# Diversity Coding in Two-Connected Networks

Péter Babarczi*, János Tapolcai*, Alija Pašić*, Lajos Rónyai[†], Erika R. Bérczi-Kovács[‡], Muriel Médard[§]
*MTA-BME Future Internet Research Group, Budapest University of Technology and Economics (BME), Hungary
[†]Computer and Automation Research Institute Hungarian Academy of Sciences and BME, Hungary
[‡]Department of Operations Research, Eötvös University, Budapest, Hungary
[§]Research Lab. of Electronics, Massachusetts Institute of Technology, Cambridge, MA 02139 USA
*{babarczi, tapolcai, pasic}@tmit.bme.hu,[†]ronyai@sztaki.hu,[‡]koverika@cs.elte.hu,[§]medard@mit.edu

***Abstract*—In this paper we propose a new proactive recovery scheme against single edge failures for unicast connections in transport networks. The new scheme is a generalisation of diversity coding where the source data $AB$ is split into two parts $A$ and $B$ and three data flows $A$, $B$ and their exclusive OR (XOR) $A \oplus B$ are sent along the network between the source and destination node of the connection. By ensuring that two data flows out of the three always operate even if a single edge fails, the source data can be instantaneously recovered at the destination node. In contrast with diversity coding we do not require the three data flows to be routed along three disjoint paths; however, in our scheme a data flow is allowed to split into two parallel segments and later merge back. Thus, our Generalised Diversity Coding (GDC) scheme can be used in sparse but still 2-connected network topologies. Our proof improves an earlier result of network coding, by using purely graph theoretical tool set instead of algebraic argument. In particular we show that when the source data is divided into two parts, robust intra-session network coding against single edge failures is always possible without any in-network algebraic operation. We present linear-time robust code construction algorithms for this practical special case in minimal coding graphs. We further characterise this question, and show that by increasing the number of edge failures and source data parts we lose these desired properties.**

***Index Terms*—robust network coding, diversity coding, single edge failures, instantaneous recovery**

## I. Introduction

Fast recovery of connections from cable cuts is one of the most important requirements in transport networks, as a long disruption of the transport service highly degrades the performance of upper layer protocols, e.g., TCP. Thus, the main goal is to keep transport network failures transparent for the applications by restoring the affected connections rapidly before the upper layer protocols sense and react to it. Although several recovery approaches exist for providing survivable connections [1]–[4], most of them are reactive schemes (e.g., shared protection approaches), where the traffic is sent along the secondary path(s) only after reacting to a failure on the primary path. They provide excellent network resource utilization, but also rely on excessive control plane signaling which may increase restoration time in large networks to an unacceptable level. Hence, in practice proactive schemes are preferred by network operators, where the traffic is sent simultaneously along the primary and secondary paths, called dedicated protection. Although these proactive schemes suffer from high resource utilization, they provide *instantaneous recovery* [5], [6], as switching matrix configurations remain unchanged after a failure occurs. Thus, signaling is completely eliminated from the recovery process of the disrupted connections, i.e., no flow rerouting or packet retransmission is required upon an edge failure, clearly satisfying the strict recovery time requirement of transport networks.

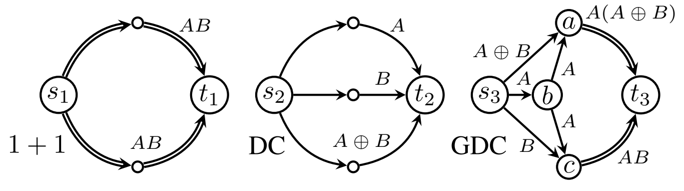


Fig. 1. Illustrative example of protection methods for instantaneous recovery. Duplicated edges forward the whole source data, while simple edges forward only the half of it.

On one hand, among these single edge failure resilient [7] dedicated protection approaches $1+1$ *protection* [8] is favored for its simplicity, because the same data flow $AB$ is sent along two edge-disjoint paths, and only a single-ended switching is required at the destination node upon failure of the primary path. As a result, this approach suffers from high capacity requirement, which often called 100% redundancy, as the secondary path carries only redundant data. Fig. 1 shows an example of $1 + 1$ protection between nodes $s_1$ and $t_1$. On the other hand, in *diversity coding* (DC) [9]–[11] the source data $AB$ is split into two parts $A$ and $B$, and these parts together with their exclusive OR (XOR) $A \oplus B$ are sent along three edge-disjoint paths (see also the connection between $s_2$ and $t_2$ in Fig. 1). Note that, the two source data parts can be instantaneously recovered from arbitrary two of the three flows with XOR coding at the destination node. Although DC – with so called 50% redundancy – is one of the most capacity efficient protection approaches in transport networks which provides instantaneous recovery [6], real network topologies are rarely 3-edge-connected [12], [13] (referred to as 3-connected hereafter), which makes its application hard in practice.

It is important to observe that DC restricts algebraic operations (i.e., XOR coding and decoding) only to the end-nodes of the connection, e.g., host computers or edge routers. However, it was demonstrated that if every node can perform involved algebraic operations (*network coding*), then the design space can be fully explored [6], [14], [15], as network coding remedies both the high capacity requirement and strict topology constraint of previous approaches. For example, between nodes $s_3$ and $t_3$ in Fig. 1 neither a disjoint path-pair on duplicated edges for $1+1$, nor three edge-disjoint paths for DC exist. Luckily, with the application of *robust network codes* – where the network code remains unchanged even when some edges fail – instantaneous recovery can be ensured even for $s_3$ and $t_3$ [16], [17]. For decodability at the destination node, it is necessary and sufficient to ensure that after any single edge failure the flow value between the source and destination node is at least as large as the bandwidth requirement of the connection, i.e., the connection is fault-tolerant. For example, the connection between $s_3$ and $t_3$ in Fig. 1 has flow value at least two, regardless if a duplicated or a simple edge fails, thus, a robust network code exists which ensures instantaneous recovery [16].

From the above example it is clear that theoretically network coding is required to reach larger flexibility and minimal capacity, but from a practical point of view upgrading all core switches with coding capabilities is not realistic at this moment [18], [19]. The main contribution of this paper is filling this gap between theory and practice by proposing a robust network coding method, called *Generalised Diversity Coding (GDC)*, which similar to DC performs simple XOR coding only at the communication end-nodes while instantaneous recovery is maintained for all types of fault-tolerant connections even in 2-connected topologies [20]. To be specific, the *robust network code of GDC consists of three end-to-end directed acyclic graphs* (called *routing DAGs*) carrying the two parts of the source data $A$, $B$ and redundant data $A \oplus B$, respectively. For example, for the connection between $s_3$ and $t_3$ the routing DAGs are $E_{A\oplus B} : s_3 \to a \to t_3$, $E_B : s_3 \to c \to t_3$ and $E_A : s_3 \to b \to \substack{a \\ c} \to t_3$, where at node $b$ the same data is duplicated and sent along two edges $b \to a$ and $b \to c$, while one of the copies (or the one operating upon failure) is selected at $t_3$ arriving on $a \to t_3$ and $c \to t_3$. Following a graph-theoretic approach, we show that such routing DAG decomposition (i.e., robust network code) always exists for two data parts if the connection is fault-tolerant.

The rest of the paper is organized as follows. In Section II we enumerate the related work on robust network coding. Section III introduces GDC and the preliminaries of the study. Section IV presents our main theorem proving that for GDC no in-network coding is required. In Section V we provide algorithms which work *linear time* in the input size for finding the three routing DAGs (i.e., robust network code). It is demonstrated in Section VI that this simplicity of robust network codes exists only for this particular special case, i.e., single edge failures and two data parts. Section VII provides our simulation results, while Section VIII concludes the paper.

## II. Related Work

Suurballe's algorithm [21] provides an edge-disjoint path-pair for $1+1$ protection – the simplest and most widespread instantaneous recovery approach – in polynomial-time, which makes it resilient against single edge failures[1]. $1+1$ does not need any further code construction for instantaneous recovery, but it allocates an excessive amount of redundant capacity (at least $100\%$ compared to a single shortest path). Diversity coding (DC) [5], [6], [9] is a promising approach to reduce the high redundancy of $1+1$ (from 100% to 50%) by applying XOR coding at the source. Although core network nodes are intact and only simple coding and decoding is required at the end-nodes, it is beneficial only in transport networks where even a third edge-disjoint path exists with a reasonable cost. However, this is rarely the case.

The importance of network coding in reducing the capacity consumption of $1+1$ in transport networks has been demonstrated in several studies [5], [6], [10], [15], [22] in different network layers (e.g., optical [23], [24], electronic domain [18], SDN [25], etc.). Most of these works assume that a matrix of traffic demands are given in advance [6], [15], [22], [23], thus, *inter-session network coding* can be applied on the data of different connections. However, recent transport network trends point towards that connection demands are arriving one after another without any knowledge of future incoming requests. Thus, in these dynamic environments *intra-session network coding* is needed for unicast connections, where coding is performed on different parts of the same source data [5], [24], [25]. Hence, each connection can be deployed and released independently, while the routing of the other connections remain intact.

