Fast Re-Routing in Networks: On the Complexity of Perfect Resilience

Matthias Bentert
University of Bergen, Norway
TU Berlin, Germany

Esra Ceylan
TU Berlin, Germany
Institute of Science and Technology Austria, Klosterneuburg, Austria

Valentin Hübner
Institute of Science and Technology Austria, Klosterneuburg, Austria

Stefan Schmid
TU Berlin, Germany

Jiří Srba
Aalborg University, Denmark

Abstract

To achieve fast recovery from link failures, most modern communication networks feature fully decentralized fast re-routing mechanisms. These re-routing mechanisms rely on pre-installed static re-routing rules at the nodes (the routers), which depend only on local failure information, namely on the failed links *incident* to the node. Ideally, a network is *perfectly resilient*: the re-routing rules ensure that packets are always successfully routed to their destinations as long as the source and the destination are still physically connected in the underlying network after the failures. Unfortunately, there are examples where achieving perfect resilience is not possible. Surprisingly, only very little is known about the algorithmic aspect of when and how perfect resilience can be achieved.

We investigate the computational complexity of analyzing such local fast re-routing mechanisms. Our main result is a negative one: we show that even checking whether a *given* set of static re-routing rules ensures perfect resilience is coNP-complete. Additionally, we investigate other fundamental variations of the problem. In particular, we show that our coNP-completeness proof also applies to scenarios where the re-routing rules have specific patterns (known as *skipping* in the literature).

On the positive side, for scenarios where nodes do not have information about the link from which a packet arrived (the so-called in-port), we present a linear-time algorithm to realize perfect resilience whenever possible (which we show can also be determined in linear time).

2012 ACM Subject Classification Networks → Network protocol design; Networks → Network properties; Theory of computation → Problems, reductions and completeness

Keywords and phrases routing in computer networks, fast re-route, perfect resilience, complexity

Digital Object Identifier 10.4230/LIPIcs.OPODIS.2025.31

Funding *Matthias Bentert*: ERC Horizon 2020 research and innovation programme (grant agreement No. 819416) and ERC Consolidator grant AdjustNet (agreement No. 864228).
Esra Ceylan: German Research Foundation (DFG) project ReNO, Schwerpunktprogramm: Resilienz in Vernetzten Welten – Beherrschen von Fehlern, Überlast, Angriffen und dem Unbekannten (SPP 2378).
Stefan Schmid: German Research Foundation (DFG) project ReNO, Schwerpunktprogramm: Resilienz in Vernetzten Welten – Beherrschen von Fehlern, Überlast, Angriffen und dem Unbekannten (SPP 2378).

© Matthias Bentert, Esra Ceylan, Valentin Hübner, Stefan Schmid, and Jiří Srba;
licensed under Creative Commons License CC-BY 4.0
29th International Conference on Principles of Distributed Systems (OPODIS 2025).
Editors: Andrei Arusoaie, Emanuel Onica, Michael Spear, and Sara Tucci-Piergiovanni; Article No. 31;
pp. 31:1–31:16
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

1.1 Context and Motivation

Communication networks form a critical backbone of many distributed systems, whether they are running in enterprises, in data centers, or are geographically distributed across the Internet. In order to meet their stringent availability requirements, modern networks feature local fast re-routing mechanisms: each router (henceforth called node) has conditional packet forwarding rules which depend on the status of its links. These rules are configured *ahead of time*, without knowledge of possible link failures, and allow routers to forward packets along alternative links in case of failures, in a fully decentralized manner.

However, configuring such conditional forwarding rules to provide a high resilience, even under multiple link failures, is algorithmically challenging, as the rules can only depend on *local information*: a node is not aware of possible additional link failures downstream, in other parts of the network. Hence, if no care is taken, the local re-routing rules of different nodes can easily result in forwarding loops.

Ideally, a network and its local fast re-routing mechanism provide *perfect resilience*: the local re-routing rules ensure that as long as the source is still connected to the destination *in the underlying network* after the failures, then the packet is also successfully routed to the destination *on the network layer*. Already at ACM PODC 2012, Feigenbaum et al. [10] gave an example which shows that certain networks inherently cannot be configured to provide perfect resilience.

On the positive side, it is known that under a single link failure, it is always possible to successfully route packets to their destinations, as long as the underlying network is connected [10, 11]. But also for multiple link failures, it is sometimes possible to achieve perfect resilience, for example on networks whose topologies are based on graph families closed under link subdivision, such as outerplanar graphs [10, 11].

