a reduction from the **K-Set Cover** problem.

Source-Grouping

Given: A graph $G(V, E)$, the set of source nodes U_s, the sink node T, and an integer h, which is equal to the number of edges whose removal disconnects the network, and leaves it divided into two partitions, one containing the sink node T, and the other containing the set of sources U_s.

Question: Is there a collection of at most C groups, where each of the groups is of size $h-1$, such that:

1) All the groups are feasible, i.e. for each group any k sources in this group reaches the sink through a set of $k+1$ edge-disjoint paths, for $1 \leq k \leq h-1$.
2) The groups cover all the source nodes.

Theorem 4. *The **Source-Grouping** problem is NP-complete.*

Proof. This problem belongs to the $\mathcal{NP}$ class, since if we are given a collection of groups, we can check the covering condition in polynomial-time, by calculating the union of all groups and checking if it is equal to U_s. We can also check the feasibility condition for each group in polynomial-time by assuming that each source node has a supply of h units of flow, that needs to be forwarded to the sink node T on the graph edges, where each edge has a capacity equal to $h-1$ (from Lemma 2). This can be accomplished using a max-flow algorithm which has an $O(n^3)$ time complexity (if we use the pre-flow push algorithm [18]), and since there can be at most $n-h+2$ groups the total time will be of order $O(n^3.(n-h+2))$ which is dominated by $O(n^4)$.

To prove the NP-completeness of **Source-Grouping** it is enough to show that part of it is NP-complete. Hence, we will ignore the feasibility condition and we will assume that we are given the set of feasible groups, and our problem is confined to finding a collection of feasible groups that covers U_s. Thus, the new version of **Source-Grouping** which we will call **Source-Covering** can be defined as follows:

Source-Covering

Given: The set of sources U_s, the collection of all the groups in U_s that are feasible, $F = \{G_1, G_2, \dots\}$, where $|G_i| = h-1, \forall i$, and a positive integer C.

Question: Can we choose at most C groups from F whose union gives L_s?

To prove that **Source-Covering** is NP-complete, we will show that any instance of the **K-Set Cover** problem can be mapped directly to an instance of **Source-Covering**. For completeness we state the **K-Set Cover** problem:

K-Set Cover

Given: A set of elements $\mathcal{E} = \{e_1, e_2, \dots, e_n\}$, a collection of subsets of $\mathcal{E}$, $\mathcal{B} = \{B_1, B_2, \dots\}$, where $|B_i| = K$, $\forall i$, and a budget D.

Question: Can we find at most D subsets from $\mathcal{B}$ whose union gives $\mathcal{E}$?

Obviously, the set $\mathcal{E}$ in the **K-Set Cover** problems maps directly to the set U_s in **Source-Covering**. Also $\mathcal{B}$ maps to F, K maps to $h-1$ and D maps to C. Therefore, any instance of the **K-Set Cover** problem can be transformed into an instance of the **Source-Covering** in time of order $O(1)$, and finding a solution to any of them solves the other, which means that **Source-Covering**, and hence **Source-Grouping** are both NP-Complete. ■

C. MILP Formulation

In this subsection we formulate the problem of source grouping as a mixed integer linear program. For convenience, 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.

Assuming that the capacity of all edges is $h-1$, the linear integer program is:

$$Minimize \sum_{c=1}^{M} \sum_{\forall(i,j)\in E} z_{ij}^{c} \tag{7}$$

Subject to:

$$\sum_{\forall j:(s(k),j)\in E} f_{s(k)j}^{kc} = g_c^k.h \ , \ \forall k, c \tag{8}$$

$$z_{ij}^{c} - \frac{\sum_{k=1}^{n} f_{ij}^{kc}}{h-1} \geq 0 \ , \ \forall c, (i,j) \in E. \tag{9}$$

$$\sum_{\forall i:(i,j)\in E} \sum_{k=1}^{n} f_{ij}^{kc} = \sum_{\forall i:(j,i)\in E} \sum_{k=1}^{n} f_{ji}^{kc} \tag{10}$$