In network coding based approaches, after calculating and reserving the capacity for the connection (i.e., compute the *coding graph*), in a second phase robust network codes [16], [17] – where no change in the coding behavior is required after a failure occurs – have to be constructed in order to ensure instantaneous recovery. In [24] a Linear Program (LP) formulation was presented, which provides coding graphs resilient against single edge failures with minimal capacity. Although it can serve as theoretical lower bound for instantaneous recovery approaches, from a practical point of view it is not applicable in transport networks as in the resulting coding graph the source data might be split into arbitrarily many parts. Furthermore, the required field size for the robust network codes could be in the range of $O(|E|)$ [16], [17].

In [25] General Dedicated Protection with Network Coding (GDP-NC) was proposed, which made a step closer to get a practical approach in transport networks by limiting the number of data parts to two; however, for the price of an NP-hard optimization problem to obtain the minimal capacity coding graph against multiple failures. However, for the special case of single edge failures polynomial-time capacity allocation algorithm exists [11]. In [5] the authors made a further step, and showed that for this practical special case – single edge failure resilience and two data parts – robust network codes exist and can be constructed in $O(|V|^2)$ time over $GF(2)$, i.e., simple XOR coding is sufficient instead of involved algebraic

[1]Note that, single edge failures are the most dominant failure events in transport networks [7].

operations over a field of size $O(|E|)$. However, this approach still requires the upgrade of the core network with XOR coding capabilities [18], [19].

## A. Our Contribution

In this paper we make the following steps towards a practical transport network coding scenario:

- The algebraic argument of [16] is a powerful tool for modelling and solving network coding problems. The work by Rouayheb et al. [5] identifies the graph theoretical nature of the investigated scenario but still focuses on algebraic properties, such as reducing the field size to $GF(2)$. Our insight was to revisit the problem with pure graph theoretical mindset, and show that *in-network coding is actually not required at all.*
- The coding algorithm in [5] runs in $O(|V|^2)$ time, because of the computation time of testing the minimality of the coding graph. Roughly speaking, minimality is important only to have a "simple" coding graph which can be efficiently handled. To improve the running time we use a less strict criteria than minimality[2], which can be verified in linear time and still results a sufficiently "simple" coding graph. As a result, *robust network codes can be constructed in $O(|V|+|E|)$ time for fault-tolerant coding graphs.*
- The above simplicity and efficiency makes GDC a viable approach to provide instantaneous recovery. We further demonstrate that the minimal capacity coding graphs of GDC can *approach the theoretical minimum of capacity requirement even in 2-connected networks*, where DC already fails for several connection requests. The trick is that the three data flows $A$, $B$ and $A \oplus B$ are routed on three end-to-end routing DAGs instead of disjoint end-to-end paths.

## III. Problem Formulation and Preliminaries

Let $\mathcal{G} = (V, E)$ be a directed *coding graph* representing the routes of the data flows of a unicast connection, where $V$ is the set of nodes and $E$ is the set of edges, and there are two distinguished nodes $s \in V$ and $t \in V$, the source and the destination node of the connection, respectively. Each edge has a *unit capacity*, thus, *multiple (parallel) edges* are allowed between two nodes representing communication channels with higher capacity. Note that, a single failure (e.g., cable cut) disrupts the whole communications channel, i.e., all parallel edges between the two nodes fail. We assume that the source data (without loss of generality with sending rate of two units[3]) can be split into two parts of equal (unit) size, denoted as $A$ and $B$.

**Definition 1.** *We call a coding graph $\mathcal{G} = (V, E)$* ***fault-tolerant*** *if there is an $s - t$ flow of value at least 2, even if a single failure occurs (i.e., the corresponding edge or parallel edges are removed from $\mathcal{G}$). We call a fault-tolerant graph $\mathcal{G}$* ***minimal****, if after removal of a single edge (either from a single edge or multiple parallel edges) the resulting graph is no longer fault-tolerant.*

See Fig. 2a as an example for a minimal coding graph. An example of a fault-tolerant, but not minimal graph can be obtained by adding any directed edge to Fig. 2a. We assume that a fault-tolerant coding graph $\mathcal{G}$ is given as the input of the GDC framework. Furhter note, that an important structural property of a minimal coding graph is that there are at most two (parallel) edges between the same pair of nodes [20], referred to as ***biedges*** in the rest of the paper. Thus, if $\mathcal{G}$ is minimal it has edges and biedges only.

Our goal is to *create a robust network code in a minimal coding graph $\mathcal{G}$* when source data can be split into two parts in order to survive any single edge failure, using simple network coding operations (XOR) at the end-nodes while coding is avoided in the core. These requirements essentially boil down to finding three disjoint edge-sets (***routing DAGs***) in graph $\mathcal{G}$ for data flows $A$ (along edges $E_1$), $B$ (along $E_2$) and redundant data $A \oplus B$ (along $E_3$) from $s$ to $t$, i.e., decompose the minimal coding graph into routing DAGs, from which at least two remain $s-t$ connected (refer to Fig. 2) after a single edge failure to ensure decodability. Although coding is avoided at intermediate nodes, two additional network functions are required in the core. The first one is called ***splitter*** node, which duplicates the packets arriving on its incoming edge, and sends the two copies on different outgoing edges ($p$ in Fig. 2a). The second one is called ***merger*** node, which is able to switch (in a failureless state) between two identical copies of data and forward only one of them on its outgoing edge ($m$ in Fig. 2a). After a failure occurs, the merger forwards the intact signal on its outgoing edge [25], [26].

For simpler arguments in the proofs, we follow the idea from [5] and introduce an auxiliary graph called the ***reduced capacity graph*** $\bar{\mathcal{G}} = (V, \bar{E}, \bar{c})$. The node set of $\bar{\mathcal{G}}$ is the same as the node set of $\mathcal{G}$. There are only simple edges in $\bar{E}$, each corresponds to a simple edge or to a biedge in $E$. Furthermore, in $\bar{\mathcal{G}}$ each edge $e \in \bar{E}$ is assigned with a capacity function $\bar{c}(e)$, called reduced capacity, depending whether it was a simple edge ($\bar{c}(e) = 1$) or it was a biedge ($\bar{c}(e) = 1.5$) in the coding graph $\mathcal{G}$ (shown in Fig. 2b).

Finally, we define a *cut* $C \subseteq V$ as a set of nodes that contains $s$ but does not contain $t$. The edges of a cut $C$ are the edges from $C$ to $V \setminus C$.

**Definition 2.** *A cut $C$ in $\mathcal{G}$ is called 2****-biedge-cut*** *if it consists of two biedges from $C$ to $V \setminus C$. A cut $C$ in $\mathcal{G}$ is called* ***3-edge-cut*** *if it consists of three edges from $C$ to $V \setminus C$.*

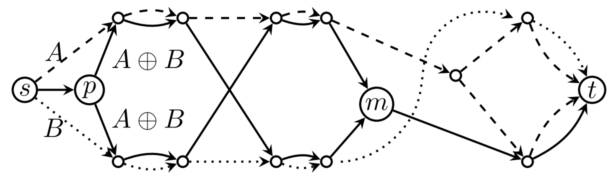
Note that both 2-biedge-cut and 3-edge-cut has a cut value of 3 in the reduced capacity graph $\bar{\mathcal{G}}$.

## A. Preliminaries

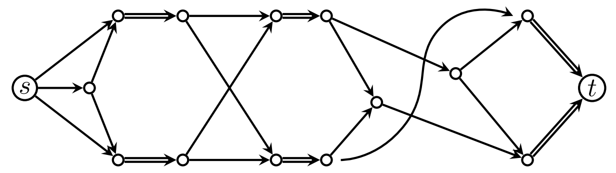
Here we summarise previous results on robust code construction for single edge failure resilience and two data parts,

[2]Properties 1-3 in Sec. V-A.

[3]As each demand is routed independently, the capacities of the coding graph can be scaled accordingly to the source's sending rate.



(a) A minimal coding graph and the robust network code (i.e., routing DAGs) on it: edge-set $E_1$ is shown with broken, $E_2$ with dotted, and $E_3$ with solid lines.



(b) The reduced capacity graph $\bar{\mathcal{G}} = (V, \bar{E}, \bar{c})$ is defined as follows: the biedges in $\mathcal{G}$ have $\bar{c}(e) = 1.5$ unit of capacity (duplicated edges in the figure), while simple edges have $\bar{c}(e) = 1$.

Fig. 2. A minimal coding graph $\mathcal{G}$ and its corresponding reduced capacity graph $\bar{\mathcal{G}}$.

and discuss the relationship between the coding and reduced capacity graphs.

**Theorem 1.** *[5, Theorem 2] $\mathcal{G}$ is fault-tolerant if and only if $\bar{\mathcal{G}}$ has an $s - t$ flow of value at least 3.*

Theorem 1 gives an elegant characterisation of fault-tolerant coding graphs, which is very useful both for theoretical and algorithmic purposes.