It is also known that achievable resilience depends on the specific local information that the forwarding rules can rely on. Although it is typically assumed that the forwarding rules can depend on the status of the incident links, the packet's destination, and the link incident to the node from which the packet arrives (the so-called *in-port*), the achievable resilience can increase if the rules can additionally also observe the packet's source. In particular, Dai et al. [6] showed that it is always possible to tolerate two link failures if the packet's source can be taken into account.

However, today, we generally still do not have a good understanding of the scenarios in which perfect resilience can be achieved.

This paper initiates the study of the computational complexity of configuring local fast re-routing algorithms. In particular, we investigate the question whether the resilience of a network can be verified efficiently:

- *Does a given network and its local fast re-route mechanism provide perfect resilience?*

Furthermore, we are interested in scenarios where networks can be efficiently configured.

- *In which scenarios can perfectly resilience networks be configured efficiently?*

Our main contribution in this paper is the proof that the first problem is coNP-complete. On the positive side, we also show that in a subclass of practically relevant scenarios, the second problem can be solved efficiently.

1.2 Contributions

In summary, we make the following contributions. We show that in the standard scenario where nodes know the in-port (the link on which a packet arrived), verifying whether a given network and its configuration provide perfect resilience is coNP-complete. This is even the case when the routing functions are restricted to the simple case of priority lists ("skipping"). The hardness result also holds when we restrict ourselves to the class of planar graphs.

On the positive side, for scenarios where nodes forward packets independently of the link they received it on (i.e., the routing is "in-port oblivious"), verifying whether a given routing is perfectly resilient, as well as deciding whether a given graph permits perfectly resilient routing, are both decidable in linear time. The in-port oblivious routings are interesting as they allow us to save memory for storing the routing tables.

We additionally contribute several smaller insights into the graph classes in which perfect resilience is possible. For example, we provide the first example of a grid that does not contain K_5 or $K_{3,3}$ as a minor, and does not permit a perfectly resilient routing. It was previously known that K_5 and $K_{3,3}$ are forbidden subgraphs for being perfectly resilient, and planar counterexamples are also known [11].

1.3 Additional Related Work

Local fast re-routing mechanisms have been studied intensively in the literature already, and we refer the reader to the recent survey by Chiesa et al. [2] for a detailed overview. There also exist industrial standards for resilient routing for most modern network protocols, from IP networks [29] to MPLS networks [28] to recent segment routing [1] and software-defined networks [35]. However, these standards typically focus on single link failures and do not provide perfect resiliency guarantees.

There is a large body of applied literature in the networking community on the topic [24, 18, 14, 5, 20, 13], typically focusing on heuristics. In contrast, in our paper, we focus on algorithms which provide formal resilience guarantees.

There are several interesting theoretical results for failure scenarios in which the number f of link failures is bounded. In this context, a local fast rerouting scheme which tolerates f link failures is called *f-resilient*. Chiesa et al. [4] showed that 2-resilient routing is always possible if the graph is 3-link-connected, and Dai et al. [6] showed that 2-resilience is always possible if rerouting rules can also depend on the source of a packet in addition to the target. In their APOCS 2021 paper, Foerster et al. [11] proved that network topologies that form outerplanar graphs are always perfectly resilient and allow for simple and efficient rerouting algorithms based on skipping (each node stores an ordered priority list of alternative links to try per in-port and failed links are then simply skipped). While skipping leads to compact routing tables, it is an open question whether rerouting algorithms which are restricted to skipping come at a price of reduced resilience [3, 11].

Perfectly resilient network configurations, when they exist, can also be generated automatically with recent tools such as SyPer [17] and SyRep [16], using binary decision diagrams with quantification. This also implies that the complexity of the problem of generating such rules is in PSPACE.

In addition to perfect resilience, researchers also study a weaker notion of resilience in the literature called *ideal resilience* [3]. Here it is assumed that initially (before the failures), the network is k-edge-connected, that is, the network cannot be partitioned into isolated components with up to $k-1$ link failures. In such highly connected cases, the routing is called *ideal* if it can tolerate up to $k-1$ link failures. Note that since $k-1$ link failures never

disconnect a k-edge-connected graph, ideal resilience indeed describes a weaker notion of resilience than perfect resilience, and that perfect resilience implies ideal resilience. It is still an open research question whether ideal resilience can always be achieved, but at least it has been shown to be always achievable for all $k \leq 5$ [3]. It is also known that at least $\lfloor \frac{k}{2} - 1 \rfloor$ failures can be tolerated for general k [4]. It has further been proved that $k-1$ failures can be tolerated for special graphs including cliques, complete bipartite graphs, hypercubes, or Clos networks, as well as in scenarios where the rerouting rules can additionally also depend on the source [3]. The corresponding algorithms are based on arborescence decompositions of the underlying graph, where all arborescences are rooted at the target. If a packet traveling along an arborescence encounters a failed link, it is rerouted to the next arborescence in a circular order (known as circular arborescence routing). This technique builds upon Edmond's classic result on edge-disjoint branchings and arborescent decompositions [8], and was used in many papers [2].