$$\forall c, j \neq x \in \{T, s(1), \dots, s(n)\}$$

$$0 \leq \sum_{k=1}^{n} f_{ij}^{kc} \leq h-1 \ , \ \forall c, (i,j) \in E \tag{11}$$

$$\sum_{c=1}^{M} g_c^k = 1 \ , \ \forall k \tag{12}$$

$$\sum_{k=1}^{n} g_c^k \leq h-1 \ , \ \forall c \tag{13}$$

The objective in (7) is to minimize the number of used links for each group. Constraint (8) 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 (9) forces z_{ij}^{c} to be equal to 1 if the flow on edge (i, j) was not 0. Constraint (10) 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 (11) 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 (12) ensures that each source participates in one group only, and (13) 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 (12) the forces each source to participate in one group only. As we will show later in Section VII, a source might have the opportunity to transmit more than once without affecting the throughput of other sources; we call this opportunistic transmission. The MILP can be modified for opportunistic transmissions as follows:

The objective function should be:

$$Minimize \ \sum_{c=1}^{M}(\sum_{k=1}^{n} g_c^k + \sum_{\forall(i,j)\in E} z_{ij}^c) \tag{14}$$

with the following modifications on constraints 12 and 13:

$$\sum_{c=1}^{M} g_c^k \geq 1 \ , \ \forall k \tag{15}$$

$$\sum_{k=1}^{n} g_c^k = h-1 \ , \ \forall c \tag{16}$$

Now the bandwidth is utilized by constraint (16) 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 (15).

D. Implementation

Assumption 8 states that a node can receive from, or transmit to, multiple nodes simultaneously. Practically, this can be through using multiple transceivers utilizing different frequency channels. On the other hand, if we want to remove this assumption completely, time scheduling of node transmissions can be used. It can be shown for some simple cases, that if D is the number of time slots, where D is a function of N and D increases as N increases, that the difference in D between $(1:N)$ protection and network coding protection will be very small. We elaborate more on scheduling in Section VIII.

E. Decoding at the sink node

We showed in Section V that {0,1} coding can be accomplished in $O(n^2)$. In this section we discuss the decoding process at the sink node. We assume that each packet carries the coding vector for the combination in that packet. Which can be done by adding a bit-map of length n in the header of each packet, where a 1 at position i indicates that the data unit from source i participates in this combination, since we use {0,1} coding.

Assuming a single-link failure, the sink is guaranteed to receive at least one trivial combination consisting of a single data unit (and at least two trivial combinations if no failure occurs), which does not need any further processing to retrieve the carried symbol. This is due to the fact that the coding tree

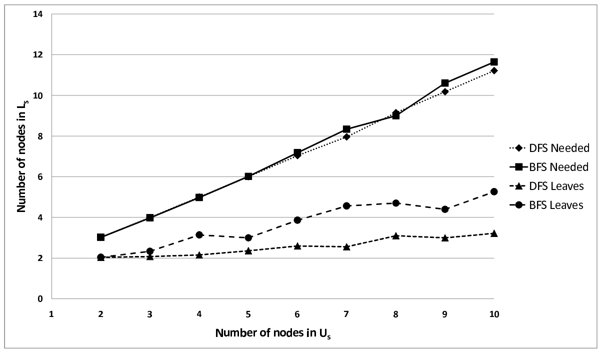


Fig. 16. Comparing the number of produced leaf nodes for a BFS-tree and a DFS-tree

will have at least two leaf nodes in L_s (when the tree is a path), which will produce these single data unit combinations; we refer to these combinations as *leaf combinations*. The details of decoding process were previously discussed in Section V-A, where we showed how to solve the set of linear combinations at the sink using a simple procedure that runs in $O(n)$. Note that when a network has a limited cut h, the decoding time will be of order $O(h)$.