**Lemma 1.** *If $\mathcal{G}$ is minimal, then a maximum $s - t$ flow in $\bar{\mathcal{G}}$ has value exactly 3.*

*Proof:* $\mathcal{G}$ is fault-tolerant, hence by Theorem 1 $\bar{\mathcal{G}}$ has a flow of value at least 3. It suffices to show that it cannot have a flow with higher value. Indeed, otherwise by the Ford-Fulkerson theorem the capacity of minimal $s - t$ cuts in $\bar{\mathcal{G}}$ would be at least 3.5. Consider one such cut, and let $e$ be an edge of the cut with $\bar{c}(e) = 1.5$. We can then decrease the capacity of $e$ to $\bar{c}(e) = 1$ and still have a fault-tolerant coding graph by Theorem 1. If minimal cuts do not contain edges with reduced capacity 1.5 then every cut has capacity at least 4 in $\bar{\mathcal{G}}$. Hence, an arbitrary edge can be removed while still retaining fault-tolerance. ■

**Lemma 2.** *[5, Lemma 3] Suppose that $\mathcal{G}$ is minimal. Then $\bar{\mathcal{G}}$ has a maximum flow whose value on the edges of $\bar{\mathcal{G}}$ is from $\{0.5,\ 1,\ 1.5\}$. In particular no edge can have flow value 0.*

Because of the integrality property [27, Theorem 9.10] there exists such a flow with respect to the reduced capacity function $\bar{c}$. Furthermore, an edge with flow value 1.5 in $\bar{\mathcal{G}}$ corresponds to a biedge carrying two data flows in $\mathcal{G}$, while edges with 0.5 and 1 are simple edges in $\mathcal{G}$ forwarding a single data flow. Based on Lemma 1 and Lemma 2, we have the following:

**Corollary 1.** *Suppose that $\mathcal{G}$ is minimal, and $C$ is a minimal cut in $\bar{\mathcal{G}}$. Then cut $C$ is either a 2-biedge-cut or a 3-edge-cut in $\mathcal{G}$.*

**Lemma 3.** *Suppose that $\mathcal{G}$ is minimal, and $e$ is a biedge. Then $e$ is part of a 2-biedge-cut.*

*Proof:* It suffices to show that $e$ is a part of a minimal cut in $\bar{\mathcal{G}}$. This holds because otherwise we could decrease $\bar{c}(e)$ to 1 while still having fault-tolerance. ■

**Lemma 4.** *[5, Proposition 4] Suppose that $\mathcal{G}$ is minimal. Then $\mathcal{G}$ is a directed acyclic graph (DAG).*

Armed with these results, we are ready to present our main structure theorem in Section IV.

## IV. Routing DAG Decomposition Theorem

Building on the definitions and preliminaries of Section III, here we prove the existence of three routing DAGs in a minimal coding graph $\mathcal{G}$ using simple flow techniques, which shows feasibility of GDC [20]. We formulate the routing DAG decomposition theorem which provides the robust network code for GDC as follows:

**Theorem 2.** *Suppose that $\mathcal{G}$ is minimal*[4]*. Then there are disjoint edge-sets (actually DAGs) $E_1, E_2, E_3$ of $\mathcal{G}$ such that after removing an arbitrary edge or biedge of $\mathcal{G}$ at least two edge-sets $E_i$ connect $s$ to $t$.*

If Theorem 2 holds, source data can be reconstructed with XOR coding at the destination from the remaining two data flows without any coding or routing change in the network, i.e., instantaneous recovery is maintained.

*Proof of Theorem 2:* Let $C_1, C_2, \ldots, C_k$ denote a maximal chain[5]

$$C_1 \subset C_2 \subset \cdots \subset C_k$$

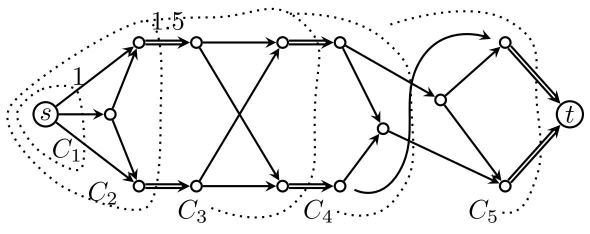
of minimum $s - t$ cuts in $\bar{\mathcal{G}}$, where $k$ is the number of cuts in the chain and $k$ is maximal. See Fig. 3 as an example, and also Section V-A for the algorithm finding such chain of cuts.

For easier understanding, we will use a set of ***coding subgraphs***, denoted by $\mathcal{G}_1, \ldots, \mathcal{G}_{k-1}$. Each $\mathcal{G}_i = (V^i, E^i)$ has a source node $s_i$ and a destination node $t_i$ besides the nodes of $C_{i+1} \setminus C_i$, for $i = 1, \ldots, k-1$. Please note, that $C_1 = \{s\}$, $C_k = V \setminus \{t\}$. The edges of $\mathcal{G}_i$ are the edges among the nodes $C_{i+1} \setminus C_i$ and the edges of $C_i$ and $C_{i+1}$. However, the source nodes of the edges of $C_i$ are replaced by the common source node $s_i$. Also, the target nodes of the edges of $C_{i+1}$ is replaced by the common destination node $t_i$. Note that there may be edges connecting $s_i$ to $t_i$. See also Fig. 3b for the transformation.

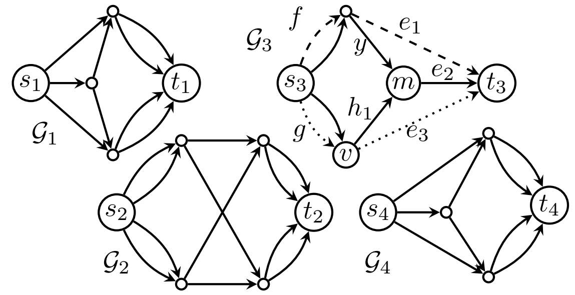
Next, for each graph $\mathcal{G}_i$ we define 3 disjoint edge sets $E^i_1, E^i_2, E^i_3$, such that every $E^i_1, E^i_2, E^i_3$ carries 1 unit of flow from $s_i$ to $t_i$. We next show that the edge sets $E^i_1, E^i_2, E^i_3$ are *fault-tolerant* in the sense that after removing an edge or biedge of $\mathcal{G}_i$, at least two of the $E^i_1, E^i_2, E^i_3$ still connect $s_i$ to

[4]For the sake of simplicity we are proving the theorem for minimal graphs. However, as every fault-tolerant graph contains a minimal one, Theorem 2 can be applied to any fault-tolerant coding graph, by first removing some edges to make it minimal.

[5]This chain might not be unique.



(a) Maximal chain of minimum $s-t$ cuts consists of $k=5$ cuts in $\bar{\mathcal{G}}$.



(b) The corresponding coding subgraphs $\mathcal{G}_i$ for $i=1,\ldots,k-1$.

Fig. 3. Illustrative example for the proof of Theorem 2.

$t_i$. Finally we indicate how to piece together $E_1, E_2, E_3$ from the local pieces $E^i_j$.

Our argument takes $\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_{k-1}$ one by one, and proves the existence of a fault-tolerant solution for each. According to Corollary 1, $C_i$ is either a 3-edge-cut or a 2-biedge-cut in $\mathcal{G}$, therefore for $\mathcal{G}_i$ we have four cases to consider:

| | |
|---|---|
| Type (i) | Both $C_i$ and $C_{i+1}$ are 3-edge-cuts. |
| Type (ii) | Both $C_i$ and $C_{i+1}$ are 2-biedge-cuts. |
| Type (iii) | $C_i$ is 2-biedge-cut and $C_{i+1}$ is 3-edge-cut. |
| Type (iv) | $C_i$ is 3-edge-cut and $C_{i+1}$ is 2-biedge-cut. |

We note beforehand that every graph $\mathcal{G}_i$ inherits a flow of value 3 from $\bar{\mathcal{G}}$, and they have only two minimum cuts with respect to the reduced capacities, namely $\{s_i\}$ and $\{s_i\} \cup C_{i+1} \setminus C_i$ (as we have a maximal chain of minimum cuts). Also they are minimal coding graphs (otherwise $\mathcal{G}$ would not be minimal). These together with Lemma 3 imply that except for the edges incident to $s_i$ and $t_i$ they can not have edges corresponding to edges with reduced capacity $\bar{c}(e) = 1.5$ (i.e., there will be no biedges in the interior of $\mathcal{G}_i$).

We remark also that the three edges of a 3-edge-cut $C_i$ must be in one-to-one correspondence with $E^i_1, E^i_2, E^i_3$, whereas for a 2-biedge-cut we must have a *dominating edge set* (i.e., which has an edge from both biedges of $\mathcal{G}$). This dominating edge set will always be denoted by $E^i_1$. We now consider Types (i)-(iv) one by one. The only complicated cases are Type (iii) and Type (iv).

In Type (i), $\mathcal{G}_i$ can not have edges corresponding to reduced capacity 1.5 hence the reduced capacities in $\mathcal{G}_i$ are integral. This implies that there are three edge disjoint paths $P_1, P_2, P_3$ from $s_i$ to $t_i$ in $\mathcal{G}_i$ and these 3 paths can be set as $E^i_1, E^i_2, E^i_3$.