Researchers have also already explored various variations for local fast rerouting models. In particular, while we in this paper focus on deterministic algorithms, there are also results on randomized algorithms: models where routers can generate random numbers or hash packet headers. Chiesa et al. in their ICALP 20216 paper considered k-connected graphs and showed that a simple randomized algorithm not only achieves a high robustness but also results in short paths (in the number of hops). Bankhamer et al. in their DISC 2021 paper considered datacenter networks (based on Clos topologies) and showed that as long as the number of failures at each node does not exceed a certain bound, their algorithm achieves an asymptotically minimal congestion up to polyloglog factors along failover paths.

Another model which has achieved attention in the literature allows routers to store and modify (and hence communicate) information in the packet headers. This has been shown to improve resilience [34, 4, 36, 23, 9, 3], but such header manipulation is not always feasible in practice. There is also classic work on models where routers themselves can store dynamic state, for example in the context of link reversal algorithms [15, 25], which can provide a high resilience but which is also less practical [10, 11].

Further upfield, our work is related to distributed computing problems without communication (such as distributed scheduling for disconnected cooperation), as nodes need to decide on the routing without communicating failures [30, 26].

1.4 Organization

The remainder of this paper is organized as follows. We introduce our model and notations more formally in Section 2. In Section 3, we provide examples of graphs which do not support perfectly resilient local routing. Our main contribution, the complexity results, appear in Section 4. We conclude and provide open questions in Section 5.

2 Definitions

For technical convenience, we shall define networks as directed graphs while requiring bi-directional edges. A *directed graph* is a pair $G = (V, E)$ where V is a finite set of nodes and $E \subseteq V \times V$ is a set of directed edges. Let $src(e) = u$ and $tgt(e) = v$ denote the source and target nodes for an edge $e = (u, v) \in E$, and let $out(v) = \{(v, w) \mid (v, w) \in E\}$ and $in(v) = \{(u, v) \mid (u, v) \in E\}$ denote the outgoing and incoming edges to a node $v \in V$. Let $local(v) = out(v) \cup in(v)$ be the set of edges that are local to the node $v \in V$. We shall assume that whenever $(u, v) \in E$ then also $(v, u) \in E$. To simulate the injection of a packet into the network, we moreover require the existence of the self-loop edge $(v, v) \in E$ for each node $v \in V$. Since these loop-back edges always exist, we will not draw them in figures.

A *path* in G is a sequence of edges $e_1e_2 \dots e_n \in E^*$ such that $tgt(e_i) = src(e_{i+1})$ for all i, $1 \leq i < n$. Two nodes $u, v \in V$ are *connected* if there is a path $e_1e_2 \dots e_n$ from u to v such that $src(e_1) = u$ and $tgt(e_n) = v$. In the rest of this paper, we shall consider only connected graphs, meaning that there is a path between any two nodes.

▶ **Definition 1** (Failure Scenario). *A* failure scenario $F \subseteq E$ *is a subset of edges such that* $F \cap \{(v, v) \mid v \in V\} = \emptyset$ *and if* $(u, v) \in F$ *then also* $(v, u) \in F$.

This means that we assume that links are always failing in both directions and by the size of F we shall mean the number of failed bi-directional links. We also assume that the self-loop edges that are used to model the arrival of packets never fail. Edges in F are referred to as *failed* edges and in $E \setminus F$ as *active* edges. Due to our assumption, every node has at least one active edge (self-loop) in any failure scenario.

▶ **Definition 2** (Routing). *A* routing ρ *is a collection of functions* $\rho_F : E \to E$ *for every failure scenario* F *such that* ρ_F *for every edge* $e \in E$ *returns an active next-hop edge* $e' = \rho_F(e)$ *satisfying* $e' \notin F$ *and* $tgt(e) = src(e')$.

From now on, we assume a fixed target node $t \in V$ such that all packets that arrive at any node in the graph should be forwarded to the node t. This corresponds to the (weakest possible) assumption of only destination matching when analyzing packet headers.