Clearly, it is better to have as many leaves as possible in the coding tree. This is because as the number of leaf combinations increases the running time will decrease. Unfortunately, this is an NP-complete problem and cannot be solved in polynomial time. Note that the coding tree algorithm does not reduce the number of leaf nodes in the initial tree (unless it is trimming unneeded leaves in L_s that will create duplicate trivial combinations), it just modifies the tree to ensure that all the leaf nodes are in L_s. Therefore, to maximize the number of leaf nodes in the coding tree we can start by finding the *maximum leaf spanning tree* on the bipartite graph containing U_s and L_s (which is a known NP-complete problem [19]), then run the coding tree algorithm to ensure that all leaf nodes are in L_s, which will transfer the maximum leaf spanning tree to the maximum leaf coding tree in polynomial time.

To keep the coding simple the initial tree must be constructed in an efficient manner. There are two well known tree search strategies, the DFS-tree (Depth First Search) and the BFS-tree (Breadth First Tree). To compare these two strategies, we ran our algorithm in Section V using both options to see which strategy performs better in terms of producing leaf combinations (leaf nodes). The results are shown in Fig.16, where each point on the graph corresponds to the average of 50 runs using the same number of nodes in U_s and choosing the number of L_s randomly between $|U_s|+1$ and $2|U_s|$. From the figure, we can see that on average BFS produces more leaf combinations than DFS. Although not optimal, in terms of producing maximum number of leaves, we use a BFS-tree as the initial tree in our algorithm for its simplicity where it can be constructed in $O(|E|)$ steps [18].

It is worth mentioning that the decoding process will be simplified if there was a node, say x, in L_s with a degree equal to n. This is because we can send n trivial combinations from nodes other than x, plus a combination carrying the XOR of all data units that is created at node x. An example is shown in Figure 11(a). If this was the case then decoding is only needed if a trivial combination was lost.

F. Sink to sources transmission

To send data to users, the sink can literally reverse the process used by the sources. That is, generate $n+1$ combinations, such that any n of them are linearly independent. Then, forward these combinations to the nodes in L_s so that they can collaboratively recover the original data units and send each of which to its designated receiver (user). One way to do this is by using the same coding vectors for the combinations received from the sources. For example, if the sink received the combination $b_i \oplus b_j$ on path k, then when sending data upstream to sources the data units for sources b_i and b_j are XORed and sent back on the same path k.

To recover the data units the users should collaborate with routers in the following manner. The recovery can start from the L_s nodes that receive single data unit combinations, which correspond to leaf nodes in the coding tree (there are at least two such nodes). These nodes send the received data units to their corresponding users (each leaf node in L_s sends to a single source). By doing this the users have received the data units destined to them. After that the users send these data units to their other neighboring nodes in L_s on the coding tree (since each source has at least two neighbors in L_s). Upon receiving a data unit, a node in L_s removes this data unit from the combination it received from the sink. The L_s node continues to remove data units until the L_s node has received data units from all but one of its U_s neighbors, say y, after which the data unit for user y will be recovered, and then sent to user y. The process repeats until all the original data units are recovered by the user nodes.

The recovery will be successful as long as there is at least one leaf combination that was received successfully by a node in L_s. We are guaranteed to have such a leaf combination when a single-link failure takes place. Basically there are two possibilities:

1) The failure affects a leaf combination: in this case as we discussed above we are guaranteed to have at least one other leaf combination.
2) The failure affects 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.

In the worst case, when the coding tree is a path, and one of the leaf combinations is lost due to a failure, this process takes $O(2n)$ transmissions or $O(2h)$ in networks with limited min-cuts.

VII. Network Performance

Since redundant data is sent to provide protection, the effective data rate will be decreased, compared to the case when there is no protection. In this section, we will study the effect of network coding-based protection on the effective data rate. Specifically, we will discuss 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.

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

1) Sources will be divided into groups, like we did in section VI, if h was not large enough to forward all the $n+1$ combinations in the case of coding, or the n data units when there is no protection.
2) There are two edge-disjoint paths from every source to the sink, i.e., minimum number of neighbors is used. This assumption will simplify the analysis, and will not affect its validity. This is because the rate is affected by the total number of combinations received at the sink, and the number of unique data symbols that can be recovered, but not by the number of occurrences of the data symbols in the combinations.
3) We assume that the selected groups are feasible as defined in section VI.

