SYPER: Synthesis of Perfectly Resilient Local Fast Re-Routing Rules for Highly Dependable Networks

Csaba Györgyi[1] Kim G. Larsen[2] Stefan Schmid[1,3] Jiří Srba[2]
[1]University of Vienna [2]Aalborg University [3]TU Berlin

***Abstract*—Modern communication networks support local fast re-routing (FRR) to quickly react to link failures. However, configuring such FRR mechanisms is challenging as the rules have to be defined ahead of time, without knowledge of the failures, and can depend only on local decisions made by the nodes incident to a failed link. Designing failover protection against multiple link failures is particularly difficult. We present a novel synthesis approach which addresses this challenge by generating FRR rules in an automated and provably correct manner. Our network model assumes that each node maintains a prioritised list of backup links (a.k.a. skipping forwarding)—an FRR method that allows for a memory-efficient deployment. We study the theoretical properties of the model and implement a synthesis method in our tool SYPER that aims to provide perfect resilience: if there are up to k link failures, we can always route traffic between any two nodes as long as they are still connected in the underlying physical network. To this end, SYPER focuses on the synthesis of efficient forwarding rules using the BDD (binary decision diagram) methodology and our empirical evaluation shows that SYPER is feasible, and can synthesize robust network configuration in realistic settings.**

***Index Terms*—Fast Re-Routing, Data Plane, Resilience, Formal Methods, Binary Decision Diagrams**

I. INTRODUCTION

Link failures are common in ISP networks [1], cloud provider WANs [2], and datacenters [3]. Accordingly many modern communication networks feature local fast re-routing (FRR) mechanisms to quickly react to link failures and reestablish connectivity. In a nutshell, FRR mechanisms typically rely on conditional forwarding rules on the routers which only apply if one or multiple incident links fails.

However, configuring such FRR mechanisms is challenging as the conditional rules have to be defined ahead of time, without the knowledge of the failures, and can only depend on the status of local link failures. Protecting networks against multiple failures is hence particularly difficult: an additional link failure along a backup path may lead to a forwarding loop if each node in the network is only incident to at most one of the failed links, and unaware of the other. Over the last decade, significant research efforts went into the design of FRR algorithms for multiple link failures [4]–[17]. While many of the solutions provide strong resilience guarantees, they were designed manually for specific network models.

We initiate the study of a general and automated approach to verify and synthesize fast rerouting mechanisms. In particular, we explore a formal approach and present a model of the FRR problem as well as an efficient synthesis algorithm based on binary decision diagrams (BDDs) [18], [19].

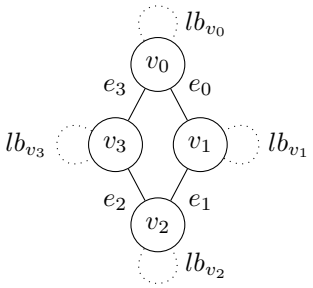
Our focus is on an efficient "skipping" routing which enables small forwarding tables [14] and can be implemented e.g. in the OpenFlow [20] framework. Here the conditional failover rules at each node are given as a prioritized list of out-edges for each in-edge. A packet is forwarded on the first non-failed edge. This representation is particularly efficient since it requires only a linear amount of space. Our goal is to provide rules achieving *perfect resilience* for k failures, meaning that packet delivery is guaranteed if the source and destination nodes are connected in the underlying network topology for up to k link failures.

a) Main contribution: Our main contribution is SYPER (standing for SYnthesis of PERfect Resiliency), a general framework and a software tool which synthesizes FRR schemes providing perfect resiliency under multiple failures whenever this is possible. To this end, we present a formal model and an efficient encoding of the FRR problem. Furthermore, we prove several theoretical results of the model that are later used to optimize performance of our BDD reformulation of the problem. BDD is a data structure introduced by Lee [18] for efficient storage and manipulation with Boolean functions. In our work, we exploit BDDs to synthesize and verify *all* possible routings for a given network topology resilient to multiple failures. Synthesizing *all* possible routings supports operator preferences in a proven and efficient way. We also show, how our tool can be used to disprove several conjectures e.g., claiming that the primary path (without failures) must follow some of the shortest paths to the destination. This may be of independent interest and also shows the usefulness of our tool. Our experiments demonstrate that the proposed method can be applied on realistic network topologies from the Topology Zoo dataset [21].

b) Organization of the paper: We provide a formal definition of our routing synthesis problem in Section II. In Section III we prove theorems that can be later used to optimize our solution while Section IV provides further insights on the problem by investigating different conjectures. Section V details our BDD formulation and describes the application of our optimizations. The tool SYPER is evaluated in Section VI. Finally, we review related work in Section VII and conclude in Section VIII.

II. PROBLEM FORMULATION

We shall first formally define the problem of synthesizing FFR routings and introduce the necessary notation. We represent a network as an undirected multigraph.



(a) Network topology.

in-edge	node	out-edges
lb_{v_1}	v_1	e_0, e_1
e_1	v_1	e_0, e_1
lb_{v_2}	v_2	e_2, e_1
e_1	v_2	e_2, e_1
e_2	v_2	e_1, e_2
lb_{v_3}	v_3	e_3, e_2
e_2	v_3	e_3, e_2

(b) Routing table with a prioritized list of out-edges.

Fig. 1: A simple network example.

Definition 1. *A* network *is an undirected multigraph* $G = (V, E, r)$ *where* V *a finite set of* nodes*,* E *a finite set of undirected* edges*, and* $r : E \to \{\{x, y\} \mid x, y \in V\}$ *a function assigning to each edge a set of endpoint nodes.*

Let $G = (V, E, r)$ be a fixed undirected multigraph representing a communication network interconnecting a set of nodes (routers) via edges (links). For technical reasons, we assume in the rest of the paper that for every node $v \in V$ there is always a *loop-back edge* called lb_v such that $r(lb_v) = \{v\}$. Figure 1a shows a simple example. Unless needed, we shall not draw the loop-back edges in our examples and we use the terms *link* and *edge* interchangeably.

A finite *path* in a network is a sequence of edges $(e_0, e_1, ..., e_n)$ such that there are nodes $v_0, v_1, ..., v_{n+1}$ where $r(e_i) = \{v_i, v_{i+1}\}$ for all $0 \leq i \leq n$. The path then *connects* the node v_0 to v_{n+1}.

A node $u \in V$ is *reachable* from $v \in V$ if there is a path connecting v to u. In the rest of this paper, we only consider connected networks, i.e. networks where every node is reachable (by at least one path) from every other node.

In our model we assume that edges in the network can fail. A *failure scenario* is represented as a set of failed edges.

Definition 2 (Failure Scenario)**.** *A* failure scenario F *is any subset of* E*, i.e.,* $F \subseteq E$*.*

To handle different failure scenarios, we provide a prioritized list of out-edges for a given in-edge and a node. This method is efficient and already introduced in literature [14]. The skipping routing defines a prioritized list in terms of an order in which edges should be tried in case of failures.

Definition 3 (Skipping Routing)**.** *A skipping* routing *in* G *is a partial function* $R : E \times V \rightharpoonup E^*$ *such that if* $R(e, v) = (e_1, e_2, \ldots, e_\ell)$ *then* $v \in r(e) \cap r(e_1) \cap \ldots \cap r(e_\ell)$*, i.e. all edges in the prioritized list as well as the incoming edge are connected to the node* v*.*

An example of a routing is given in Table 1b. The intuition is that, for a given a failure scenario F, if $R(e, v) = (e_1, e_2, \ldots, e_\ell)$ then any packet arriving on the edge e to the node v will be routed via the first available edge $e_i \notin F$ where all the edges with higher priority are failed, meaning that $e_1, \ldots, e_{i-1} \in F$. If $e_1, \ldots, e_\ell \in F$ then the packet is dropped. This is formalized by the notion of a next hop.

Definition 4 (Next Hop)**.** *Let* F *be a failure scenario and* R *a routing. We define the next-hop via the routing* R *under the failure scenario* F *as a partial function* $hop_R^F : E \times V \rightharpoonup E \times V$ *such that*

$$hop_R^F(e, v) = \begin{cases} (e_i, v_i) & \text{if } R(e, v) = (e_1, \ldots, e_k) \text{ and} \\ & e_1, \ldots, e_{i-1} \in F \text{ and} \\ & e_i \notin F \text{ where } r(e_i) = \{v, v_i\} \\ \textit{undefined} & \text{otherwise .} \end{cases}$$

A packet is delivered to a given destination node via a sequence of hops (that can be recursively applied).