▶ **Definition 3** (Perfect resilience). *Let* $G = (V, E)$ *be a directed graph and let* $t \in V$ *be a given target node. A routing* ρ *is* perfectly resilient *if for every failure scenario* F *and every node* $v \in V$ *that is connected to* t *via some path in* $(E \setminus F)^*$*, there is a nonnegative number* $n \geq 0$ *such that* $tgt(\rho_F^n((v, v))) = t$.

In other words, a routing ρ is perfectly resilient if for any failure scenario F and any node v that is connected to t under F, a packet injected via the self-loop to the node v is eventually delivered to the target t.

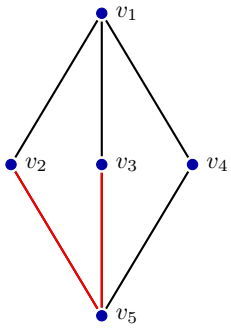
Clearly, if the routing ρ has complete knowledge of the global failure scenario, it is always possible to construct a perfectly resilient routing. As we are interested in routings that allow for fast re-routing based only on the information about locally failing links, we define the notion of a local routing function.

▶ **Definition 4** (Local Routing). *A routing* ρ *is* local *if* $\rho_F(e) = \rho_{F'}(e)$ *for every* $e \in E$ *and any two failure scenarios* F *and* F' *such that* $F \cap local(tgt(e)) = F' \cap local(tgt(e))$.

Hence, the decision of the next-hop can only depend on the knowledge of active and failing links directly connected to a given node. We shall also call such a general local routing function *combinatorial* because the size of a local routing table for a node v requires $2^{|local(v)|}$ routing entries and hence grows exponentially with the degree of v (size of $local(v)$). Storing such routings is impractical due to high memory requirements. In order to save memory in routing tables, several works consider the more practical notion of skipping routing [11, 3, 4].

▶ **Definition 5** (Skipping Routing). *A* skipping priority list *is a function* $\pi : E \to E^*$ *such that for every* $e \in E$ *the function* $\pi(e) = e_1e_2 \dots e_n$ *returns a permutation of all edges in* $out(tgt(e))$. *A given skipping priority list* π *determines a* skipping routing ρ *such that* $\rho_F(e) = e_i$ *where* $\pi(e) = e_1e_2 \dots e_n$ *and* i *is the lowest index such that* $e_i \in E \setminus F$.

▶ **Observation 6.** *Clearly, any skipping routing is also local (combinatorial) because the forwarding decision is based only on the status of edges outgoing from a given node. Moreover, to represent a skipping priority list for a node* v *on a given incoming edge requires us to store only* $|local(v)|$ *outgoing edges.*



(a) Example graph with the target node v_5; all edges are bi-directional.

node	in-port	priority list
v_1	v_1	v_2, v_3, v_4, v_1
v_1	v_2	v_3, v_4, v_2, v_1
v_1	v_3	v_4, v_2, v_3, v_1
v_1	v_4	v_2, v_3, v_4, v_1
v_2	v_1	v_5, v_1, v_2
v_2	v_2	v_5, v_1, v_2
v_3	v_1	v_5, v_1, v_3
v_3	v_3	v_5, v_1, v_3
v_4	v_1	v_5, v_1, v_4
v_4	v_4	v_5, v_1, v_4

(b) Perfectly resilient skipping routing; a routing path with no failing edges when a packet is injected to v_1 is $(v_1, v_1)(v_1, v_2)(v_2, v_5)$.

node	in-port	priority list
v_1	v_1	v_2, v_3, v_4, v_1
v_1	v_2	v_3, v_4, v_2, v_1
v_1	v_3	v_4, v_2, v_3, v_1
v_1	v_4	v_2, v_3, v_4, v_1
v_2	v_1	$\cancel{v_5}$, v_1, v_2
v_2	v_2	$\cancel{v_5}$, v_1, v_2
v_3	v_1	$\cancel{v_5}$, v_1, v_3
v_3	v_3	$\cancel{v_5}$, v_1, v_3
v_4	v_1	v_5, v_1, v_4
v_4	v_4	v_5, v_1, v_4

(c) A routing path for the given failure scenario $F = \{(v_2, v_5), (v_5, v_2), (v_3, v_5), (v_5, v_3)\}$ is $(v_1, v_1)(v_1, v_2)(v_2, v_1)(v_1, v_3)(v_3, v_1)(v_1, v_4)(v_4, v_5)$.

Figure 1 Example of a graph with target v_5 and a perfectly resilient skipping routing.