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

$$\mathcal{R} = n$$

Case 1.b: 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. 17 shows an example, where the sources S_1, S_2, ...,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 2.a: When $h \geq n+1$ the sink can receive the $n+1$ combinations at the same time and recover all the n data units

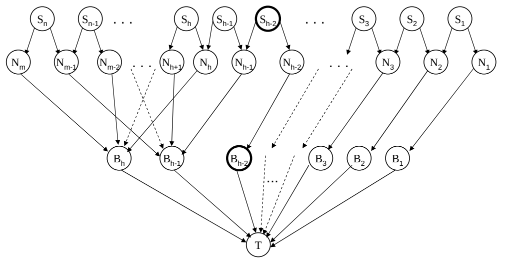


Fig. 17. Worst case grouping

using any n combinations, that is:

$$\mathcal{R} = n$$

Case 2.b: 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 we assume protection is 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 1.b, 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.17 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.

VIII. Scheduling

As mentioned in Subsection VI-D, if each node has a single transceiver, a scheduling mechanism for the sources transmissions should be used. In this section we introduce a greedy algorithm that constructs a feasible schedule for the transmissions of the sources in U_s, taking into consideration their connectivity with their neighbors in L_s in the original graph (see appendix). In the appendix, we discuss the case if the original induced graph was not bipartite and a transformation was applied.

We use this algorithm as a common ground to compare the performance of our protection scheme with the $1:N$, and the $1+1$ protection schemes, in terms of the number of needed time slots. Before stating the algorithm, let us define the feasibility of a schedule for the three protection schemes, assuming that the max-flow between L_s and the sink is h:

For Network Coding-Based protection the following must hold:

- The number of sources in a certain slot does not exceed $h-1$.
- If source x is scheduled in a time slot, then no other source that has a common neighbor with x can be scheduled in the same time slot.

For 1:N protection the following must hold:

- The number of sources in a certain slot does not exceed $h-1$.
- If source x is scheduled in a time slot, then there must be at least *one* neighbor for x, which does not receive from any source other than x in that slot.

For 1+1 protection the following must hold:

- The number of sources in a certain slot does not exceed $h/2$.
- If source x is scheduled in a time slot, then there must be at least *two* neighbors for x, which do not receive from any source other than x in that slot.

Taking the feasibility conditions into account, Algorithm 3 shows how to build a schedule that satisfies those conditions for the case of network coding-based protection (we will discuss the other two cases shortly). For each time slot, the algorithm selects the source with the least degree in U_s. Then, it excludes all the sources that will violate the feasibility conditions from future choices, by putting them in the $Colliding\_Sources$ set. The previous two steps are repeated until no more sources can be added to the current time slot. After that, $Colliding\_Sources$ is reinitialized and the algorithm starts filling the next time slot. The process repeats until all sources are scheduled.

For the case of $1:N$ protection, one slight modification is needed. That is, in step 22 $Colliding\_Sources$ should be modified to include *All the* U_s *neighbors of the one node in* L_s *that is a neighbor to* x*, and has the least degree*.

However, in the case of $1+1$ protection, two modifications are required:

1) The condition of the **if** statement in 11 should be modified to ($index > h/2 \mid\mid Found$ == FALSE).
2) In step 22, $Colliding\_Sources$ should be modified to include *All the* U_s *neighbors of the two nodes in* L_s *that are neighbors to* x*, and have the least degrees*.