For Type (ii) there must be four edge disjoint paths $P_1, P_2, P_3, P_4$ between $s_i$ to $t_i$, because the value of minimal cuts in coding subgraph $\mathcal{G}_i$ is 4, and the edges in $C_{i+1} \setminus C_i$ all have $\bar{c}(e) = 1$ in this setting by Lemma 3. We intend to form $E^i_1, E^i_2, E^i_3$ by taking the union of a suitably chosen path pair $P_a, P_b$, $(a \neq b)$ as $E^i_1$, and $E^i_2, E^i_3$ will be the remaining two paths. We have to make sure that for the selected pair $(a, b)$ $P_a$ and $P_b$ do not contain the two edges of the same biedge, because in this case $E^i_2, E^i_3$ has to traverse the other biedge in the 2-biedge-cut, and the failure of the latter biedge would disconnect both $E^i_2, E^i_3$. There are at most 4 such edges, the edges of $C_i$ and $C_{i+1}$, and one such edge rules out at most one pair $(a, b)$. Thus, among the 6 possible path-pairs there will be a suitable one.

For Type (iii) the biedges $f, g$ are adjacent to $s_i$, and $e_1, e_2, e_3$ are the edges adjacent to $t_i$. See also $\mathcal{G}_3$ on Fig. 3b for illustration. There are three edge-disjoint routing DAGs in a Type (iii) coding subgraph (see Lemma 5 and its proof in the Appendix), such that two of them are simple paths ($E^i_2$ traverses $f$ and $e_x$, and $E^i_3$ passes through $g$ and $e_y$, where $x \neq y$), while the third routing DAG is composed of three path segments ($s_i \to m$ traversing edge $f$, $s_i \to m$ containing edge $g$, and $m \to t_i$ passing through $e_z$, where $m$ is a merger node, $z \neq y$ and $z \neq x$), which will be the dominant edge set as $E^i_1$.

Type (iv) is similar to Type (iii), essentially the argument of Lemma 5 works with a (reverse) flow of value 3 from $t_i$ to $s_i$. This finishes the local parts of the construction.

We claim next that the edge sets $E^i_j$ and $E^{i+1}_j$ $(j = 1, 2, 3)$ can be meaningfully glued together to obtain $E_1, E_2$ and $E_3$. This is done as follows: an edge of the form $(s_i, v)$ or $(v, t_i)$ is replaced by the respective edges of $\mathcal{G}$ they are obtained from. There is one ambiguity here when the cut is a 2-biedge-cut with edges $f$ and $g$. Then $E^i_1$ and $E^{i+1}_1$ are joined along an edge from $f$ and along an edge from $g$. By doing this at every $i$, finally we have 3 edge-disjoint edge-sets $E_1, E_2, E_3$ in $\mathcal{G}$, which connect $s$ to $t$. This finishes the proof. ■

Theorem 2 proves that to *find a robust network code in a minimal coding graph* $\mathcal{G}$ against single edge failures and two data parts *is equivalent with finding three routing DAGs in* $\mathcal{G}$, where none of the $E_j$ edge sets contain biedges. Built on this observation, we improve the $O(|V|^2)$ time complexity of previous XOR coding based robust code construction of [5] to linear in Section V.

## V. Linear-Time Robust Code Construction Algorithms

In this section we present linear time robust network code construction (i.e., routing DAG decomposition) algorithms for the GDC problem. In Section V-A we use the construction of the proof in Section IV to obtain the three routing DAGs in fault-tolerant coding graphs. In Section V-B a more direct and more intuitive algorithm is introduced for minimal coding graphs, which highly builds on the properties of the investigated scenario.

### A. Routing DAG Decomposition of Feasible Graphs

A closer inspection of the proof of Theorem 2 reveals that we do not use in full force the fact that $\mathcal{G}$ is minimal. In fact, the following three *properties* are sufficient to carry out the construction of the edge-sets $E_1, E_2, E_3$ in $\mathcal{G}$:

1) $\mathcal{G}$ is a fault-tolerant coding graph such that $\bar{\mathcal{G}}$ has a maximum flow of value 3.
2) The maximum flow has a value 0.5, or 1 or 1.5 for every edge of $\bar{\mathcal{G}}$.
3) We have a maximal chain $C_1 \subset C_2 \subset \cdots \subset C_k$ of minimum $s-t$ cuts in $\bar{\mathcal{G}}$ in such a way that there is no edge $e=(u,v)$ of capacity $\bar{c}(e)=1.5$ with both $u$ and $v$ belonging to $C_{j+1} \setminus C_j$ for some $j$.

We claim that from arbitrary fault-tolerant $\mathcal{G}$ we can construct a subnetwork with Properties 1-3 in *linear time*. Thus, an initial and expensive (quadratic time, to be specific) computation of a minimal coding graph can be dispensed with. The algorithm is built on the following two observations.

- First, *finding a fixed number of edge-disjoint paths can be done in $O(|V|+|E|)$ time*. It is because, the augmenting paths in the residual graph can be found with breadth-first search (BFS) [27, Chap. 3][6]. By always selecting the minimum hop augmenting path we can achieve the graph supporting the flow is actually a DAG.
- Second, *finding a maximal chain of minimum $s-t$ cuts $C_i$ in $\bar{\mathcal{G}}$ can be done in linear time*, if the max flow of $\bar{\mathcal{G}}$ has value 3. Let $\bar{\mathcal{G}}_r$ denote the residual graph $\bar{\mathcal{G}}$ built up by the Ford-Fulkerson algorithm, after the flow of value 3 is found. Let $S_1, S_2, \cdots, S_k$ be the strongly connected components of the residual graph, listed in an order reverse to the topological ordering of the components. Then there may be an edge from $S_i$ to $S_j$ with $i \neq j$ only if $j < i$. This implies that $C_i = \cup_{j=1}^{i} S_j$, $i=1,\ldots,k-1$ is a maximal chain of minimal $s-t$ cuts in $\bar{\mathcal{G}}$. Indeed, please note that if a strong component $S_i$ intersects a minimal cut $C$ non-trivially, then necessarily $S_i \subseteq C$ holds. We observe also, that the strong components and their topological ordering can be computed in $O(|V|+|E|)$ time [28].

Based on these observations, the construction works as follows. First, search for 3 disjoint $s-t$ paths in $\bar{\mathcal{G}}$. If such a solution exists, then DC is a feasible solution, as three disjoint paths exist in graph $\mathcal{G}$. As a consequence of Definition 1 a fault-tolerant coding graph has 2 disjoint $s-t$ paths in $\bar{\mathcal{G}}$. Thus, in the rest of this subsection we assume that the maximal number of disjoint $s-t$ paths in $\bar{\mathcal{G}}$ is 2. Note that, Menger's theorem [27, Thm. 6.7] states that the maximal number of disjoint $s-t$ paths in $\bar{\mathcal{G}}$ equals to the size of the minimum $s-t$ cut of $\bar{\mathcal{G}}$. Thus, $\bar{\mathcal{G}}$ has a cut with 2 edges, which in any fault tolerant coding graph must be a 2-biedge $s-t$ cut with capacity 3. In other words the maximal flow in $\bar{\mathcal{G}}$ is exactly 3.

Next we find a maximal chain $\{C_k\}$ of cuts in time $O(|V|+|E|)$ as indicated previously. We delete edges with flow 0, and scan the edges $e=(u,v)$ of $\bar{\mathcal{G}}$ which violate Property 3, and decrease the reduced capacity of $e$ to $\bar{c}'(e)=1$ in one round for all such edge $e$. Now, we have to show that the resulting graph $\bar{\mathcal{G}}' = (V, \bar{E}, \bar{c}')$ (with the same node and edge set as $\bar{\mathcal{G}}$ but with the modified reduced capacities $\bar{c}'$) still allows an $s-t$ flow of value 3 (i.e., satisfies Property 1). It does indeed, otherwise $\mathcal{G}$ had a cut consisting of two biedges, one of them (say $e$) with capacity lowered to $\bar{c}'(e)=1$. These two edges must form a cut of value 3 in $\bar{\mathcal{G}}$. This is impossible, because in the residual graph for the 3 value flow of $\bar{\mathcal{G}}$ there is a directed path from $u$ to $v$. Hence, $e$ cannot be the part of a minimum cut there.

Next, we have to compute a 3 value flow in $\bar{\mathcal{G}}'$ and possibly a new chain of minimal cuts $\{C'_j\}$. Note that minimal cuts $\{C_k\}$ of $\bar{\mathcal{G}}$ we obtained earlier will still remain minimal cuts, but there may be new ones. Essentially we may refine the original chain. We discard the edges with 0 flow. Note that, in $\bar{\mathcal{G}}'$ an arbitrary edge of capacity $\bar{c}'(e)=1.5$ is in a minimal cut, because all the edges with $\bar{c}'(e)=1.5$ capacity belong to a 2-biedge-cut[7].

Thus, the graph $\bar{\mathcal{G}}'$ and the new $\{C'_j\}$ chain of cuts satisfies Properties 1-3, i.e., using $\bar{\mathcal{G}}'$ one can construct edge sets $E_1, E_2, E_3$ by following the theoretical construction of Theorem 2 along the chain of cuts $\{C'_j\}$ in time $O(|V|+|E|)$.

## B. Routing DAG Decomposition of Minimal Graphs

In dynamic routing, where demands arrive one after the other, an efficient algorithm for intra-session code construction has utmost importance. Given a minimal coding graph $\mathcal{G}$, in this section we provide a linear time algorithm to perform robust network code construction, i.e., to obtain the three routing DAGs in $\mathcal{G}$.