We refer to Figure 1 for an example of a graph with a perfectly resilient skipping routing. We depict the priority list function $\pi : E \to E^*$ as a routing table where the first column is the node that receives a packet on the given in-port (in the second column) and returns the respective priority list in the third column. For example, the first row in the routing table denotes the routing entry $\pi((v_1, v_1)) = (v_1, v_2)(v_1, v_3)(v_1, v_4)(v_1, v_1)$. The example shows routing paths for a packet injected at the node v_1 in the scenario where no edges fail and when the edges between v_2 and v_5 as well as between v_3 and v_5 fail. As the routing is perfectly resilient, in both scenarios the packet is delivered to the target node v_5.

A more restricted type of routing is the one that does not consider the in-port information for the routing decision. This is formalized in the following definition.

▶ **Definition 7** (In-port Oblivious Routing). *A routing ρ is* in-port oblivious *if $\rho_F(e) = \rho_F(e')$ for any failure scenario F and any two edges $e, e' \in E$ such that $tgt(e) = tgt(e')$.*

Hence, when using in-port oblivious routing, a node v makes always the same next-hop, regardless of which edge (port) the packet arrives to v. In-port oblivious routing is more memory efficient as it requires fewer entries in the routing tables.

Finally, we shall introduce two decision problems related to perfect resilience. In the *verification problem*, we are given a directed graph $G = (V, E)$, a target node $t \in V$ together with a local routing ρ and we have to decide whether ρ is perfectly resilient. In the *synthesis problem*, we ask whether there exists a local (or skipping) routing that is perfectly resilient for a given graph and a target node. As we shall see in the next section, there are graphs that do not have any perfectly resilient local routing.

3 Examples of Graphs with no Perfectly Resilient Local Routing

Foerster et al. [11, 12] examined the resilience of different graphs and graph classes and showed that all outerplanar graphs admit a perfectly resilient skipping routing. Conversely, their study demonstrated that certain non-planar graphs, such as K_5 and $K_{3,3}$ do not show perfect resilience. Due to the minor stability result [11], proving that if a graph is perfectly resilient so are all its minors, they conclude that all non-planar graphs are not perfectly resilient. Furthermore, they observed that among planar graphs, there exist some that are perfectly resilient and others that are not.

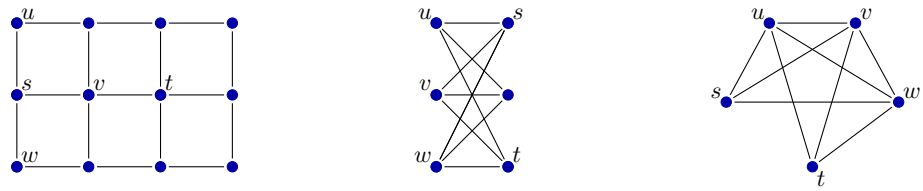


Figure 2 Applications of Theorem 8 on 3×4 grid, $K_{3,3}$ and K_5 without the edge between s and t.

The different examples of non-perfectly resilient graphs published in the literature require specialized proofs for each case. We shall now present a theorem that generalizes these proofs into a single property that prevents the existence of perfectly resilient routings.

▶ **Theorem 8.** *Let $G = (V, E)$ be a directed graph and let $t \in V$ be a target node. If there exist four different nodes $s, u, v, w \in V \setminus \{t\}$ such that*

- *u, v and w are neighbours of s, and*
- *for any permutation of the nodes u, v and w*
  - *there is a path from u to v that does not contain any of the nodes s, w and t, and*
  - *from w we can reach the target node t without visiting any node on the path connecting u to v*

then the graph G has no local (and hence also no skipping) perfectly resilient routing.

Proof. We shall demonstrate that for any local routing on G, we can always construct a failure scenario that causes a forwarding loop. Let ρ be an arbitrary local routing and we shall construct such a failure scenario F for ρ. First of all, we add to F all edges connected to s, except for the edges connecting s with u, v and w. We can assume w.l.o.g. that a packet injected to s is as the first priority forwarded to the node u, i.e., $\rho_F((s, s)) = (s, u)$. This is because we consider an arbitrary permutation of u, v and w in the premise of the theorem.

Next, we assume w.l.o.g. that $\rho_F((v, s)) = (s, u)$. In case that the node s never forwards a packet received on any of the in-ports v and w to the node u, we can create a forwarding loop by failing all edges connected to all nodes on the path from v to w, except for the edges that are on the v-w path and those connecting v and w with s. A packet injected to the node v will now necessarily arrive back to the node s but keep cycling, never reaching the target node t even if the node u is connected to the target node. This is because ρ is a local routing and the node s cannot see that additional edges were failed at other parts of the graph. Hence, such a ρ cannot be perfectly resilient and we can w.l.o.g. assume that $\rho_F((v, s)) = (s, u)$.