To be as practical as possible in our comparison, the sources in U_s should have small degrees. This is because 1) the sources only see the nearby neighbors (routers), which fall within their vicinity, and 2) the routers in reality are not placed too close to each other, so that the maximum amount of users are covered

Algorithm 3 Scheduling Algorithm

```
1: {Defining Variables}
2: $SchdSet$ = $\emptyset$; {a set that contains the scheduled sources}
3: $Colliding\_Sources[|U_s|]$ = $\emptyset$; {a set that contains the sources that will collide with the scheduled source if both transmit at the same time}
4: $Schedule[|U_s|][|U_s|]$; {transmissions schedule}
5: $Slot$ = 0; {the number of needed time slots}
6: $Index$ = 0; {source index in a time slot}
7: $h$ = Max-flow; {the max-flow from $L_s$ to the sink}
8: $Found$ = TRUE; {a boolean variable, which is TRUE if a new source is added}
9: while ($|SchdSet| < |U_s|$) do
10:     $x = \emptyset$;
11:     if ($Index > h - 1$ || $Found$ == FALSE) then
12:         $Slot$++;
13:         $Index$ = 0;
14:         $Colliding\_Sources[|U_s|]$ = $\emptyset$;
15:     end if
16:     $x$ = Select the node in $L_s$ with the least degree, say $u$,such that ($u \notin Colliding\_Sources$) AND ($u \notin SchdSet$);
17:     if ($x$ == $\emptyset$) then
18:         $Found$ = FALSE;
19:     else
20:         $SchdSet = SchedSet \cup x$;
21:         $Schedule[Slot][Index] = x$;
22:         $Colliding\_Sources$ = $Colliding\_Sources$ $\cup$ {All the $U_s$ neighbors of the $L_s$ neighbors of $x$ that are not in $SchdSet$}
23:         $Index$++;
24:     end if
25: end while
```

with the minimum number of routers. In addition to the source degree, the max-flow between L_s and the sink should also be small, since in reality the network gets narrower as we approach the sink.

We compared the three schemes based on the algorithm and the following setup. The cardinality of U_s was varied from 2 to 20 in 19 steps. In each step, we generated 10 different topologies, with the cardinality of L_s being randomly chosen between $|U_s|+1$ and $2|U_s|$, and with connectivity conditions in Lemma 1 satisfied. The algorithm was then executed for the three protection schemes on each of the ten topologies, and the average was taken.

We conducted two experiments with two different values for the max-flow. Specifically, we made h equals $(L_s/4)+1$ and $(L_s/6)+1$, the results are shown in Figures 18 and 19 respectively. It can be seen that the difference between 1:N and network coding is very small, and it shrinks further as the max-flow decreases, i.e., in more practical cases. Obviously, 1+1 protection performance is poor compared to the other two schemes, since it approximately consumes twice the resources used by the 1:N or the network coding-based protection. Thus, fewer sources can be scheduled together.

These results compare the performance of the three schemes when no failure occurs, which is unjust to the 1+1 and network coding-based protection schemes when compared to the 1:N protection. This is because, in the case of a failure, the performance of 1+1 and network coding will not be affected, i.e., no more time slots are required. On the other hand, the performance of 1:N will get worse, because it will consume more time slots to do the rescheduling, which establishes the

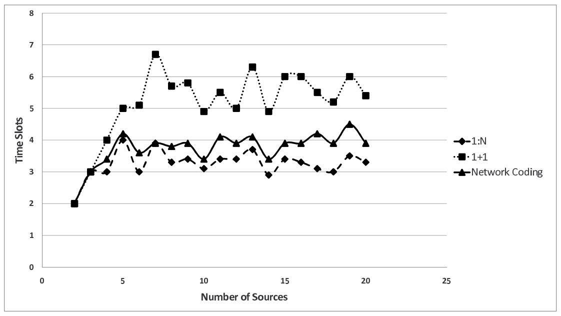


Fig. 18. Max-flow=$(L_s/4)+1$

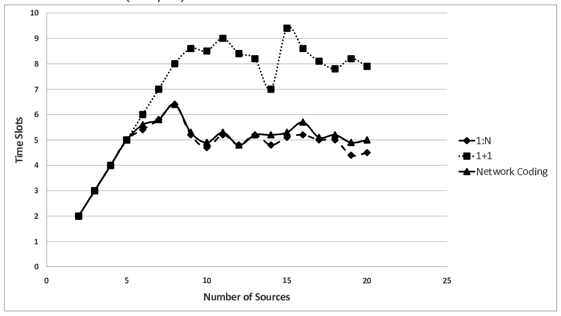


Fig. 19. Max-flow=$(L_s/6)+1$

advantage of network coding-based protection over the other two schemes.