We build the construction on the following observation. In the proof of Theorem 2, the graph $\mathcal{G}$ is divided into subgraphs based on a maximal chain of minimal $s-t$ cuts $\{C_k\}$, denoted by $\mathcal{G}_i, i=1,\ldots,k-1$ (shown in Fig. 3), and in each coding subgraph segments of $E_1^i$, $E_2^i$, and $E_3^i$ are defined. As $\mathcal{G}$ was minimal, the coding graph has no edges other than $E_1^i$, $E_2^i$, and $E_3^i$, thus, $\mathcal{G}_i$ satisfies the following properties:

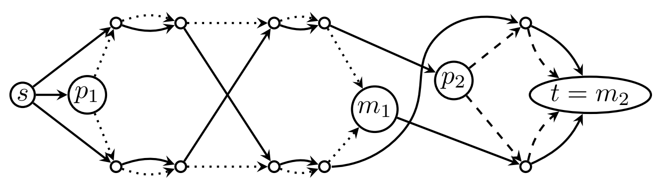
- In Type (i)-(ii) $\mathcal{G}_i$ there are no splitter or merger nodes.
- In Type (iii) $\mathcal{G}_i$ there is a single merger $m$ and no splitter.
- In Type (iv) $\mathcal{G}_i$ there is a single splitter $p$ and no merger.

Note that, a Type (iii) $\mathcal{G}_i$ can be followed by zero or a series of Type (i) subgraphs until a Type (iv) $\mathcal{G}_j$ subgraph, and similarly, a Type (iv) $\mathcal{G}_j$ subgraph can be followed by zero or a series of Type (ii) subgraphs until a Type (iii) $\mathcal{G}_i$ subgraph. Thus, $\{C_k\}$ *defines a unique sequence of splitter and merger nodes in* $\mathcal{G}$. We further define $s$ as a splitter if it has 2 outgoing biedges, and similarly, we define $t$ as merger if it has 2 incoming biedges. As a result, we have a well-defined sequence of $p_l - m_l$ splitter-merger pairs (Fig. 4a).

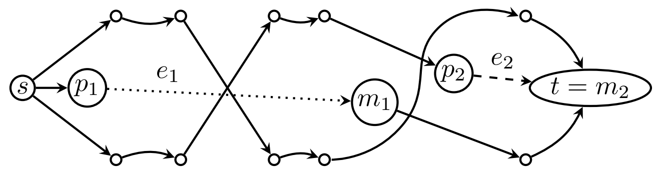
Based on the above argument, the method to find the disjoint edge-sets $E_1, E_2, E_3$ is provided in Algorithm 1. In Step (2) a maximal chain of minimal $s-t$ cuts $\{C_k\}$ in $\bar{\mathcal{G}}$ is calculated,

[6]BFS is a graph traversal algorithm, which starting at a node, first visits its neighbour nodes, then the unvisited neighbours of these neighbours, etc. Finally, it will find the minimum hop path to every node in linear time.

[7]It is easy to see that in $\bar{\mathcal{G}}$ there are no edges $e=(u,v)$ of capacity $\bar{c}(e)=1.5$ for which $u \in C_i, v \in C_j$, and $j < i$.



(a) The splitter-merger pairs and the disjoint paths in Iteration (4).



(b) The auxiliary graph in Step (8).

Fig. 4. An example of the network code construction in Algorithm 1.

**Algorithm 1:** Robust Code Construction

```
   Input: $\mathcal{G} = (V, E)$, where $\mathcal{G}$ is minimal, nodes $s, t$
   Result: Edge sets $E_1, E_2, E_3$
1  begin
2      With a max-flow algorithm in $\bar{\mathcal{G}}$ create maximal chain
       of minimal $s - t$ cuts $C_1 \subset C_2 \subset \ldots \subset C_k$ in $\mathcal{G}$.
3      Identify the $l$ splitter merger pairs $p_j - m_j$.
4      for $j = 1, 2, \ldots l$ do
5          Find a biedge-disjoint path-pair in $\mathcal{G}$ between $p_j - m_j$
           (edges are stored in set $S_j$).
6          Add edge $e_j = (p_j, m_j)$ to $\mathcal{G}$.
7          $\forall e \in S_j$ : remove $e$ from $\mathcal{G}$.
8      Find three disjoint paths containing edges $E_1, E_2, E_3$,
       respectively in $\mathcal{G}$ between $s - t$.
9      for $k = 1, 2, 3$ do
10         if $E_k$ contains $e_j, j = 1, 2, \ldots l$ then
11             Replace $e_j \in E_k$ with the edges in $S_j$.
```

which can be done in linear time as discussed in Section V-A. In Step (3) the splitter-merger pairs are identified in the subsequent subgraphs $\mathcal{G}_i$ by inspecting each node once. It is easy to see that in Step (5) there exists a biedge-disjoint pair of paths (i.e., the path-pair cannot traverse both edges of a biedge to maintain resilience) between a splitter and its corresponding merger in coding graph $\mathcal{G}$. Note that, Step (5) runs on disjoint parts (in different coding subgraphs) of coding graph $\mathcal{G}$, thus, the total complexity of finding disjoint paths could be upper bounded with $O(|V| + |E|)$.

Note that, between a splitter $p_i$ and its corresponding merger $m_i$ there are only Type (ii) minimal cuts (2-biedge cuts, e.g., $C_2$ and $C_3$ in Fig. 3). Thus, by replacing a disjoint path-pair with a single edge of capacity 1, flow value at least three is still maintained in the residual graph between these cuts, without affecting others. Thus, in Step (8) the residual graph contains three disjoint paths (Fig. 4b), which can be found in $O(|V| + |E|)$ time (as the augmenting paths in the residual graph can be found with BFS). Finally, in Iteration (9) the edges of the auxiliary graph are transformed back to the input topology in order to obtain the three disjoint edge-sets (routing DAGs). Thus, the network code construction in Algorithm 1 takes $O(|V| + |E|)$ time in total.

For the sake of explanation, we present three special minimal coding graphs $\mathcal{G}$ to demonstrate that the robust code construction of Algorithm 1 is able to provide routing DAG decomposition for the coding graph calculated with an arbitrary survivable routing algorithm which ensure instantaneous recovery from sinlge edge failures for two data parts [6]:

*1) Diversity coding:* we have 3 disjoint paths between $s$ and $t$, i.e., there are no splitter and merger nodes in the coding graph. Thus, Step (8) gives directly the edge sets $E_1, E_2, E_3$ of the routing DAGs, i.e., $A$, $B$ and $A \oplus B$ are sent along three disjoint paths.

*2) $1 + 1$ path protection:* we have 2 disjoint paths between $s$ and $t$ each with capacity 2, thus every edge in $\mathcal{G}$ is a biedge. It means, in $\mathcal{G}$ source $s$ has 2 outgoing biedges and $t$ has two incoming biedges, i.e., $p_1 = s$ and $m_1 = t$. In Step (4) we search for biedge-disjoint paths in $\mathcal{G}$, remove the traversed edges (stored in $S_1$) in Step (7), and add edge $e_1 = (s, t)$ to $\mathcal{G}$. Now in Step (8) we will have three disjoint paths, two actual paths in $\mathcal{G}$ which are using without loss of generality $E_2$ and $E_3$, while the third is the virtual edge $e_1$. In Step (11) the virtual edge in $E_1 = \{e_1\}$ is replaced with the edges in $S_1$, and the coding algorithm terminates. As a result, we have a resilient solution for $1 + 1$, as we send $A \oplus B$ on the two disjoint paths in $E_1$. Thus, any single edge failure will disrupt only one flow, and both streams can be recovered (from $A$ and $A \oplus B$, or from $B$ and $A \oplus B$) at the destination[8].

*3) General diversity coding solution:* In Figure 4 the steps of Algorithm 1 are demonstrated on a general minimal coding graph. First, a maximal chain of minimal $s - t$ cuts is found (shown in Fig. 3), and the splitter and merger nodes are identified in Step (3). Next, we search for disjoint paths in $\mathcal{G}$ between each $p_j - m_j$ pairs (denoted by dotted and dashed lines in Figure 4a). Second, the edges of the disjoint paths are replaced with a single edge $e_j$, shown in Figure 4b. As a result, this graph contains three disjoint paths (found in Step (8)), and can easily return the corresponding edge sets of the routing DAGs in Step (11).

As a summary, we claim that if an arbitrary single edge failure resilient minimal coding graph is given for two data parts, robust XOR codes can be constructed in linear time (in fact, in $O(|V| + |E|)$ time). Furthermore, core network switches do not need to be upgraded to perform any algebraic operations. Unfortunately, as discussed in Section VI, this simplicity generally does not hold.

## VI. Scope of Routing DAG Decomposition

We have seen that robust code construction is effective with routing DAG decomposition for GDC, i.e., for the case of two data parts and single edge failure resilience. Two straightforward generalisations may be to increase the number of data parts or the number of possible edge failures. In this

[8]We note here that for $1 + 1$ a solution exists without coding. However, our goal was to demonstrate the generality of our approach, i.e., it contains both diversity coding and $1 + 1$ as special cases.

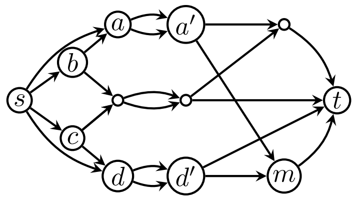


Fig. 5. A dual failure resilient connection in $\mathcal{G}$ with two data parts where 4 end-to-end routing DAGs do not exist, i.e., GDC cannot be generalised to multiple failures.