Definition 5 (Packet Delivery)**.** *A node* s delivers *to a node* d *via the routing* R *under the failure scenario* F *if there is an* $n \in \mathbb{N}$ *such that* $d \in r((hop_R^F)^n(lb_s, s))$ *where* lb_s *is the loop-back edge such that* $r(lb_s) = \{s\}$*.*

Consider the routing R described by Table 1b for the network depicted in Figure 1a. The goal is to reach the destination v_0 from any other node in the network. Let e_3 be the only failed edge ($F = \{e_3\}$). If the packet starts at v_2, it is forwarded to v_3 using e_2 because $hop_R^F(lb_{v_2}, v_2) = (e_2, v_3)$. Next, the packet comes back to v_2 because the default (first priority) edge e_3 is down and thus $hop_R^F(e_2, v_3) = (e_2, v_2)$. After two more hops where $hop_R^F(e_2, v_2) = (e_1, v_1)$ and $hop_R^F(e_1, v_1) = (e_0, v_0)$), the packet is delivered to v_0.

Our goal is now to synthesise (all) skipping routings that deliver packets to a given destination node under failure scenarios with up to k failed links. This is formalized as follows where for a network $G = (V, E, r)$ we let $G^F = (V, E \setminus F, r|_{E \setminus F})$ be a network with all failed edges in F removed.

Definition 6 (Perfect Resilience [14], [22])**.** *Let* $d \in V$ *be a destination node. A routing* R *is* perfectly k-resilient*, if* s *delivers to* d *for all source nodes* s *and all failure scenarios* F *where* $|F| \leq k$ *whenever there is a path between* s *and* d *in* G^F*. A routing is* perfectly resilient *if it is perfectly* k*-resilient for all* k*.*

III. Theoretical Foundations for Optimization

Our overall aim is to present an approach for automatic generation of resilient failover tables. We shall first formulate a number of results that are later on used to significantly speedup the performance of our method by reducing the size of the searchable state space.

The first observation is that certain edges can be declared as blacklisted, meaning that they never need to be considered in any routing table as following these edges cannot help with delivering a packet to the destination node.

Definition 7 (Blacklisted Edge)**.** *Let* G *be a network and* d *a destination node. An edge* $e \in E$ *where* $r(e) = \{v, v'\}$ *is* blacklisted *at a node* v *for the destination* d*, if every path from* v' *to* d *contains the node* v*.*

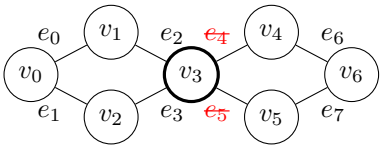


Fig. 2: Blacklisted edges at v_3 for the destination is v_0.

Figure 2 illustrates an example of blacklisted edges. If the destination node is v_0 then at node v_3 both e_4 and e_5 are blacklisted and we shall show that we can w.l.o.g. require that $R(e_2, v_3)$ and $R(e_3, v_3)$ do not contain e_4 nor e_5.

Definition 8 (Blacklisted-free Routing)**.** *Let G be a network and d a destination node. A routing R is a* blacklisted-free routing *if for every $e \in E$ and every $v \in V$ where $R(e, v) = (e_1, \ldots, e_\ell)$, the skipping sequence $e_1, \ldots, e_\ell$ does not contain any blacklisted edge at the node v for the destination d.*

The following theorem states that excluding blacklisted edges does not change the solvability of the routing problem. This allows us to limit the number of possible routings that need to be explored.

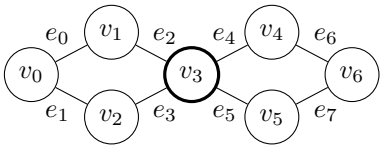
Theorem 1 (No Blacklisted Edges)**.** *If there is a perfectly k-resilient routing then there is also a perfectly k-resilient blacklisted-free routing.*

Sketch. Let G be a network and d a destination node. Let R be a perfectly k-resilient routing for the destination node d. We show that if for some edge e and a node v the skipping sequence of backup edges $(e_1, \ldots, e_\ell) = R(e, v)$ contains blacklisted edges, we can change the routing $R(e, v)$ so that it remains perfectly k-resilient but does not include blacklisted edges anymore. By repeating this argument, we can construct a perfectly k-resilient blacklisted-free routing.

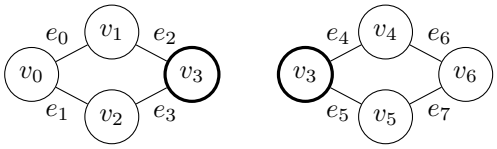
Let $R(e, v) = (e_1, \ldots, e_\ell)$ and let e_i, $1 \leq i \leq \ell$, be the first blacklisted edge in the list. Let us consider the sequence of hops (with no failed edges) starting at (e_i, v') such that $r(e_i) = \{v, v'\}$. Because e_i is blacklisted then clearly in the routing sequence there must be a next hop of the form (e', v), i.e. the path towards the destination d enters the node v again. Let us consider the first hop (e', v) on this path to the destination d where $R(e', v) = (e'_1, \ldots, e'_m)$ and where e'_1 is not blacklisted. Such a hop must exist as R delivers the packet to the destination d and this is not possible by only forwarding via the blacklisted edges. We modify R such that

$$R(e, v) := (e_1, e_2, \ldots, e_{i-1}, e'_1, e'_2, \ldots, e'_m)$$

which guarantees that the first i edges in the list are not blacklisted. We continue this process until $R(e, v)$ does not contain any blacklisted edges or until the beginning of the sequence of edges consists of only non-blacklisted edges where at least one of them repeats twice (this must eventually happen as there are only finitely many edges). In the latter case we truncate the backup edges at the first repeated non-blacklisted edge. Let us call this modified routing R'.



(a) Original network G with destination v_0.



(b) Networks G_2 and G_1 with destinations v_0 and v_3, resp.

Fig. 3: Decomposition at node v_3.

Let F be a failure scenario such that $|F| \leq k$. We shall now argue that if R delivers to d from every node in the network, so does R'. Clearly, if $e_1, \ldots, e_{i-1} \notin F$, then the modification at the entry $R(e, v)$ does not change any of the routings and the claim holds. Let us so analyze the case where all the edges $e_1, \ldots, e_{i-1}$ fail (belong to F). In this case, when following the original routing R, the next hop when arriving to v via the edge e will be to the node v' via the blacklisted edge e_i. Let us follow this routing sequence, until we encounter the next hop (e', v), i.e. enter the node v again. During this sequence, we assume that no edges are failed; if on the contrary some edges $e''_1, \ldots, e''_n$ after and including the blacklisted edge e_i are failed, we can instead consider the failure scenario $F' = F \setminus \{e''_1, \ldots, e''_n\}$ and because $|F'| \leq |F| \leq k$, we know that R will still deliver the packet to d also under such F'. After the packet made the hop (e', v), we know that none of the edges $e'_1, \ldots, e'_m$ are blacklisted and because R can deliver the packet to the destination d via the first non-failed edge e'_j on this list, so can do the routing R' that will take the edge e'_j directly without going through the blacklisted edges. □

Theorem 1 is the foundation that enables us to decompose the routing problem to the destination node d for a network G into two subproblems. Such a decomposition is attractive since solving two smaller problems separately is more efficient than solving the combined network and it can be moreover parallelized. Figure 3 depicts an illustrative example. Let us pick a node $v_c \in V$ and let V_c be the set of all nodes $v \in V \setminus \{v_c\}$ such that every path from v to the destination d contains the node v_c. If V_c is nonempty, we can divide the graph G into G_1 induced by the nodes $V_c \cup \{v_c\}$ and G_2 induced by the nodes $V \setminus V_c$. The solution for the graph G can now be obtained from the solutions to G_1 and G_2 due to the following corollary.

Corollary 1 (Decomposition)**.** *There is a perfectly k-resilient routing in G for the destination d if and only if there are perfectly k-resilient routings for G_1 with the destination v_c and G_2 with the destination d.*

Proof (sketch). By noticing that every routing of a packet from a node in G_1 to the destination d must visit the node v_c and

that every routing from a node in G_2 to d can avoid to enter G_1 (due to blacklisted edges and Theorem 1). □

The next theorem argues that perfectly k-resilient routings should never, as the first priority, send a packet back to the node from where it arrived. Not limiting the possible solutions by this constraint can lead to the exploration of routings where a packet unnecessarily reverts its direction without any reason.