IX. Conclusions

In this paper we presented a novel network coding-based approach that provides protection to many-to-one flows at the speed of proactive protection but at the cost of reactive protection. We derived and proved the necessary and sufficient conditions to achieve this protection for n source nodes. A polynomial-time algorithm was presented to perform the coding with {0,1} coefficients. We considered some of the practical issues related to our approach, such as adapting our scheme to general network topologies. In addition, we studied the effect of network coding on the performance of the network. Finally, we considered the implementation of our approach when each network node has a single transceiver. This was done through a scheduling algorithm, which showed that the performance of our approach in terms of the number of needed time slots, and hence throughput, is comparable to the performance of 1:N when no failures occur, and is better in the case of failures.

Appendix

A. Finding Bipartite equivalent

Creating a feasible set of combinations at the nodes in L_s depends on the data units that can reach these nodes from the sources in U_s. The assumption of a bipartite induced graph in Section III-A accurately captures the data unit reachability to the nodes in L_s. Therefor, if the original graph induced by the nodes in U_s and L_s was not bipartite we need to find a bipartite equivalent.

As mentioned earlier, user nodes do not communicate directly in practice, rather, they use the infrastructure formed by the routers to communicate. Therefore, there is only one case in which the induced graph is not bipartite, and that is when there is a link between two routers in L_s. Such a graph is called semi-bipartite since only one of its partitions can contain both ends of a link. An example is shown in Fig. 20. In a semi-bipartite graph the links from U_s to L_s nodes are not enough to accurately capture the reachability of data units to the routers. For example in Fig. 20 S_1 can reach router C through router node B, but there is no direct link between S_1 and C.

In this section we introduce a simple transformation to find the bipartite equivalent of any semi-bipartite graph, in terms of data unit reachability in a certain level i. Assume that the edge (R_i, R_j) connects router R_i with router R_j, then to remove (R_i, R_j) we add the following logical edges:

1) Add edge (u, R_j), if it does not already exist, for all u such that $(u, R_i) \in E$.
2) Add edge (u, R_i), if it does not already exist, for all u such that $(u, R_j) \in E$.

The transformation of the semi-bipartite subgraph shown in Fig. 20 is illustrated in Fig. 21. Where we represent the ability of users to reach routers through other router nodes by direct links. For example, S_1 is now connected directly to router C. It is worth mentioning that this transformation is not necessary to find a feasible set of combinations if the condition of Lemma 1 is already satisfied. That is, we can just ignore the links between the routers. However, the transformation may help in reducing the needed number of combinations by adding the logical links. For example, the condition in Lemma 1 is satisfied in the network in Fig. 22. Thus, we can ignore the link (D, E), and in this case we need n+2 combination to be able to tolerate a single link failure. However, if the bipartite equivalent is used, only n+1 are needed as shown in Fig. 23. Therefore, it is better to do the transformation whenever the induced graph by the nodes in U_s and L_s is semi-bipartite, since this will allow us to maximize the throughput and reduce the used resources.

Using this transformation, we can always assume in the paper that the induced graph by the nodes in U_s and L_s is bipartite. In the next subsection we discuss how this transformation works with the coding and scheduling mechanisms introduced in Sections III and VIII.

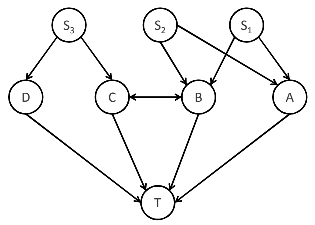


Fig. 20. A semi-bipartite graph

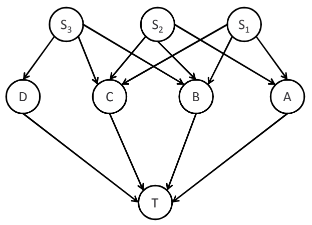


Fig. 21. Transformed graph

B. The effect of transformation on coding and scheduling

In this section we briefly discuss how the transformation introduced in the previous subsection works with the coding and scheduling mechanisms presented in sections III and VIII respectively.