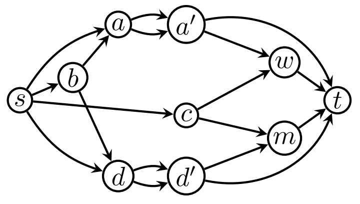


Fig. 6. A single failure resilient connection in $\mathcal{G}$ with three data parts where 4 end-to-end routing DAGs do not exist, i.e., GDC cannot be generalised to more than two source data parts.

section we present minimal coding graphs to show that none of these generalisations are directly possible, i.e., higher field for coding, signaling in the recovery process or core network modifications might be necessary.

### A. Dual Edge Failure Resilience with Two Data Parts

Figure 5 presents a coding graph, where two data parts have to be sent from $s$ to $t$ while dual edge failure resilience is maintained. One can easily check that the above connection can survive two failures, because removing any two edges (or biedges) of $\mathcal{G}$, there are at least two remaining paths in $\mathcal{G}$ from $s$ to $t$, i.e., the minimal capacity of two units is maintained. Note that the graph is minimal with respect to this property.

Edges $(s, a), (s, b), (s, c), (s, d)$ form a 4-edge-cut, so any two of them can remain after the failure of the other two. This shows that 3 end-to-end subgraphs do not guarantee protection against two failures, so we need 4 routing DAGs, and we need a network code at the source such that the original data is decodable when receiving at least two of them at the destination (clearly in a field larger than $GF(2)$). However, we show that even in this case edges of the coding graph in Figure 5 cannot be split into 4 end-to-end routing DAGs.

Assume indirectly that it can be, and let $E_1, E_2, E_3, E_4$ denote the edge-sets. Edges in a 4-edge-cut have to belong to separate routing DAGs, hence it is easy to see that edges entering nodes $a$ and $d$ belong to two completely different edge-sets. Without loss of generality we may assume that edge $(a', m), (m, t) \in E_1$ and $(d', m) \in E_2$. Then edge $(d', m)$ does not have an end-to-end connection in $E_2$, contradicting the assumption.

Thus, robust network code in the form of routing DAGs does not exist in general for this special case.

### B. Single Edge Failure Resilience with Three Data Parts

Figure 6 presents a coding graph, where the source data is divided into three data parts ($A$, $B$ and $C$) and these data parts have to be sent from $s$ to $t$ while single edge (or biedge) failure resilience is maintained. In Figure 6, the failure of any edge preserves a flow of value 3 from $s$ to $t$, i.e., the necessary requirement for instantaneous recovery is maintained.

However, edges $(s, a), (s, b), (s, c), (s, d)$ form a 4-edge-cut, so with the GDC approach again 4 end-to-end edge-sets are needed to resist at most the failure of a biedge in $\mathcal{G}$. We show that such partition of the edges does not exist. Suppose that indirectly, there are such sets $(E_1, E_2, E_3, E_4)$ in the graph and assume that $(s, c) \in E_1$. Then so is $(c, w)$ and $(c, m)$. Since $(w, t), (m, t), (a', t), (d', t)$ also form a 4-edge-cut, exactly one of them belongs to $E_1$, and it is either $(m, t)$ or $(w, t)$. Because of symmetry we can assume it is $(m, t)$ while $(w, t) \in E_2$. Then after the failure of biedge $(a, a')$, only two data parts can be transmitted to $t$, even if node $w$ switches to $(c, w)$, which clearly requires control plane signaling, which results increased restoration time, i.e., instantaneous recovery is not ensured.

Note that, by adding $k$ further edges from $s$ to $t$, the same argument holds for $k + 3$ data parts.

## VII. Experimental Results

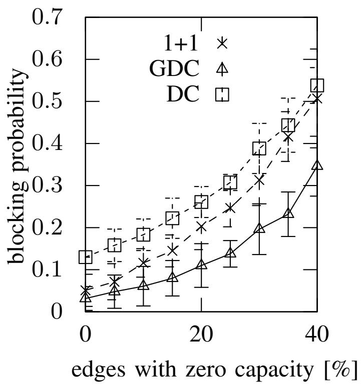
The main contributions of the paper – no in-network coding is required and the robust network codes, i.e., routing DAGs can be constructed in linear time – made GDC as a competitor of DC and $1 + 1$ to provide instantaneous failure recovery. The simulation section is devoted to provide help in the decision between the three approaches for network operators by investigating the resource requirements of minimal coding graphs $\mathcal{G}$ of GDC compared to its counterparts in at least 2-connected network topologies.

We considered two performance metrics in this study: blocking probability and resource saving. A connection request with bandwidth of 2 units ($A$ and $B$) between nodes $s - t$ is *blocked* if no fault-tolerant solution is found with the given protection method (see Fig. 1), i.e.,

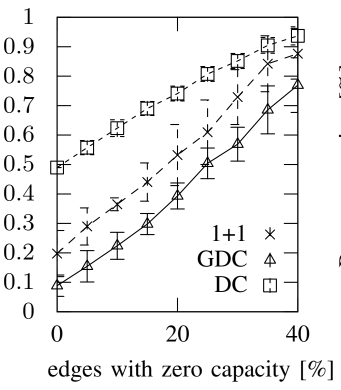
| | |
|---|---|
| $1 + 1$ | an edge-disjoint path-pair from $s$ to $t$ with 2 units of bandwidth; |
| DC | three edge-disjoint paths from $s$ to $t$ with 1 unit of bandwidth; and |
| GDC | a fault-tolerant coding subgraph according to Definition 1. |

The *blocking probability* is the ratio of the blocked connection requests to all generated connection requests. The *resource saving* is the amount of bandwidth save of DC and GDC compared to $1+1$ protection with respect to the total capacity needed for routing all generated connection requests.

In order to investigate the performance of the methods independently from the traffic pattern, we generated connection requests between all $s - t$ pairs in a given network, and calculated the resource saving and blocking probability for a fixed capacity setting on the edges. Hence, in a given setting edges with 2 units of capacity are usable by all methods; there might be bottleneck edges (have only 1 unit of capacity,

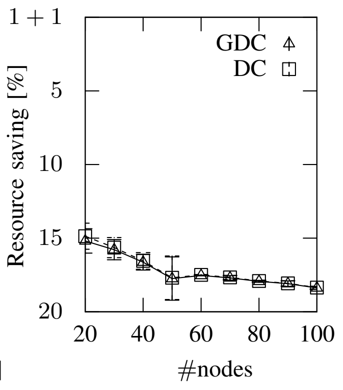


(a) Random dense networks

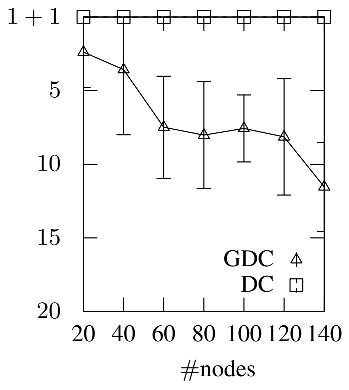


(b) Random sparse networks

Fig. 7. Blocking probability in randomly generated 60 node 2-connected planar networks. Average of random problem instances of networks with high traffic load, where $x\%$ of the edges have zero capacity, 20% have capacity of 1 unit, and the rest $(80 - x)\%$ have capacity of 2 units.



(a) 3-connected



(b) 2-connected (single 2-biedge cut)

Fig. 8. Resource saving with the GDC and DC approaches on planar networks compared to $1 + 1$. For topologies with a single 2-biedge cut, the source and the target nodes of the connection requests are selected from different 3-connected components.

thus, not usable for $1 + 1$); and some edges might be fully saturated (zero capacity). To measure the blocking probability we generated random topologies, where $x\%$ of randomly selected edges have zero capacity, 20% are bottlenecks with capacity of 1 unit, and the rest $(80 - x)\%$ have capacity of 2 units, i.e., available for all methods. In order to make a fair comparison, for the resource saving simulations we assume that each edge has 2 units of capacity, thus, none of the methods will be limited by the capacity constraint. However, a connection request may still be blocked by DC owing to the network topology, i.e., if there are no 3 disjoint paths between $s$ and $t$. In this case we count zero resource saving for DC compared to $1 + 1$.

Suurballe's algorithm [21] was used to calculate a fault-tolerant solution with minimum capacity requirement for $1+1$ and DC, on the edges with 2 units of capacity and with at least 1 unit of capacity, respectively. To obtain the minimal capacity coding graph $\mathcal{G}$ for GDC with two data parts, we used the Integer Linear Program (ILP) presented for GDP-NC [25] with single failures. On this minimal coding graph $\mathcal{G}$ we can run Algorithm 1 to obtain the routing DAG decomposition for GDC. Finally, we also compare our method to the theoretical upper bound in resource savings [24] for any proactive approach providing instantaneous recovery, which assumes single edge failure resilience, but do not limit the number of data parts to two [16]. Although the practical limitations of two data parts in GDC seems to be restrictive, we show that performance-wise we have to pay limited price for it.