Theorem 2 (No Send-back Edges as First Priority)**.** *Let G be a network and d a destination node. If there is a perfectly k-resilient routing R for G then there is also a perfectly k-resilient routing R' such that for every $e \in E$ and $v \in V$ if $R'(e, v) = (e_1, \ldots, e_\ell)$ then $e_1 \neq e$.*

Proof. Let R be a perfectly k-resilient routing and we modify it into R' as follows. Let $R(e, v) = (e, \ldots)$ for some edge e and some node v and let $r(e) = \{v, v'\}$. Now let $R'(e, v) := \epsilon$ (empty list of prioritised edges) and if $v \neq d$ then whenever $R(e', v') = (e_1, \ldots, e_j, e, \ldots)$ for some edge e' such that $e_i \neq e$ for all i, $1 \leq i \leq j$, we let $R'(e', v') := (e_1, \ldots, e_j, e'_1, \ldots, e'_m)$ where $R(e, v') = (e'_1, \ldots, e'_m)$. By repeating this for every e and v such that $R(e, v) = (e, \ldots)$, we obtain a routing R' that does not contain any send-back edges as the first priority. The routing in R' still delivers to the destination d whenever R does. This is because if we forward the packet on e to the node v, meaning that the edge e is not failed, v will send it immediately back to v' along the edge e (which is not failed even in the opposite direction) and the routing will continue according to the priorities given in $R(e, v')$. In R' we simply replace this unnecessary excursion to the node v by directly following the priorities in $R(e, v')$. □

Finally, we present a theorem proving that sending a packet back on the edge where it arrived from should only be used as the very last priority.

Theorem 3 (No Send-back Edges Unless Last Priority)**.** *Let G be a network and d a destination node. If there is a perfectly k-resilient routing R for G then there is also a perfectly k-resilient routing R' where for every $e \in E$ and $v \in V$ if $R(e, v) = (e_1, \ldots, e_\ell)$ then $e_i \neq e$ for all i, $1 \leq i < \ell$.*

Proof (sketch). Notice that if a packet arrives to the node v via the edge e then e cannot be failed and if e appears in the priority list of $R(e, v)$ then none of the edges after e can ever be used for forwarding. □

IV. Further Insights for Optimizations

When designing a routing, one may assume that it is safe to choose as the primary path to the destination one of the shortest paths. A similar conjecture is that for every k there exists a perfectly k-resilient routing. In this section, we shall formulate some of these conjectures and prove/disprove them for certain k values. These insights are also used to improve the performance of our tool.

Conjecture 1 (Existence of Perfectly k-resilient Routing)**.** *For any network G and any destination node d, there is a perfectly k-resilient routing.*

Conjecture 1 does not hold for all values of k. There is a network with 12 nodes and 22 edges that does not allow for perfect 14-resilience [22]. A smaller counterexample with 6 nodes and 9 edges that is not perfectly 3-resilient is given in [14]. Using our tool, we found a similar counterexample with only 8 edges. The problem for $k = 2$ is open, even though for circular routing (more strict routing policy than ours) there exists a counterexample. We reviewed this counterexample using our tool and observed that there actually does exist a perfect 2-resilient routing (using our routing model).

We shall now prove the claim for $k = 1$ (perfect 1-resilience is also known to hold for a more powerful routing model that does not require a prioritized list of edges but where the next-hop may instead depend on the actual failed link [22]). Moreover, we shall prove even a stronger claim that this routing can follow as the primary hop some (a priori fixed) shortest path edge, unless the packet arrives in the opposite direction of the shortest path edge. Thanks to this theorem, we can efficiently fix most of the default edges and thus greatly reduce the number of routing entries to be synthesized.

Theorem 4 (Existence of Perfectly 1-Resilient Routing)**.** *Let G be a network and let d be a destination node. Let for every node $v \in V$ fix an arbitrary edge e_v where $r(e_v) = \{v, v'\}$ such that e_v is on some shortest path from v to d. Then there exists a perfectly 1-resilient routing R for the destination d that moreover satisfies for every $e \in E$ and every $v \in V$ where $R(e, v) = (e_1, \ldots, e_\ell)$ that $e_1 = e_v$ unless $e = e_v$.*

Proof. By fixing the edges e_v for each node v, we also fix a unique shortest path from each node $v_0 \in V$ to the destination d as the following sequence of edges e_{v_0}, e_{v_1}, e_{v_2}, ... where $r(e_{v_i}) = \{v_i, v_{i+1}\}$ for all i. Clearly, such a shortest path eventually reaches the destination d. All such shortest paths for all nodes form a tree with d as the root. We now introduce some notation necessary for describing the 1-resilient routing R we want to construct for the network and the destination d.

Let $v \in V$. By $pre(v)$ we denote the set of all nodes that reach the node v by following their shortest paths to the destination d. Clearly $v \in pre(v)$ and $pre(d) = V$. Similarly, by $post(v)$ we denote the set of all nodes that are on the shortest path from v to d. Both v and d belong to $post(v)$.

We say that a node $v \in V$ is of *level* ℓ, if there is an edge e where $r(e) = \{v, v'\}$ such that the shortest path from v' to d intersects $post(v)$ in at most ℓ nodes. The minimum (lowest) level of a node v is denoted by $mlevel(v)$ and the corresponding edge e is called its *mlevel edge*. For example, a node v has minimum level 1 if there is an edge e where $r(e) = \{v, v'\}$ such that the shortest paths from v and v' to d only share the destination d.

Let $v \in V$ be a node. An edge e where $r(e) = \{v, v'\}$ and $e \neq e_v$ is called *backup edge* for v if either

- $mlevel(v) = \min_{v'' \in pre(v)} mlevel(v'')$ and e is its mlevel

edge, or

- $e = e_{v'}$ is the shortest path edge for some $v' \in pre(v)$ such that $\min_{v'' \in pre(v)} mlevel(v'') = \min_{v'' \in pre(v')} mlevel(v'')$.

In other words, the backup edge for v is either its *mlevel edge* in case v has the smallest *mlevel* among all nodes in $pre(v)$, or the backup edge is some shortest path edge for some node $v' \in pre(v)$ such that some smallest *mlevel* node from $pre(v)$ is also in $pre(v')$.

We now construct a perfectly 1-resilient routing R that from every node v as the first priority always follows the shortest path edge e_v, unless we arrived to v via the edge e_v itself (getting further away from d) in which case we select as the first priority edge the one that bring us to a node from $pre(v)$ with the lowest mlevel. If there are several nodes with the same mlevel, we pick any of them (e.g. in alphabetical order).

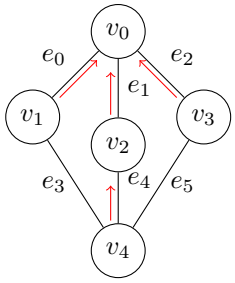
Let R be a routing such that for every edge $e \in E$ and every node $v \in V \setminus \{d\}$

- if $e \neq e_v$ then $R(e, v) = (e_v, e')$ where e' is the backup edge for v, or
- $R(e_v, v) = (e')$ where e' is the backup edge for v.

We must argue now that R is perfectly 1-resilient routing. Let F be a single edge failure scenario such that $F = \{e\}$. If e is not any of the shortest path edges then clearly every node in V still delivers the packet to the destination d by following the shortest path edges. Let us assume so that $e = e_v$ is a shortest path edge for some node v. Clearly, any node that is not in $pre(v)$ still delivers to d even if e_v fails. If the node v is not anymore connected to d (after e_v failed) then none of the nodes from $pre(v)$ is connected to d neither and we do not have to deliver the packet to d from these nodes.

We must so analyze the remaining case where e_v fails and the node v has an alternative path to d. We want to argue that every node from $pre(v)$ delivers to d following the routing R defined above. Clearly, every node from $pre(v)$ follows the shortest path route that brings the packet to the node v and where the next hop via the edge e_v is not available. Because from v there is an alternative path to d and because the distance of all nodes from $pre(v)$ to the destination d is at least $k = |post(v)| - 1$, the alternative path from v to d must eventually leave the nodes from $pre(v)$. Let $v'' \notin pre(v)$ be first such a node on this alternative path that is reached from some node $v' \in pre(v)$. Clearly, by following the shortest path from v'' to d, we will not use the failed edge e_v (otherwise v'' would belong to $pre(v)$). This means that $mlevel(v') < |post(v)|$. The routing R will by definition follow a (possibly different) path to d by exiting $pre(v)$ from a node with the minimum level over all nodes in $pre(v)$ which must be clearly less or equal to $mlevel(v')$. This implies that this path followed by the routing R cannot contain the failed edge e_v and hence delivers to d. This completes the proof. □

Theorem 4 proved for $k = 1$ may lead us to the following conjecture that tries to generalize this result to higher k and claims that a routing can always follow any chosen shortest path as its primary path.