The coding tree algorithm tells each node in L_s how to incorporate data units in the combination originating from it.

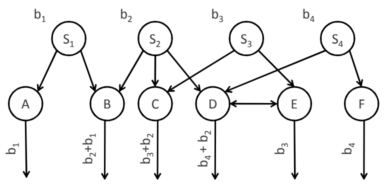


Fig. 22. need n+2 combinations if transformation is ignored

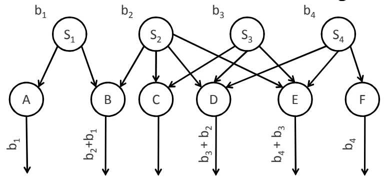


Fig. 23. need n+1 combinations if transformation is done

This is simply known from the tree structure found by the algorithm, where the edges are assigned a coefficient of 1 if they are on the tree and 0 otherwise. It is important to notice that this only works if all the neighbors of a node in L_s were in U_s. Otherwise, if the tree included a link between two nodes in L_s (in a semi-bipartite graph) these two nodes will not be able to correctly decide on the data units to be included. Therefore, the transformation is a must for the coding tree to run properly.

Unlike the coding tree, the scheduling algorithm introduced in Section VIII must not use the transformation and must be executed over the original graph to construct the proper schedule. This is because if the transformed graph is used, then the newly added edges will incorrectly increase the contention between sources. For example, in the graph shown in Fig. 20 the transmission of source S_1 does not reach router C directly, and thus, if the scheduling algorithm used the bipartite equivalent in Fig. 21 it will incorrectly consider C as a neighbor to S_1. Therefore, scheduling must be done using the original graph.

References

[1] G. Parissidis, V. Lenders, M. May, and B. Plattner. Multi-path routing protocols in wireless mobile ad hoc networks: A quantitative comparison. NEW2AN 2006, LNCS 4003, pp. 313 - 326, 2006.

[2] P. Rogers and N. Abu-Ghazaleh. *Algorithms and Protocols for Wireless, Mobile Ad Hoc Networks*, chapter 3. Wiley-IEEE Press, November 10 2008.

[3] A. Srinivas and E. Modiano. Minimum energy disjoint path routing in wireless ad-hoc networks. In the proceedings of the 9th annual international conference on Mobile computing and networking (Mobicom 2003). Pages: 122 - 133.

[4] D. S. Lun, M. Medard, and M. Effros. On coding for reliable communication over packet networks. In the Proceedings of the 42nd Annual Allerton Conference on Communication, Control, and Computing (Allerton 2004), Sept./Oct. 2004, invited paper.

[5] N. Li and J. C. Hou. Flss: A fault-tolerant topology control algorithm for wireless networks. In the proceedings of the 10th annual international conference on Mobile computing and networking (MobiCom 2004). Pages: 275 - 286.

[6] S. El Rouayheb, A. Sprintson, and C. Georghiades. Simple network codes for instantaneous recovery from edge failures in unicast connections. USCD 2006.

[7] J. Tang, G. Xue, and W. Zhang. Energy efficient survivable broadcasting and multicasting in wireless ad hoc networks. In the proceedings of Military Communications Conference, 2004 (MILCOM 2004).Pages:1165 - 1171 Vol. 3.

[8] M. C. Domingo, D. Remondo, and O. Len. A simple routing scheme for improving ad hoc network survivability. In the proceedings of IEEE Global Telecommunications Conference, 2003 (GLOBECOM 2003). Page(s):718 - 723 Vol.2.