## A. *Performance Study on Random Network Topologies*

First we used random 2-connected planar networks with different densities. We group the networks into two categories according to the average number of edges bounding the regions, called faces. The first category is called *dense networks* where most of the faces have size 4, while the second category is called *sparse networks* with typical faces of size at least 6.

In Fig. 7 we present the blocking probabilities in randomly generated 60 node dense (Fig. 7a) and sparse (Fig. 7b) networks. Generation of 20% bottleneck edges with 1 unit of capacity and $x\%$ of zero capacity edges was done for five different random seeds. One can observe that the GDC significantly outperforms the DC and $1 + 1$, as it contains both methods as special cases. To be specific, the blocking probability of GDC is on average $5 - 15\%$ less than by $1 + 1$, and shows even higher flexibility compared to DC, where besides the capacity constraints blocking occurs owing to the lack of 3 disjoint paths in the 2-connected topology.

In Fig. 8 we investigated the *resource saving* which can be reached with end-to-end coding approaches DC and GDC in 3-connected planar (Delaunay triangulated) topologies between all $s - t$ pairs with 2 units of capacity on each edge. The resource saving increases as network size increases (Fig. 8a), which meets our expectation as the opportunities to decrease resource consumption (i.e., split or merge the data flow) grow. As the Delaunay triangulated graphs are dense networks (average nodal degree is around 5.3), the DC method performs similarly to our GDC approach as a third edge-disjoint path with a moderate length exists in the topology. Thus, the average resource saving which can be reached through end-to-end coding (either with DC or GDC) is around $15 - 20\%$.

In Fig. 8b we considered two identical 3-connected topologies interconnected with two edges between distinct node-pairs, and varied the inter-connection points between the two components. For example, the 20 node topology in Fig. 8b means that two 10-node Delaunay triangulated graphs are interconnected. Clearly, the two edges between the two 3-connected components is a cut with 2 edges, making the topology 2-connected. Connection requests with the source node in the one and the target node in the other 3-connected component were generated. As such, DC cannot bring any benefit to these connection requests, while the GDC framework still approaching $5 - 10\%$ resource saving, and clearly

TABLE I

SIMULATION RESULTS ON SOME WELL-KNOWN NETWORKS (UPPER PART: SNDLIB [12], LOWER PART: ROCKETFUEL ASS [13]). BLOCKING PROBABILITY RESULTS ARE SHOWN WITH RANDOM 20% UNIT CAPACITY EDGES AND $80 - x\%$ 2 CAPACITY EDGES, WHILE $x$ (THE NUMBER OF UNUSABLE ZERO CAPACITY EDGES) INCREASES. RESOURCE SAVING RESULTS CORRESPOND TO A NON-BLOCKING SCENARIO, WHERE ONLY A LIMITED NUMBER (THREE) OF BOTTLENECK (UNIT CAPACITY) EDGES ARE PRESENT, WHILE OTHER EDGES HAVE CAPACITY OF 2 UNITS.

| | Graph | | | Blocking probability [%] | | | | | | | | | $1+1$ | Resource saving [%] | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | 0% zero capacity edges | | | 10% zero capacity edges | | | 20% zero capacity edges | | | avg. | theor. | | |
| | $\|V\|$ | $\|E\|$ | diam. | GDC | $1+1$ | DC | GDC | $1+1$ | DC | GDC | $1+1$ | DC | edges | bound | GDC | DC |
| Pan-European | 16 | 22 | 6 | 21.6 | 25.8 | 62.5 | 35 | 56.6 | 83.3 | 48.3 | 70 | 85 | 8.87 | 23.52 | 23.52 | 22.81 |
| European | 22 | 45 | 5 | 12.1 | 12.1 | 41.1 | 17.3 | 17.3 | 45 | 21.1 | 25.5 | 48 | 6.18 | 13.93 | 13.47 | 10.95 |
| Italian | 33 | 56 | 9 | 6.1 | 11.7 | 67.4 | 23.8 | 28.6 | 72.3 | 33.9 | 56.8 | 76.7 | 10.29 | 14.14 | 14.09 | 11.55 |
| Cost 266 | 37 | 57 | 8 | 8.4 | 20.5 | 51 | 35.4 | 41.9 | 65.3 | 48.6 | 50.6 | 73.7 | 10.83 | 19.29 | 19.25 | 17.08 |
| North American | 39 | 61 | 10 | 0 | 7.3 | 48.9 | 7.4 | 55.3 | 65.7 | 54.9 | 76.5 | 87.4 | 11.76 | 19.47 | 19.42 | 17.34 |
| NSFNET | 79 | 108 | 16 | 34.2 | 34.2 | 72 | 42.2 | 42.2 | 79.2 | 42.2 | 59.3 | 85.8 | 18.10 | 15.06 | 14.99 | 11.91 |
| AS6461 (Abovenet) | 17 | 37 | 4 | 5.9 | 5.9 | 51.4 | 5.9 | 5.9 | 51.5 | 11 | 11 | 59.5 | 5.70 | 18.45 | 17.39 | 15.46 |
| AS1755 (Ebone) | 18 | 33 | 5 | 9.1 | 9.1 | 49 | 18.9 | 18.9 | 55.5 | 18.9 | 24.8 | 67.3 | 6.30 | 12.86 | 12.79 | 11.86 |
| AS3967 (Exodus) | 21 | 36 | 5 | 17.1 | 17.1 | 56.6 | 17.1 | 17.1 | 63.8 | 28.6 | 28.6 | 75.7 | 7.84 | 23.11 | 23.10 | 21.58 |
| AS7018 (AT&T) | 22 | 38 | 5 | 24.6 | 29.4 | 76.2 | 29.4 | 37.7 | 76.2 | 44.1 | 48 | 79.2 | 6.86 | 11.40 | 11.23 | 9.24 |
| AS3257 (Tiscali) | 27 | 64 | 4 | 7.4 | 7.4 | 51.2 | 15.1 | 15.1 | 55.3 | 21.4 | 21.4 | 63 | 5.34 | 14.11 | 13.45 | 11.56 |
| AS1239 (Sprintlink) | 30 | 69 | 6 | 9.2 | 9.2 | 51.5 | 15.4 | 15.4 | 58.6 | 25.9 | 41.1 | 61.8 | 7.25 | 14.95 | 14.04 | 11.53 |

demonstrates the advantage of GDC over DC in 2-connected topologies.

### *B. Performance Study on Real-World Topologies*

In order to investigate the performance of the proposed approaches in a realistic scenario, we selected some SNDLib [12] and Rocketfuel [13] transport network topologies. We examine the blocking probabilities in the same way as in Fig. 7, i.e., the capacity of randomly selected 20% of the edges is set to 1 unit, and 0, 10% and 20% of the edges have zero capacity. The results are presented in Table I. As GDC contains both DC and $1 + 1$ as special cases, obviously it improves the blocking probability of both approaches, as it provides a fault-tolerant solution in all networks where any of these approaches provide. Furthermore, owing to its superb flexibility, GDC provides solution for networks where the previous methods may fail.

For the resource saving simulations in each real network topology we identified three edges most prone to congestion based on their betweenness centrality value (assuming shortest path routing), and considered them as bottlenecks (with edge capacity of 1 unit), and the rest of the edges have 2 units of capacity to ensure that $1+1$ has a fault-tolerant solution for all connection requests. One can observe that the resource saving of our GDC approach is about $10-20\%$ compared to the $1+1$ approach, while the optimality gap of GDC is less than 1% in comparison to the theoretical upper bound in the investigated transport topologies. Although DC performs close the GDC in these networks, it blocks several connection requests owing to these real-world networks are not 3-connected topologies, while GDC provides a fault-tolerant solution for all $s-t$ pairs.

## VIII. CONCLUSIONS

In this paper we showed that robust network codes for single edge failure resilient unicast connections where user data can be split into two parts are equivalent with finding three end-to-end routing DAGs. Built on this observation, we gave linear time algorithms to obtain the routing DAGs in fault-tolerant and minimal coding graphs, instead of involved gadget transformations presented in [5]. We also showed that such an equivalence between robust network codes and network flow problems may not exist for other transport network scenarios. Through simulations we demonstrated that the minimal capacity coding graphs for GDC reach about $15-20\%$ resource saving compared to $1 + 1$ protection, while outperform DC with $5 - 10\%$ even in topologies with a single 2 edge cut. Furthermore, as GDC contains both $1 + 1$ and DC as special cases, it is solvable in all network topologies (and even more) where any of these methods is solvable. Thus, we conclude that GDC might be an alternative of DC in 2-connected topologies and its improvement in denser networks to reduce the resource consumption of $1 + 1$ without in-network coding.

## REFERENCES

[1] Y. Liu, D. Tipper, and P. Siripongwutikorn, "Approximating optimal spare capacity allocation by successive survivable routing," *IEEE/ACM Transactions on Networking*, vol. 13, no. 1, pp. 198–211, Feb. 2005.

[2] D. Wang and G. Li, "Efficient distributed bandwidth management for mpls fast reroute," *IEEE/ACM Transactions on Networking*, vol. 16, no. 2, pp. 486–495, 2008.

[3] M. Médard, R. Barry, S. Finn, W. He, and S. Lumetta, "Generalized loop-back recovery in optical mesh networks," *IEEE/ACM Transactions on Networking*, vol. 10, no. 1, p. 164, 2002.