(a) Network (shortest paths denoted by red arrows).

in-edge	node	pref. list
lb_{v_1}	v_1	e_0, e_3
e_3	v_1	e_0, e_3
lb_{v_2}	v_2	e_1, e_4
e_4	v_2	e_1, e_4
lb_{v_3}	v_3	e_2, e_5
e_5	v_3	e_2, e_5
lb_{v_4}	v_4	e_3, e_4, e_5
e_3	v_4	e_4, e_5, e_3
e_4	v_4	e_5, e_3, e_4
e_5	v_4	e_3, e_4, e_5

(b) Perfectly 2-resilient routing.

Fig. 4: A counterexample for Conjecture 2.

Conjecture 2. *Let G be a network and d a destination node. Let us for every node $v \in V$ fix an arbitrary edge e_v where $r(e_v) = \{v, v'\}$ such that e_v is on some shortest path from v to d. If there is a perfectly k-resilient routing R for G then there is also a perfectly k-resilient routing R' where for every $e \in E$ and $v \in V$ if $R'(e, v) = (e_1, \ldots, e_\ell)$ then $e_1 = e_v$ unless $e = e_v$.*

Surprisingly, this conjecture does not hold already for $k = 2$. We discovered this using our tool (described in the next section) by brute-forcing all connected simple graphs with up to 7 nodes or 9 edges. Figure 4 depicts the smallest counterexample found by our tool. The destination node is v_0 and the selected shortest path edges for v_1, v_2, v_3, v_4 are e_0, e_1, e_2, e_4, respectively. Let us inject a packet to the node v_4. Then we arrive to v_2 trough e_4 (because of the chosen shortest path). Suppose now that the link e_1 is down, so we return to v_4 using e_4 and without loss of generality let $R(e_4, v_4) = (e_3, \ldots)$ (in case that the first edge is e_5 the example works in the same way due to its symmetry). If the second failed edge is now e_0, we return to v_4 using e_3. Since we have to make the next hop to v_2 via e_4 (due to the shortest path requirement), we enter an infinite forwarding loop.

On the other hand, there exists a perfectly 2-resilient routing as shown in Table 4b, which however does not satisfy the conditions of Conjecture 2 because at the node v_4 it cycles through all three out-edges (depending on the in-edge) and does not necessarily follow the edge e_4 that was selected as the default one for v_4.

Conjecture 2 seems to fail because we follow the shortest path even when we already deviated from the default decisions during the forwarding. Hence we formulate Conjecture 3 that tries to avoid this pitfall.

Conjecture 3. *Let G be a network and d a destination node. Let us for every node $v \in V$ fix an arbitrary edge e_v where $r(e_v) = \{v, v'\}$ such that e_v is on some shortest path from v to d. If there is a perfectly k-resilient routing R for G then there is also a perfectly k-resilient routing R' where for every $e \in E$ and $v \in V$ if $R'(e, v) = (e_1, \ldots, e_\ell)$ then $e_1 \neq e$, and if the edge $e = \{v, v''\}$ is on some shortest path from v'' to d or $v = v''$ then $e_1 = e_v$.*

Conjecture	$k=1$	$k=2$	$k=3$
1	true by Theorem 4	holds up to 5 nodes or 6 edges	false [14]
2	true by Theorem 4	false see Figure 4	false [14]
3	true by Theorem 4	false see Figure 5	false [14]

TABLE I: Summary (loop-back edges are not counted).

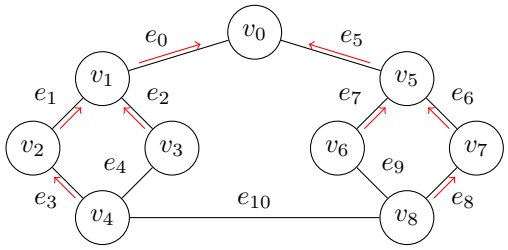
Unfortunately, Conjecture 3 also has a counterexample for $k = 2$ as depicted in Figure 5. This does not falsify Conjecture 1 for $k = 2$ as Table 5b shows a perfectly 2-resilient routing (that we independently verified by an automatic brute-force search). In this routing the node v_8 on the loop-back edge clearly does not follow the shortest path edge e_8.

To understand the counterexample, Table 5c shows the possible options for the routing satisfying the conditions of Conjecture 3 with up to two failed links. The edges in bold are enforced (there are no alternatives for them) and the entries with question mark are not relevant for our contradiction. Clearly, the first priority edges are fixed by the condition of the conjecture. Moreover, the priority list for $R(e_1, v_1)$ must be (e_0, e_2, e_1) otherwise a packet starting from v_2 would get into a loop if the edges e_0 and e_3 fail. The same reasoning can be used for $R(e_2, v_1)$, $R(e_7, v_5)$, $R(e_6, v_5)$. We have two options for the first priority edge of $R(e_{10}, v_4)$ and $R(e_{10}, v_8)$ but we can consider only one of them because of the routing rules in the two subgraphs are symmetric. Finally, the first priority edge for $R(e_3, v_4)$ must be e_{10} otherwise a packet starting from v_3 gets into a loop if e_0 is failed; the same reasoning applies for $R(e_4, v_4)$, $R(e_8, v_8)$ and $R(e_9, v_8)$. Table 5c reflects all these decisions. In the failure scenario $F = \{e_0, e_6\}$ all nodes are still connected to v_0 but a packet starting on the loop-back edge at v_2 now enters an infinite loop, contradicting Conjecture 3. For $k = 3$ our tool found even a smaller counterexample to Conjecture 3 with only 5 nodes and 7 edges.

A summary of the state-of-the-art is in Table I. It indicates that Conjecture 1 for $k = 2$ has no counterexample with less than 5 nodes or 6 edges and it remains an open problem. Even if some of the conjectures do not hold, for a concrete network, we can try to apply them anyway and if a resilient routing is found, this solves the synthesis problem. Our experiments show that this is indeed the case in most cases.

V. BDD Formulation of Routing Synthesis

At the heart of our tool SyPer lies a BDD formulation of the routing synthesis problem. We compactly represent multiple possible routings using binary decision diagrams (BDDs), a data structure introduced by Lee [18] that efficiently stores Boolean functions as rooted directed acyclic graphs (DAGs). Later Bryant [19] developed a reduced ordered version of BDDs with fixed variable ordering supporting isomorphic combinations for a more compact representation. Besides the basic logical operators, universal and existential quantifiers are



(a) Network topology (shortest paths denoted by red arrows).

in-edge	node	pref. list
lb_{v_1}	v_1	e_0, e_2, e_1
e_1	v_1	e_0, e_2, e_1
e_2	v_1	e_0, e_1, e_2
lb_{v_2}	v_2	e_1, e_3
e_1	v_2	e_3, e_1
e_3	v_2	e_1, e_3
lb_{v_3}	v_3	e_4, e_2
e_2	v_3	e_4, e_2
e_4	v_3	e_2, e_4
lb_{v_4}	v_4	e_3, e_4, e_{10}
e_3	v_4	e_4, e_{10}
e_4	v_4	e_{10}, e_3, e_4
e_{10}	v_4	e_3, e_4, e_{10}

in-edge	node	pref. list
lb_{v_5}	v_5	e_5, e_6, e_7
e_6	v_5	e_5, e_7, e_6
e_7	v_5	e_5, e_6, e_7
lb_{v_6}	v_6	e_7, e_9
e_7	v_6	e_9, e_7
e_9	v_6	e_7, e_9
lb_{v_7}	v_7	e_8, e_6
e_6	v_7	e_8, e_6
e_8	v_7	e_6, e_8
lb_{v_8}	v_8	e_9, e_8, e_{10}
e_8	v_8	e_{10}, e_9, e_8
e_9	v_8	e_8, e_{10}
e_{10}	v_8	e_9, e_8, e_{10}

(b) Perfectly 2-resilient routing.