[9] Q. Dong, S. Banerjee, M. Adler, and A. Misra. Minimum energy reliable paths using unreliable wireless links. In the proceedings of the 6th ACM international symposium on Mobile ad hoc networking and computing (MobiHoc 2005). Pages: 449 - 459.

[10] S. Park, R. Vedantham, R. Sivakumar, and Ian F. Akyildiz. A scalable approach for reliable downstream data delivery in wireless sensor networks. In the proceedings of the 5th ACM international symposium on Mobile ad hoc networking and computing (MobiHoc 2004). Pages: 78 - 89.

[11] L. Kant and W. Chen. Service survivability in wireless networks via multi-layer self-healing. In the proceedings of IEEE Wireless Communications and Networking Conference (WCNC 2005). Page(s):2446 - 2452 Vol. 4.

[12] N. Rahnavard and F. Fekri. Crbcast: A collaborative rateless scheme for reliable and energy efficient broadcasting in wireless sensor networks. In the proceedings of the fifth international conference on Information processing in sensor networks (IPSN 2006). Pages: 276 - 283.

[13] D. Zhou and S. Subramaniam. Survivability in optical networks. IEEE Network, vol. 14, pp. 1623, Nov./Dec. 2000.

[14] R. Ahlswede, N. Cai, S. R. Li, and R. Yeung. Network information flow. Information Theory, IEEE Transactions on. Volume 46, Issue 4, July 2000 Page(s):1204 - 1216.

[15] R. W. Yeung, S. R. Li, N. Cai, and Z. Zhang. *Network Coding Theory*. now Publishers Inc, 2006.

[16] S. Katti, H. Rahul, W. Hu, D. Katabi, M. Medard, and J. Crowcroft. Xors in the air: Practical wireless network coding. In the proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM 2006). Pages: 243 - 254.

[17] J. Kleinberg and E. Tardos. *Algorithm Design*. Addison Wesley, 2005.

[18] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows*. Prentice Hall, 1993.

[19] P. C. Li and M. Toulouse. Variations of the maximum leaf spanning tree problem for bipartite graphs. Information Processing Letters archive. Volume 97 , Issue 4 (February 2006). Pages: 129 - 132.

Osameh Al-Kofahi received his B.Sc. in electrical/computer engineering from Jordan University of Science and Technology (JUST) in 2002. He is currently working towards his Ph.D. in electrical and computer engineering at Iowa State University. His research has focused on developing network-coding based survivability techniques for multi-hop wireless networks.

Ahmed E. Kamal (S'82-M'87-SM'91) received a B.Sc. (distinction with honors) and an M.Sc. both from Cairo University, Egypt, and an M.A.Sc. and a Ph.D. both from the University of Toronto, Canada, all in Electrical Engineering in 1978, 1980, 1982 and 1986, respectively. He is currently a professor of Electrical and Computer Engineering at Iowa State University. Earlier he held faculty positions in the Department of Computing Science at the University of Alberta, Canada, and the Department of Computer Engineering at Kuwait University, Kuwait. He was also an adjunct professor at the Telecommunications Research Labs, Edmonton, Alberta.

Kamal's research interests include high-performance networks, optical networks, wireless and sensor networks and performance evaluation. He is a senior member of the IEEE, a member of the Association of Computing Machinery, and a registered professional engineer. He was the co-recipient of the 1993 IEE Hartree Premium for papers published in Computers and Control in IEE Proceedings for his paper entitled Study of the Behaviour of Hubnet. He served on the technical program committees of numerous conferences and workshops, was the organizer and co-chair of the first and second Workshops on Traffic Grooming 2004 and 2005, respectively, and was the co-chair of the Technical Program Committees of a number of conferences including the Communications Services Research (CNSR) conference 2006, the Optical Symposium of Broadnets 2006, and the Optical Networks and Systems Symposium of the IEEE Globecom 2007, the 2008 ACS/IEEE International Conference on Computer Systems and Applications (AICCSA-08), and the ACM International Conference on Information Science, Technology and Applications, 2009. He is on the editorial boards of the Computer Networks journal, and the Journal of Communications.