We summarize that necessarily $\rho_F((s, s)) = (s, u)$ and $\rho_F((v, s)) = (s, u)$ can be the only alternative for a possible perfectly resilient local routing. However, we can now add to F all edges connected to all nodes on the u–v path, except for the edges that are on this path and those that connect u and v to the node s. Now, by the assumption that ρ is a local routing, a packet injected at the node s will keep cycling on the path $(s, u)u$–$v(v, s)$ and never be delivered to the node t via the path available from s through the node w.

As we exhaustively analyzed all possible local routings ρ and showed that none of them is perfectly resilient, we can conclude that the graph G has no perfectly resilient local routing. ◀

▶ Remark 9. None of the graphs in Figure 2 (3×4 grid, $K_{3,3}$, and K_5 with one removed edge) has a perfectly resilient routing for the depicted target node t. In each case, we can choose the nodes s, u, v, and w as depicted and then apply Theorem 8 as any two nodes

from the set $\{u, v, w\}$ are connected by a path that does not intersect with any of the nodes on the path from the remaining node to the target t. The fact that $K_{3,3}$ does not have any perfectly resilient local routing was previously shown by a dedicated proof in [12].

4 Complexity Results

We shall start by stating an obvious upper-bound on the complexity of perfect resilience verification and at the same time point out to an interesting fact that the synthesis problem is decidable in polynomial time thanks to the well-know Robertson and Seymour [33] theorem.

▶ **Theorem 10.** *Verification of* Perfect Resilience *(the question to decide whether a given local routing function on a given graph is perfectly resilient) is in* coNP*. The problem of* Perfect Resilience *synthesis (the question to decide whether for a given graph there exists some perfectly resilient local routing function) is in* P*.*

Proof. For the verification problem, we can guess a failure scenario and a source node that is connected to the target but where the routing tables create a forwarding loop, which can be for the given failure scenario checked in polynomial time. Hence, if there exists a failure scenario creating a forwarding loop, the given instance of the verification problem is not perfectly resilient, showing that the problem belongs to coNP.

The containment in P for the synthesis problem is a direct consequence of the minor-stability property for this kind of graphs [12] and the seminal result of Robertson and Seymour [33] showing that any such property can be characterized by a finite set of forbidden minors – together with a polynomial time algorithm for checking whether a graph contains a forbidden minor, we know that there must exist an algorithm with a polynomial running time that decides perfect resilience for the case where all nodes can become possible target nodes. Using rooted minors (the target t is a root), it is easy to show that deciding whether a given graph with a fixed given target node is perfectly resilient can be also done in polynomial time [31, 32]. ◀

However, the line of arguments used in the proof of P containment of perfect resilience does not provide a concrete algorithm with a polynomial running time, only points to its existence. The problem of effectively (in polynomial time) designing perfectly resilient routing tables, in case they exist, remains a major open problem.

Next, we shall study the complexity of in-port oblivious perfect resilience, which is an interesting problem from the practical point of view as in-port oblivious routing tables can be stored with reduced memory foot-print.

4.1 Routings with no In-port Matching

We shall first state a necessary and sufficient condition for a graph to allow for in-port oblivious routings.

▶ **Lemma 11.** *Let $G = (V, E)$ be a connected graph. There exists a perfectly resilient in-port oblivious local (or skipping) routing for any given target node if and only if all simple cycles of G have length at most 3.*

Proof. "$\Rightarrow$": For the sake of contradiction, assume that there is an in-port oblivious perfectly resilient local routing ρ for a given target node t and that the graph G contains a simple cycle of length 4 or more. The existence of this cycle implies that there are three different nodes $s, u, v \in V \setminus \{t\}$ such that

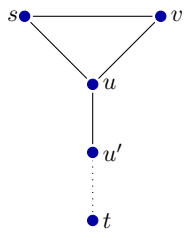


Figure 3 Triangle.

- $(s, u) \in E$ and $(s, v) \in E$,
- there is a path from u to t which does not contain the node v and s, and similarly
- there is a path from v to t which does not contain the node u and s.

Now a packet injected to s in a failure scenario where all outgoing edges from s are failed, except for (s, u) and (s, v), will have to be forwarded to either u or v. Let us w.l.o.g. assume that it is forwarded to u. We extend the failure scenario by failing also all outgoing edges from u, except for the edge between u and s. This means that the packet returns back to s and because the routing is local and cannot see the failed edges at the node u, and we are in-port oblivious, the node s has to forward the packet back to u. A forwarding loop is hence created; however, the node s is still connected to the target t via the node v. This is a contradiction to the assumption that ρ is a perfectly resilient routing.