in-edge	node	pref. list
e_1	v_1	$\mathbf{e_0, e_2, e_1}$
e_2	v_1	$\mathbf{e_0, e_1, e_2}$
e_7	v_5	$\mathbf{e_5, e_6, e_7}$
e_6	v_5	$\mathbf{e_5, e_7, e_6}$
e_1	v_2	$\mathbf{e_3, e_1}$
e_3	v_2	$\mathbf{e_1, e_3}$
e_2	v_3	$\mathbf{e_4, e_2}$
e_4	v_3	$\mathbf{e_2, e_4}$
e_7	v_6	$\mathbf{e_9, e_7}$

in-edge	node	pref. list
e_9	v_6	$\mathbf{e_7, e_9}$
e_6	v_7	$\mathbf{e_8, e_6}$
e_8	v_7	$\mathbf{e_6, e_8}$
e_{10}	v_4	$e_3, ?, ?$
e_3	v_4	$\mathbf{e_{10}}, ?, ?$
e_4	v_4	$\mathbf{e_{10}}, ?, ?$
e_{10}	v_8	$e_8, ?, ?$
e_8	v_8	$\mathbf{e_{10}}, ?, ?$
e_9	v_8	$\mathbf{e_{10}}, ?, ?$

(c) Partial routing for establishing a contradiction.

Fig. 5: Counterexample for Conjecture 3.

also supported. Moreover, all these operations are polynomial in the size of the underlying BDDs. We shall now give the details of our BDD encoding (for more technical details on the correspondence of our formulation with BDDs, see e.g. [23]).

Any finite set S can be encoded using $n = \lceil \log(|S|) \rceil$ Boolean variables. Let us pick a fixed enumeration of the elements: $s_0, s_1, ..., s_{|S|-1}$ and let the Boolean variables be $\overline{\mathbf{x}} = (x_0, x_1, ..., x_n)$. Any truth assignment μ to $\overline{\mathbf{x}}$ may be seen as a binary representation of a natural number $n(\mu) \in \mathbb{N}$ representing the $n(\mu)$'th element of S. Let $s(\mu)$ be the short representation of $s_{n(\mu)}$. By $\overline{\mathbf{x}}(s)$ we denote the Boolean encoding of $s \in S$.

For a better understanding, recall our example from Figure 1a. It has four nodes (v_0, v_1, v_2, v_3) that can be encoded using two Boolean variables where e.g. the assignment $(1, 1)$ refers to the node v_3. The 8 edges in our example, including the loop-back ones, can be similarly encoded using three Boolean variables.

A. Synthesis of perfectly 1-resilient routings

We first present the Boolean formulae used for synthesising perfectly 1-resilient routings and discuss the general case later on. We introduce the $state$ Boolean function expressing that a given edge belongs to a given node.

$$state(\overline{\mathbf{v}}, \overline{\mathbf{e}}) = \bigvee_{v \in V} \bigvee_{\substack{e \in E \\ v \in r(e)}} (\overline{\mathbf{v}}(v) \wedge \overline{\mathbf{e}}(e))$$

Similarly, we encode all additionally introduced loop-back edges as the $loopback$ Boolean function.

$$loopback(\overline{\mathbf{e}}) = \bigvee_{v \in V} \overline{\mathbf{e}}(lb_v)$$

Using these formulae we can express the valid possible routing entries (prioritized list) for every possible e in-edge and v node where $v \in r(e)$. For 1-resilient routings, we consider only the default out-edge (represented by the variables $\overline{\mathbf{e}}^d$) and one backup out-edge (represented by $\overline{\mathbf{e}}^b$).

$$\begin{aligned} \mathcal{V}_{v,e}(\overline{\mathbf{e}}^d, \overline{\mathbf{e}}^b) = state(\overline{\mathbf{v}}(v), \overline{\mathbf{e}}^d) \wedge state(\overline{\mathbf{v}}(v), \overline{\mathbf{e}}^b) \\ \wedge \neg loopback(\overline{\mathbf{e}}^d) \wedge \neg loopback(\overline{\mathbf{e}}^b) \end{aligned} \tag{1}$$

Encoding of valid transitions. The basic building block of packet routing is the valid transition (next hop) from an (e_{in}, v) state to (e_{out}, v') under a given $F = \{f\}$ failure scenario. The Boolean formula $\mathcal{T}$ describes all valid transitions of a packet where $\overline{\mathbf{e}}_{in}, \overline{\mathbf{v}}, \overline{\mathbf{e}}_{out}, \overline{\mathbf{v}}'$ encode the in-edge, the current node, the chosen out-edge and the next node. We consider maximum one failed edge encoded by $\bar{\mathbf{f}}$. The failure scenario without failed links ($\emptyset \subseteq E$) is represented by assigning a lb_d loop-back edge to $\bar{\mathbf{f}}$ where d is the destination node. The $\overline{\mathbf{e}}^d_{v,e}, \overline{\mathbf{e}}^b_{v,e}$ variables define the default and backup edges in the routing. Predefined and fixed routing entries can be substituted into the formula.

$$\begin{aligned} \mathcal{T}(\overline{\mathbf{e}}_{in}, \overline{\mathbf{v}}, \overline{\mathbf{e}}_{out}, \overline{\mathbf{v}}', \bar{\mathbf{f}}, [\overline{\mathbf{e}}^d_{v,e}, \overline{\mathbf{e}}^b_{v,e} : {}^{e \in E}_{v \in r(e)}]) = \\ \bigwedge_{e \in E \wedge v \in r(e)} \mathcal{V}_{v,e}(\overline{\mathbf{e}}^d_{v,e}, \overline{\mathbf{e}}^b_{v,e}) \wedge \\ state(\overline{\mathbf{v}}, \overline{\mathbf{e}}_{in}) \ \wedge state(\overline{\mathbf{v}}, \overline{\mathbf{e}}_{out}) \ \wedge state(\overline{\mathbf{v}}', \overline{\mathbf{e}}_{out}) \ \wedge \\ \overline{\mathbf{e}}_{out} \neq \bar{\mathbf{f}} \ \wedge \\ \bigwedge_{e \in E} \bigwedge_{v \in r(e)} (\overline{\mathbf{e}}_{in}(e) \wedge \overline{\mathbf{v}}(v) \wedge \overline{\mathbf{e}}^d_{v,e} \neq \bar{\mathbf{f}} \implies \overline{\mathbf{e}}_{out} = \overline{\mathbf{e}}^d_{v,e}) \\ \bigwedge_{e \in E} \bigwedge_{v \in r(e)} (\overline{\mathbf{e}}_{in}(e) \wedge \overline{\mathbf{v}}(v) \wedge \overline{\mathbf{e}}^d_{v,e} = \bar{\mathbf{f}} \implies \overline{\mathbf{e}}_{out} = \overline{\mathbf{e}}^b_{v,e}) \end{aligned} \tag{2}$$

In the second line, we make sure that the provided routing entries are valid and in the next two lines we require that the transition step is aligned with the network topology and the current failure scenario. In the last two lines, we enforce the use of the appropriate routing entries.

Encoding of delivery. Let $d \in V$ be the destination we want to reach. A packet arriving to the $v \in V$ node from the $e_{in} \in E$ input edge is delivered (in one step) to $d \in V$ trough the $e_{out} \in E$ under the $\{f\} \subseteq E$ failure scenario iff they form a valid transition as described before. In short, a packet in the (e_{in}, v) state is delivered. By the nature of packet forwarding, every packet at the (e'_{in}, v') state is also delivered to d if it transitions to (e_{in}, v). Using the introduced notation, all delivery within graph G can be described using a Boolean formula by calculating the fixed point $\mathcal{D}$ starting from $\mathcal{D}_0(\overline{\mathbf{e}}_{in}, \overline{\mathbf{v}}, \bar{\mathbf{f}}, [\overline{\mathbf{e}}^d_{v,e}, \overline{\mathbf{e}}^b_{v,e} : {}^{e \in E}_{v \in r(e)}]) = (\overline{\mathbf{v}} = \overline{\mathbf{d}})$ where $\overline{\mathbf{d}}$ is the Boolean encoding of d destination node:

$$\begin{aligned} \mathcal{D}_{n+1}(\overline{\mathbf{e}}_{in}, \overline{\mathbf{v}}, \bar{\mathbf{f}}, [\overline{\mathbf{e}}^d_{v,e}, \overline{\mathbf{e}}^b_{v,e} : {}^{e \in E}_{v \in r(e)}]) = \\ \mathcal{D}_n(\overline{\mathbf{e}}_{in}, \overline{\mathbf{v}}, \bar{\mathbf{f}}, [\overline{\mathbf{e}}^d_{v,e}, \overline{\mathbf{e}}^b_{v,e} : {}^{e \in E}_{v \in r(e)}]) \vee \\ \Big(\exists \overline{\mathbf{v}}', \overline{\mathbf{e}}_{out} : \mathcal{T}(\overline{\mathbf{e}}_{in}, \overline{\mathbf{v}}, \overline{\mathbf{e}}_{out}, \overline{\mathbf{v}}', \bar{\mathbf{f}}, [\overline{\mathbf{e}}^d_{v,e}, \overline{\mathbf{e}}^b_{v,e} : {}^{e \in E}_{v \in r(e)}]) \\ \wedge \ \mathcal{D}_n(\overline{\mathbf{e}}_{out}, \overline{\mathbf{v}}', \bar{\mathbf{f}}, [\overline{\mathbf{e}}^d_{v,e}, \overline{\mathbf{e}}^b_{v,e} : {}^{e \in E}_{v \in r(e)}]) \Big) \end{aligned} \tag{3}$$