[4] B. Wu, K. Yeung, and P.-H. Ho, "Ilp formulations for p-cycle design without candidate cycle enumeration," *IEEE/ACM Transactions on Networking*, vol. 18, no. 1, pp. 284–295, 2010.

[5] S. Rouayheb, A. Sprintson, and C. Georghiades, "Robust network codes for unicast connections: A case study," *IEEE/ACM Transactions on Networking*, vol. 19, no. 3, pp. 644–656, 2011.

[6] P. Babarczi, G. Biczók, H. Overby, J. Tapolcai, and P. Soproni, "Realization strategies of dedicated path protection: A bandwidth cost perspective," *Elsevier Computer Networks*, vol. 57, no. 9, pp. 1974 – 1990, 2013.

[7] A. Markopoulou, G. Iannaccone, S. Bhattacharyya, C. Chuah, and C. Diot, "Characterization of failures in an IP backbone," in *Proc. IEEE Infocom*, vol. 4, 2004, pp. 2307–2317.

[8] A. Fumagalli and L. Valcarenghi, "IP restoration vs. WDM protection: is there an optimal choice?" *IEEE Network*, vol. 14, no. 6, pp. 34–41, 2000.

[9] E. Ayanoglu, I. Chih-Lin, R. Gitlin, and J. Mazo, "Diversity coding for transparent self-healing and fault-tolerant communication networks," *IEEE Transactions on Communications*, vol. 41, no. 11, pp. 1677–1686, 1993.

[10] S. Avci and E. Ayanoglu, "Link failure recovery over large arbitrary networks: The case of coding," *IEEE Transactions on Communications*, vol. 63, no. 5, pp. 1726–1740, May 2015.

[11] A. Pasic, J. Tapolcai, P. Babarczi, E. Bérczi-Kovács, Z. Király, and L. Rónyai, "Survivable routing meets diversity coding," in *Proc. IFIP Networking*, 2015, pp. 1–9.

[12] S. Orlowski, R. Wessäly, M. Pióro, and A. Tomaszewski, "Sndlib 1.0—survivable network design library," *Networks*, vol. 55, no. 3, pp. 276–286, 2010.

[13] N. Spring, R. Mahajan, and D. Wetherall, "Measuring isp topologies with rocketfuel," *ACM SIGCOMM Computer Communication Review*, vol. 32, no. 4, pp. 133–145, 2002.

[14] I. Barla, F. Rambach, D. Schupke, and G. Carle, "Efficient Protection in Single-Domain Networks using Network Coding," in *IEEE GLOBECOM 2010*, 2010, pp. 1–6.

[15] S. A. Aly, A. E. Kamal, and O. M. Al-Kofahi, "Network protection codes: Providing self-healing in autonomic networks using network coding," *Elsevier Computer Networks*, vol. 56, no. 1, pp. 99 – 111, 2012.

[16] R. Koetter and M. Médard, "An algebraic approach to network coding," *IEEE/ACM Trans. on Networking*, vol. 11, no. 5, pp. 782–795, 2003.

[17] S. Jaggi, P. Sanders, P. Chou, M. Effros, S. Egner, K. Jain, and L. Tolhuizen, "Polynomial time algorithms for multicast network code construction," *IEEE Transactions on Information Theory*, vol. 51, no. 6, pp. 1973–1982, 2005.

[18] A. E. Kamal, A. Ramamoorthy, L. Long, and S. Li, "Overlay protection against link failures using network coding," *IEEE/ACM Transactions on Networking (TON)*, vol. 19, no. 4, pp. 1071–1084, 2011.

[19] E. D. Manley, J. S. Deogun, L. Xu, and D. R. Alexander, "All-optical network coding," *Journal of Optical Communications and Networking*, vol. 2, no. 4, pp. 175–191, 2010.

[20] P. Babarczi, J. Tapolcai, L. Rónyai, and M. Médard, "Resilient flow decomposition of unicast connections with network coding," in *Proc. IEEE Intl. Symp. on Information Theory (ISIT)*, 2014, pp. 116–120.

[21] J. W. Suurballe, "Disjoint paths in a network," *Networks*, vol. 4, pp. 125–145, 1974.

[22] M. Mohandespour and A. Kamal, "1+N protection in polynomial time: a heuristic approach," in *Proc. IEEE GLOBECOM*, 2010, pp. 1–6.

[23] A. Kamal, "1+ N network protection for mesh networks: network coding-based protection using p-cycles," *IEEE/ACM Transactions on Networking*, vol. 18, no. 1, pp. 67–80, 2010.

[24] P. Babarczi, J. Tapolcai, P.-H. Ho, and M. Médard, "Optimal Dedicated Protection Approach to Shared Risk Link Group Failures using Network Coding," in *Proc. IEEE International Conference on Communications (ICC)*, 2012, pp. 3051–3055.

[25] P. Babarczi, A. Pasic, J. Tapolcai, F. Németh, and B. Ladóczki, "Instantaneous recovery of unicast connections in transport networks: Routing versus coding," *Elsevier Computer Networks*, vol. 82, pp. 68–80, 2015.

[26] B. Ladóczki, C. Fernandez, O. Moya, P. Babarczi, J. Tapolcai, and D. Guija, "Robust network coding in transport networks," in *Proc. 34th IEEE INFOCOM Demo session*, 2015, pp. 1–2.

[27] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network flows: theory, algorithms, and applications*. Prentice hall, 1993.

[28] B. Aspvall, M. F. Plass, and R. E. Tarjan, "A linear-time algorithm for testing the truth of certain quantified boolean formulas," *Information Processing Letters*, vol. 8, no. 3, pp. 121–123, 1979.

[29] B. Bollobás, *Modern Graph Theory*, ser. Graduate Texts in Mathematics. Springer-Verlag GmbH, 1998.

## Appendix

**Lemma 5.** *There are three edge-disjoint routing DAGs in a Type (iii) coding subgraph, such that two of them are simple paths (one of them traversing $f$ and $e_x$, and the other passing through $g$ and $e_y$, where $x \neq y$), while the third routing DAG is composed of three path segments ($s_i \to m$ traversing edge $f$, $s_i \to m$ containing edge $g$, and $m \to t_i$ passing through $e_z$, where $m$ is a merger node, $z \neq y$ and $z \neq x$).*

*Proof:* To show this, we need to dig into the Ford-Fulkerson theorem.

Let $F$ be an integer flow of value 3 in $\mathcal{G}_i$. The existence of this flow implies that there are three edge-disjoint paths $P_1, P_2, P_3$ from $s_i$ to $t_i$ in $\mathcal{G}_i$. The edges $e_1, e_2, e_3$, and 3 of the 4 edges corresponding to $f, g$ are traversed by $F$ because they originate from minimum cuts in $\bar{\mathcal{G}}$. Without loss of generality we may assume that $P_1$ and $P_2$ traverses the edges in $\mathcal{G}_i$ corresponding to biedge $f$, and $P_3$ traverses one of the edges of biedge $g$ and hence passes through node $v$, where $g = (s_i, v)$. There must exist a $v - t_i$ flow in $\mathcal{G}_i$ of value at least 2, otherwise $\mathcal{G}_i$ would not be fault-tolerant (removing $f$ disrupts two routing DAGs). For example, $P_1 = \{f, e_1\}, P_2 = \{f, y, e_2\}, P_3 = \{g, e_3\}$ in $\mathcal{G}_3$.

Let $P = \{P_3 \setminus g\}$, i.e., the $v \to t_i$ path segment of $P_3$ (e.g., $P = \{e_3\}$ in $\mathcal{G}_3$). We can send a flow of value 1 through $P$ from $v$ to $t_i$. This is not a maximal $v - t_i$ flow, hence there exists an augmenting path for it (see Bollobás [29, Chapter III] or [27, Chap. 7] for basic facts related to the Ford-Fulkerson theorem). To form such a path we can use the edges of $P$ in the reverse direction and all other edges of $\mathcal{G}_i$ in their original direction. In fact, it suffices to search for an augmenting path until the first node ($m$ in $\mathcal{G}_3$ on Fig. 3b) traversed by $P_1$ or $P_2$ (say, it is $P_2$). From $m$ one can walk along the edges of $P_2$ to reach $t_i$. Let $P^*$ be the $v \to m$ portion of the augmenting path ($h_1$ in $\mathcal{G}_3$). Let $Q$ be the union of the edges of $P$ and $P^*$ (from which we delete every edge of $P$ which occurs reversed in $P^*$, together with this reverse edge). In $Q$ the outdegree of $v$ is 2, the indegree of $m$ and $t_i$ is 1, and for all other nodes the indegree is the same as the outdegree. The indegree of $v$ and the outdegree of $m$ and $t_i$ is 0. These imply that $Q$ is the disjoint union of a path from $v$ to $t_i$ (dented as $Q_1$) and a path from $v$ to $m$ (denoted as $Q_2$). We have also $Q \subseteq E^i$, and the two paths are edge disjoint from from $P_1$ and $P_2$ as well. These facts together imply the lemma.

Finally, the path traversing $f$ and $e_x$ is $P_1$, the other path is $\{g, Q_1\}$, and the third routing DAG is $P_2 \cup \{g, Q_2\}$. ∎