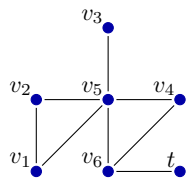
"$\Leftarrow$": Let us assume that the graph G contains only cycles of length at most 3. This implies that from any node in G there is a unique shortest path to the given target node t. We shall construct an in-port oblivious, perfectly resilient skipping (and hence also local) routing ρ as follows. For every node s that is not on any simple cycle of length 3 in G, we define $\rho((*, s)) = (s, u)$ where u is the first node on the unique shortest path from s to t, and where $*$ stands for any neighbour node of s. There is no point in sending the packet along any other outgoing edge as it will have to necessarily return back to s and hence create a forwarding loop. Let us now consider three nodes $s, u, v \in V$ that are on a cycle of length 3 as depicted in Figure 3. Let u be the unique node that has a path to the target t without visiting s or v. We define $\rho((*, u)) = (u, u')$ where u' is the unique next-hop node on the shortest path from u to t. As before, this is the only choice should ρ be perfectly resilient and the edges following after (u, u') in the priority list are irrelevant. Finally, we set $\rho((*, s)) = (s, u)(s, v)$ and $\rho((*, v)) = (v, u)(v, s)$. By case analysis, we can observe that this yields a perfectly resilient routing. ◀

Let us observe that the way we constructed the routing ρ (except for the unimportant routing entries that cannot help us to reach a target) is the only possible and unique choice of priorities should ρ remain perfectly resilient. If we instead, e.g., defined $\rho((*, s)) = (s, v)(s, u)$ then a failure scenario where the edge between v and u fails creates a forwarding loop for a packet injected to the node s.

In Figure 4 we can see a graph which allows for a perfectly resilient skipping routing for the target node t constructed according to the algorithm in the proof of the previous lemma. Only the highlighted unique priority list entries are important.

We can now conclude with the fact that both the verification and synthesis problems for in-port oblivious routings are efficiently solvable.

▶ **Theorem 12.** *Verification and synthesis problems for* Perfect Resilience *using in-port oblivious local and skipping routings are decidable in linear time.*



node	in-port	priority list
v_1	$*$	v_5, v_2, v_1
v_2	$*$	v_5, v_1, v_2
v_3	$*$	v_5, v_3
v_4	$*$	v_6, v_5, v_4
v_5	$*$	v_6, v_4, v_1, v_2, v_3, v_5
v_6	$*$	t, v_4, v_5, v_6

Figure 4 A graph and a perfectly resilient in-port oblivious skipping routing for the target t.

Algorithm 1 Tests the existence of a perfectly resilient routing. Takes a graph (represented as V and E) and a target node t and outputs true if such a routing exists; false otherwise.

```
procedure SYNTHESIS(V, E, t)
    Initialize queue Q ← [t]
    for all v ∈ V do
        visited[v] ← false
        parent[v] ← null
    visited[t] ← true
    while Q ≠ ∅ do
        u ← DEQUEUE(Q)
        cycleFound ← false
        for all (u, v) ∈ E do
            if not visited[v] then
                visited[v] ← true
                parent[v] ← u
                ENQUEUE(Q, v)
            else                                        ▷ (u, v) is an off-tree edge
                if parent[u] ≠ parent[v] then
                    return false
                if cycleFound then                      ▷ u has more than one off-tree edge
                    return false
                cycleFound ← true
    return true
```

Proof. First, by using Lemma 11, we shall check for the existence of cycles of length 4 or more in the connected component containing t. Such a check can be performed in linear time using, e.g., a slightly modified BFS from t where for each node v, we keep track of the parent $p(v)$ and for each link not part of the BFS tree, we check that the two endpoints have the same parent and each vertex is only incident to one such edge. If this linear-time check fails, we output that no perfectly resilient local routing exists. Otherwise, by Lemma 11 there is an in-port oblivious skipping routing ρ and we have a positive instance for the synthesis problem. Algorithm 1 is a pseudocode implementation of this algorithm.