Clearly the sets $\mathcal{D}_n$ are increasing and the minimum fixed point of Equation 3 is obtained after some finite number of iterations N, where $\mathcal{D}_{N+1} = \mathcal{D}_N = \mathcal{D}$.

Encoding of perfectly 1-resilient routings. The previously introduced Boolean function $\mathcal{D}$ describes all possible delivery scenario. We are only interested in routings that deliver from every possible (lb_v, v) state where $v \in V \setminus \{d\}$ under every $\{f\} \subseteq E$ failure scenario. This can be formulated as follows:

$$\begin{aligned} \mathcal{P}([\overline{\mathbf{e}}^d_{v,e}, \overline{\mathbf{e}}^b_{v,e} : {}^{e \in E}_{v \in r(e)}]) = \forall \bar{\mathbf{f}} : \forall (\overline{\mathbf{lb}}_v, \overline{\mathbf{v}}) : \exists \overline{\mathbf{e}}_{out} : \\ \gamma(v(\overline{\mathbf{v}}), f(\bar{\mathbf{f}}), d) \implies \\ \mathcal{D}(\overline{\mathbf{lb}}_v, \overline{\mathbf{v}}, \overline{\mathbf{e}}_{out}, \bar{\mathbf{f}}, [\overline{\mathbf{e}}^d_{v,e}, \overline{\mathbf{e}}^b_{v,e} : {}^{e \in E}_{v \in r(e)}]) \end{aligned} \tag{4}$$

where $\gamma(v_1, e, v_2) : V \times E \times V \longrightarrow \{true, false\}$ is true iff there is a path between v_1 and v_2 not containing e. Now the Boolean function $\mathcal{P}$ represents exactly all perfectly 1-resilient routings that from every node deliver a packet to its destination whenever they are connected in the given failure scenario.

B. Synthesis of perfectly k-resilient routings

The previously described method can be further extended to handle $\{f_1, ...f_k\} \subseteq E$ multi-failure scenarios by introducing multiple $\overline{\mathbf{e}}^{b_1}_{e_{in},v}, ..., \overline{\mathbf{e}}^{b_k}_{e_{in},v}$ variables for additional backup edges and $\bar{\mathbf{f}}_1, ..., \bar{\mathbf{f}}_k$ variables for the failures. Furthermore, Equation 1 must require that these new backup edges are compatible with the network topology and that they are not loop-back edges. Equation 2 must ensure that the $\overline{\mathbf{e}}_{out}$ out-edge is not equal to any of the failures and that edges with a higher priority are all failed. The fixed-point calculation described in Equation 3 remains the same. Finally, the γ function in Equation 4 must be generalized to multiple failed edges.

C. Application of our optimisations

Our results from Sections III and IV can be applied to speeding up the BDD computations showcasing their practicality. Our strategies can be divided into two main categories.

Preprocessing strategies are different ways of providing predefined rules as defined in Conjecture 2 and 3, thus reducing the number of variables in the BDD computation. Preprocessing strategies are used *before* the BDD computation described in Section V. Besides the use of our conjectures, if there is only one option for a given routing entry then we pick

that one instead of letting the BDD figure it out. Thanks to these methods, we can reduce the number of $\overline{\mathbf{e}}^d_{e_{in},v}$ and $\overline{\mathbf{e}}^b_{e_{in},v}$ variables while the BDD computation remains unchanged.

Runtime strategies are different ways of constraining the BDD computation to increase its efficiency. More concretely, we leverage Theorem 1, Theorem 3, and Corollary 1. Runtime strategies are applied *during* the BDD computation by slightly modifying the method described in Section V. When applying Theorem 1, Equation 1 must also require that the chosen $\overline{\mathbf{e}}^d$, $\overline{\mathbf{e}}^b$ edges are not blacklisted. Theorem 3 also modifies Equation 1 by requiring that only the (last) backup edge can encode the e edge if we have other options for higher priorities. Corollary 1 decomposes the network into multiple smaller graphs that are solved separately and the routings later combined together. Besides relying on our theoretical results, we can also check for solutions after every step of the fixed-point calculation described in Equation 3 and perform an *early exit* once $\mathcal{P}([\overline{\mathbf{e}}^d_{v,e}, \overline{\mathbf{e}}^b_{v,e} : {}^{e\in E}_{v\in r(e)}])$ contains at least one solution.

VI. Performance Evaluation

We implemented our solution using the CUDD [24] backend of the Omega [25] Python library. For the evaluation, we use the Topology Zoo dataset [21] to measure the performance on realistic networks. We consider all the 243 connected graphs with up to 700 nodes. The measurements are run on Intel Xeon Gold 6209U (2.10GHz) CPUs with a 128GB memory limit and 20 minute timeout. A reproducibility package is available at [26]. In the performance plots, the x-axis contains the network topologies sorted (independently for each method) by the respective running times (on y-axis).

Perfectly 1-resilient routings. We use Theorem 4 to select the default (first priority) routing entries for each topology in the preprocessing phase and then measure the effect of the optimizations discussed earlier. The results are shown in Figure 6. As expected, exiting at the first solution is faster than finding all perfectly 1-resilient routings. Theorems 1 and 3 show a similar performance gain when applied separately. Corollary 1 yields better results than Theorem 1 since the decomposition also eliminates blacklisted edges besides creating smaller subtasks. Combining all our optimizations yields considerably better performance than using them individually. All together, we are able to generate the BDDs describing all resilient routings for 221 topologies out of 243 within the given resource limits. This number increases to 232 when the computation is halted after the first resilient routing is found.

Perfectly 2-resilient routings. We use Conjecture 3 to constrain the default routing entries for the networks. Note that this does not guarantee that we always find a 2-resilient routing if it exists (see Section IV). Besides having fewer default entries fixed ahead of time, the method has to find both the first and second backup entries too. The results are depicted in Figure 7. Although one can observe the increased resource demands, combining our runtime strategies still terminates for 124 topologies, where 107 of them yield a solution. This number increases to 136 with an early exit, providing us with 120 perfectly 2-resilient routing. Interestingly, Theorem 1 shows a larger impact than Theorem 3 as opposed to the perfectly 1-resilient case.

Effect of different missing entries. We now use the Oxford topology from the Topology Zoo to gain insights on the running time of our tool compared to how many routing entries are fixed in preprocessing and compared to how many must by synthesized. First, we fix some perfectly 1-resilient routing and then randomly remove a given number of default and backup entries. This method ensures, that the problem always has a solution. Finally, we measure the time required for the BDD calculation to fill in the missing entries in order to synthetise all perfectly 1-resilient routings (by filling in the missing entries). The runtimes are depicted in Figure 8. One can observe that missing default entries cause a much steeper increase. This reinforces the importance of providing well-chosen fixed default routes as we discussed in Section IV.

VII. Related Work

The design of fast rerouting algorithms has already been studied intensively. While there exists much interesting work on how to efficiently react to link failures in the control plane, e.g. [27], [28], the reaction times provided by these solutions are significantly higher compared to mechanisms in the dataplane [5]—the focus of our paper. Dataplane-based failover mechanisms are implemented in most protocols, IP [29]–[31], MPLS [13], [32], BGP [33], SDN [34], [35], segment routing [36], [37] , etc. We refer to the recent survey by Chiesa et al. [4].

Fast failover algorithms for the dataplane can be categorized according to whether they require dynamic packet headers modifications, according to which header fields a router needs to be able to match, and whether they need randomization. By rewriting packet headers, it becomes possible to, e.g., carry failure information in the packets [7], [38], [39] or to employ graph exploration algorithms [8], [40]. It is also known that the design of resilient fast failover algorithms is simplified if the source can be matched [9], [10], [41] and if forwarding can be randomized per packet [11], [12]. In this paper we are interested in practical solutions with minimal assumptions without packet header modifications and random generators.

Feigenbaum et al. [22] initiated the study of perfect resilience in networks without header modifications and showed that it is not always possible to achieve, and presented a deterministic algorithm which is provably 1-resilient; Dai et al. [17] later generalized this algorithm for 2-resilience, however, requiring source matching. For the case of circular routing, there is an example of a network without any 2-resilient routing [41], however, this routing model is stricter than ours. As a result, the problem of perfect 2-resilience studied in this paper is still open. Foerster et al. [14] established a connection between perfect resilience and graph minors to derive possibility/impossibility results on different graph classes. Contrary to our paper, none of these approaches have been implemented.

The state-of-the-art approach to design highly resilient failover algorithms is based on arc-disjoint arborescence cov-

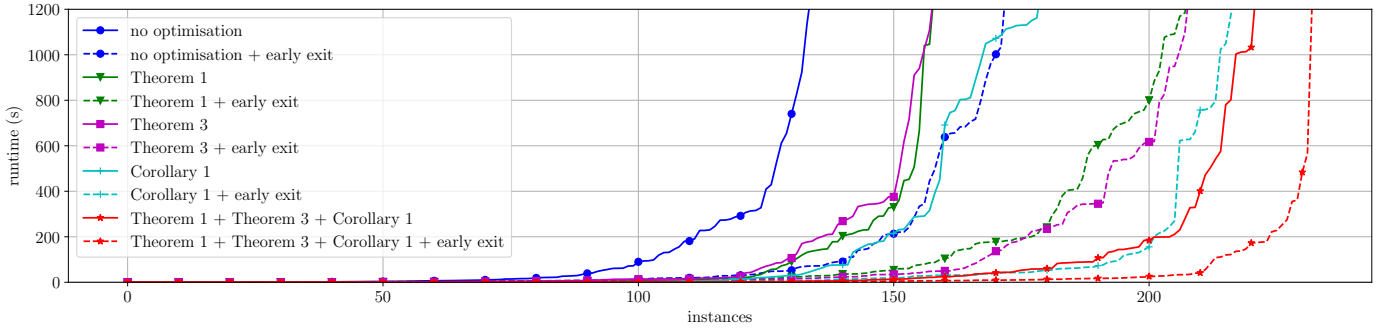


Fig. 6: Effect of different runtime strategies on generating perfectly 1-resilient routings for the connected Topology Zoo graphs.

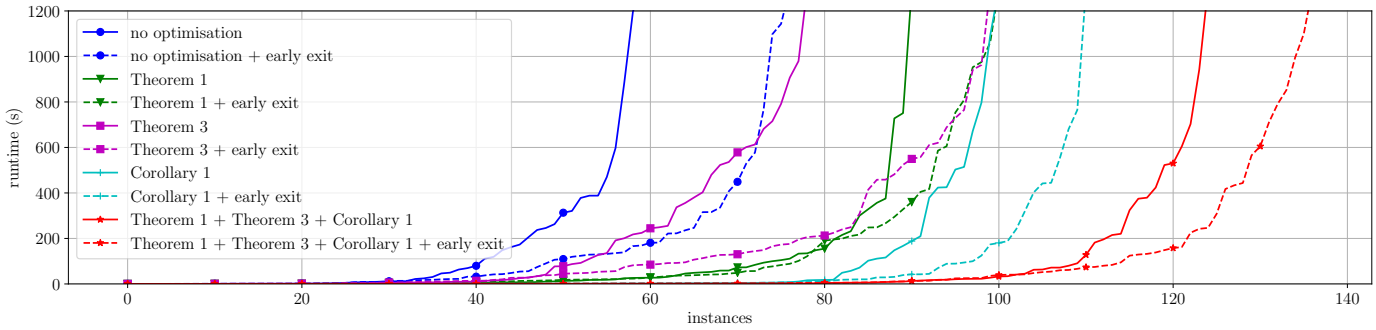


Fig. 7: Effect of different runtime strategies on generating perfectly 2-resilient routings for the connected Topology Zoo graphs.

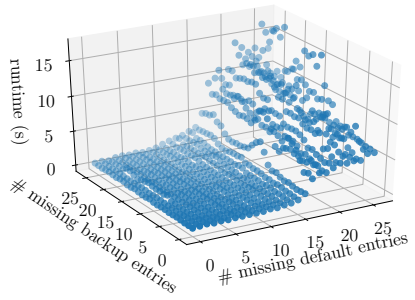


Fig. 8: Effect of missing default and backup routing entries.

ers [10], [11], [41]–[45]. This approach generalizes the widely-used approaches based on spanning trees [46]. However, while this approach may work well on graphs which are homogeneously k-connected (this is still an open question), it is not well-suited if directly applied to the general setting considered in our paper. Yang et al. [6] initiated the study of approaches beyond spanning trees and acyclic graphs, however, without providing strong formal connectivity guarantees.

Synthesis and formal approaches for other aspects of networking are also widely studied in the literature [47]–[59]. In particular, SyNET [47] synthesizes correct network configurations and inputs for a stratified Datalog program, supporting multiple interacting routing protocols such as BGP and OSPF. In the context of MPLS networks, AalWines [23] supports an automated what-if verification of the policy-compliance of routes under multiple failures in MPLS networks. There is also much work on automated approaches for correctly updating network configurations. For example, NetComplete [57] assists operators in modifying existing network-wide configurations to comply with new routing policies. Notably, the network update tool AllSynth [60] also uses BDDs. However, we are not aware of any approach supporting an automated synthesis of perfectly resilient routings as we do in SYPER.

VIII. CONCLUSION

We presented SYPER, a synthesis approach that generates perfectly k-resilient FRR rules in an automated and provably correct manner. In other words, our approach guarantees that we can always route traffic between two nodes if they are still connected in the underlying physical network and there are no more than k link failures. SYPER focuses on an efficient synthesis of prioritised forwarding rules and leverages the BDD technology. Our empirical evaluation shows that SYPER, in combination with the theoretical underpinning developed in this paper, is feasible and can synthesize robust network configurations on a large range of real-world network topologies. As a future work, we plan to increase the solution's scalability and consider alternative tools like QSAT solvers.

The authors have provided public access to their code and experimental data at [26].

ACKNOWLEDGMENT

Research supported by the Vienna Science and Technology Fund (WWTF) project, grant 10.47379/ICT19045 (WHATIF), 2020-2024, by the Danish DFF project QASNET, and by the Villum Investigator Grant S4OS, 2020–2027.

REFERENCES

[1] A. Markopoulou *et al.*, "Characterization of failures in an operational IP backbone network," *IEEE/ACM Trans. Netw.*, vol. 16, no. 4, pp. 749–762, 2008.

[2] R. Govindan *et al.*, "Evolve or die: High-availability design principles drawn from googles network infrastructure," in *SIGCOMM*, 2016, p. 58–72.

[3] P. Gill *et al.*, "Understanding network failures in data centers: measurement, analysis, and implications," in *SIGCOMM*. ACM, 2011, pp. 350–361.

[4] M. Chiesa *et al.*, "A survey of fast recovery mechanisms in the data plane," *TechRxiv*, May 2020.

[5] J. Liu *et al.*, "Ensuring connectivity via data plane mechanisms," in *NSDI*, 2013.

[6] B. Yang *et al.*, "Keep forwarding: Towards k-link failure resilient routing," in *INFOCOM*, 2014, pp. 1617–1625.

[7] M. Canini *et al.*, "A distributed and robust SDN control plane for transactional network updates," in *INFOCOM*. IEEE, 2015, pp. 190–198.

[8] M. Borokhovich *et al.*, "Provable data plane connectivity with local fast failover: introducing openflow graph algorithms," in *HotSDN*. ACM, 2014, pp. 121–126.

[9] M. Chiesa *et al.*, "Exploring the limits of static failover routing (v4)," vol. abs/1409.0034.v4, 2016.

[10] ——, "The quest for resilient (static) forwarding tables," in *INFOCOM*, 2016, pp. 1–9.

[11] ——, "On the resiliency of randomized routing against multiple edge failures," in *ICALP*, 2016, pp. 134:1–134:15.

[12] G. Bankhamer *et al.*, "Local fast rerouting with low congestion: A randomized approach," in *ICNP*, 2019.

[13] J. S. Jensen *et al.*, "P-rex: fast verification of MPLS networks with multiple link failures," in *CoNEXT*. ACM, 2018, pp. 217–227.

[14] K.-T. Foerster *et al.*, "On the feasibility of perfect resilience with local fast failover," in *Proc. SIAM APOCS*, 2021, pp. 55–69.

[15] J. Feigenbaum *et al.*, "Brief announcement: On the resilience of routing tables," in *Proc. ACM PODC*, 2012, p. 237–238.

[16] T. Holterbach *et al.*, "Blink: Fast connectivity recovery entirely in the data plane," in *USENIX NSDI*, 2019, pp. 161–176.

[17] W. Dai *et al.*, "A tight characterization of fast failover routing: Resiliency to two link failures is possible," in *Proc. ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, 2023, p. 153–163.

[18] C. Y. Lee, "Representation of switching circuits by binary-decision programs," *The Bell System Technical Journal*, vol. 38, no. 4, pp. 985–999, 1959.

[19] Bryant, "Graph-based algorithms for boolean function manipulation," *IEEE Transactions on Computers*, vol. C-35, no. 8, pp. 677–691, 1986.

[20] M. Borokhovich *et al.*, "The show must go on: Fundamental data plane connectivity services for dependable sdns," *Computer Communications*, vol. 116, pp. 172–183, 2018.

[21] S. Knight *et al.*, "The internet topology zoo," *IEEE Journal on Selected Areas in Communications*, vol. 29, no. 9, pp. 1765–1775, 2011.

[22] J. Feigenbaum *et al.*, "Brief announcement: on the resilience of routing tables," in *PODC*. ACM, 2012, pp. 237–238.

[23] P. G. Jensen *et al.*, "Aalwines: A fast and quantitative what-if analysis tool for mpls networks," in *ACM CoNEXT*, 2020, p. 474–481.

[24] F. Somenzi, "Cudd: Cu decision diagram package release 3.0.0," *University of Colorado at Boulder*, 2015. [Online]. Available: http://vlsi.colorado.edu/ fabio/CUDD/

[25] "Omega python package," 2023. [Online]. Available: https://github.com/tulip-control/omega

[26] C. Györgyi *et al.*, "Reproducibility Package for SYPER: Synthesis of Perfectly Resilient Local Fast Re-Routing Rules for Highly Dependable Networks," Jan. 2024, https://zenodo.org/doi/10.5281/zenodo.10479017. [Online]. Available: https://zenodo.org/doi/10.5281/zenodo.10479017

[27] C. Busch *et al.*, "Analysis of link reversal routing algorithms for mobile ad hoc networks," in *ACM SPAA*, 2003, p. 210–219.

[28] E. Gafni and D. P. Bertsekas, "Distributed algorithms for generating loop-free routes in networks with frequently changing topology," *IEEE Trans. Communications*, vol. 29, no. 1, pp. 11–18, 1981.

[29] A. Atlas and A. Zinin, "Basic specification for IP fast reroute: Loop-free alternates," *RFC*, vol. 5286, pp. 1–31, 2008.

[30] G. Rétvári *et al.*, "IP fast reroute: Loop free alternates revisited," in *INFOCOM*. IEEE, 2011, pp. 2948–2956.

[31] A. Bashandy *et al.*, "Topology independent fast reroute using segment routing," *Working Draft*, 2018.

[32] P. Pan *et al.*, "Fast reroute extensions to RSVP-TE for LSP tunnels," *RFC*, vol. 4090, pp. 1–38, 2005.

[33] C. Filsfils *et al.*, "Bgp prefix independent convergence," *Cisco*, 2011.

[34] Switch Specification 1.3.1, "OpenFlow," in *https://bit.ly/2VjOO77*, 2013.

[35] M. Chiesa *et al.*, "PURR: a primitive for reconfigurable fast reroute: hope for the best and program for the worst," in *CoNEXT*. ACM, 2019, pp. 1–14.

[36] P. François *et al.*, "Topology independent fast reroute using segment routing," 2014. [Online]. Available: http://hdl.handle.net/20.500.12761/32

[37] K.-T. Foerster *et al.*, "Ti-mfa: keep calm and reroute segments fast," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2018, pp. 415–420.

[38] T. Elhourani *et al.*, "IP fast rerouting for multi-link failures," in *INFOCOM*, 2014.

[39] B. Stephens *et al.*, "Plinko: Building provably resilient forwarding tables," in *Proceedings of the Twelfth ACM Workshop on Hot Topics in Networks*, 2013, pp. 1–7.

[40] O. Reingold, "Undirected connectivity in log-space," *J. ACM*, vol. 55, no. 4, pp. 17:1–17:24, 2008.

[41] M. Chiesa *et al.*, "On the resiliency of static forwarding tables," *IEEE/ACM Trans. Netw.*, vol. 25, no. 2, pp. 1133–1146, 2017.

[42] K.-T. Foerster *et al.*, "Grafting arborescences for extra resilience of fast rerouting schemes," in *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*. IEEE, 2021, pp. 1–10.

[43] ——, "Improved fast rerouting using postprocessing," in *SRDS*. IEEE, 2019, pp. 173–182.

[44] ——, "Bonsai: Efficient fast failover routing using small arborescences," in *DSN*. IEEE, 2019, pp. 276–288.

[45] ——, "Local fast failover routing with low stretch," *Computer Communication Review*, vol. 48, no. 1, pp. 35–41, 2018.

[46] J. Tapolcai, "Sufficient conditions for protection routing in ip networks," *Optimization Letters*, vol. 7, no. 4, pp. 723–730, 2013.

[47] A. El-Hassany *et al.*, "Network-wide configuration synthesis," in *Computer Aided Verification: 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part II 30*. Springer, 2017, pp. 261–281.

[48] J. McClurg *et al.*, "Efficient synthesis of network updates," in *ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, 2015, p. 196–207.

[49] A. El-Hassany *et al.*, "NetComplete: Practical network-wide configuration synthesis with autocompletion," in *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, 2018, pp. 579–594.

[50] T. Schneider *et al.*, "On the complexity of network-wide configuration synthesis," in *2022 IEEE 30th International Conference on Network Protocols (ICNP)*. IEEE, 2022, pp. 1–11.

[51] L. Beurer-Kellner *et al.*, "Learning to configure computer networks with neural algorithmic reasoning," *Advances in Neural Information Processing Systems*, vol. 35, pp. 730–742, 2022.

[52] K. G. Larsen *et al.*, "Allsynth: A bdd-based approach for network update synthesis," in *Science of Computer Programming (SCICO)*, vol. 230, 2023.

[53] S. Steffen *et al.*, "Probabilistic verification of network configurations," in *Proc. ACM SIGCOMM*, 2020, p. 750–764.

[54] S. Prabhu *et al.*, "Plankton: Scalable network configuration verification through model checking," in *17th USENIX Symposium on Networked Systems Design and Implementation ({NSDI} 20)*, 2020, pp. 953–967.

[55] J. McClurg *et al.*, "Efficient synthesis of network updates," in *Acm Sigplan Notices*, vol. 50, no. 6. ACM, 2015, pp. 196–207.

[56] B. Finkbeiner *et al.*, "Model checking data flows in concurrent network updates (full version)," *arXiv preprint arXiv:1907.11061*, 2019.

[57] A. El-Hassany *et al.*, "Netcomplete: Practical network-wide configuration synthesis with autocompletion," in *USENIX NSDI*, 2018, pp. 579–594.

[58] C. J. Anderson *et al.*, "Netkat: Semantic foundations for networks," *Acm sigplan notices*, vol. 49, no. 1, pp. 113–126, 2014.

[59] S. Schmid *et al.*, "R-mpls: Recursive protection for highly dependable mpls networks," in *Proc. ACM CoNEXT*, 2022, p. 276–292.

[60] K. G. Larsen *et al.*, "Allsynth: Transiently correct network update synthesis accounting for operator preferences," in *Proc. TASE*, 2022, pp. 344–362.