For the verification problem, we just have to check that the important priority list entries for the given skipping or combinatorial routing agree with the choices made by ρ in any local failure scenario. If they agree with the entries computed by the procedure described in Lemma 11 then the given routing is perfectly resilient, otherwise it is not. ◀

4.2 Hardness of Perfect Resilience Verification

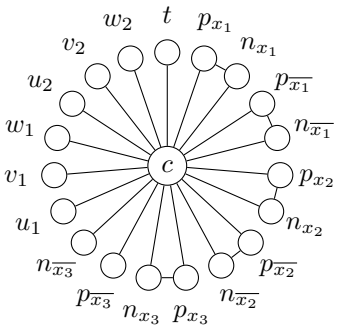
In this section, we show that the verification of perfect resilience problem is coNP-complete. This results contrasts to the fact that the seemingly more difficult problem of perfect resilience synthesis is decidable in polynomial time (see Theorem 10). To show hardness of the verification problem, we reduce from 3-Sat. The main idea for the reduction is to have one central node c, which is the only node that is connected to t, and is furthermore connected to one gadget representing each clause of the given 3-Sat instance. Each failure set is interpreted in each gadget as an assignment of truth values to the variables of the respective clause. We construct a routing function that, starting from v, iterates through the gadgets. Each gadget is connected to v with two links. If the gadget represents a clause that is satisfied, it routes back to v on one link, and if the clause is not satisfied, it routes back on the other. In that way, the routing function at v knows whether a clause that it just iterated through is satisfied or not. If a clause that is not satisfied is encountered, v immediately routes to t. Otherwise, v routes to the next gadget. In case all clauses are satisfied, the routing function will cycle through all gadgets indefinitely and never route to t. Therefore, a failure set that prevents a packet inserted at v from reaching t exists exactly if the 3-Sat instance is satisfiable.

▶ **Theorem 13.** *Verification for Perfect Resilience with skipping priorities is coNP-complete even if the input graph is planar.*

Proof. Containment in coNP is proved in Theorem 10. To show coNP-hardness, we present a reduction from 3-Sat, a variant of Satisfiability where each clause contains exactly 3 literals. This problem is among Karp's original 21 NP-complete problems [21]. We show a polynomial-time reduction from 3-Sat to the verification problem of Perfect Resilience where the constructed skipping routing is perfectly resilient if and only if the original instance of 3-Sat (a formula ϕ) is *not* satisfiable.

Let $\mathcal{V} = \{x_1, x_2, \ldots, x_n\}$ and $\mathcal{C} = \{C_1, C_2, \ldots, C_m\}$ be the set of variables and clauses of an input formula ϕ. We construct an instance of the verification problem for Perfect Resilience with skipping priorities. We start with two nodes c and t, which are connected by a link and where t is the target. Next, for each variable $x_i \in \mathcal{V}$, we add four nodes $p_{x_i}, n_{x_i}, p_{\overline{x_i}}$, and $n_{\overline{x_i}}$ and the six links $(c, p_{x_i}), (c, n_{x_i}), (p_{x_i}, n_{x_i}), (c, p_{\overline{x_i}}), (c, n_{\overline{x_i}})$, and $(p_{\overline{x_i}}, n_{\overline{x_i}})$ together with the respective link in the other direction. For each clause $C_j \in \mathcal{C}$, we add three nodes u_j, v_j, w_j and three links $(c, u_j), (c, v_j), (c, w_j)$ and the links in the other direction. See Figure 5a for an example of the construction and note that the graph is planar and the number of nodes and links is linear in the number of variables and clauses in ϕ.

We continue with the priority lists. Note that if the link (c, t) fails, then only packets starting in t can reach t. Hence, we assume in the remainder of the proof that this link does not fail. Hence, we only need to state the priority lists for c (for different in-ports) up to t. The priority lists are given in Figure 5 (right). Before we formally state the proof, let us give some high-level intuition. The goal is to find a set of failing links that cause the packet to loop forever between c and variable nodes ($p_{x_i}, n_{x_i}, p_{\overline{x_i}}$ and $n_{\overline{x_i}}$) and clause nodes (u_j, v_j, and w_j). First, if the center node c ever receives the packet from a node p_{x_i} or $p_{\overline{x_i}}$ for some x_i, then it immediately forwards the packet to t. We next show for each i that exactly one of the links (c, n_{x_i}) and $(c, n_{\overline{x_i}})$ fails in any solution. Assume that c receives the packet from $n_{x_{i-1}}$ or $n_{\overline{x_{i-1}}}$ (or from u_m, v_m, w_m or it starts in c for $i = 1$). Then not both the links (c, p_{x_i}) and $(c, p_{\overline{x_i}})$ can fail as otherwise the packet is sent to t. Assume the packet is sent to $p_{\overline{x_i}}$ (the other case is symmetric). If either of the links $(p_{\overline{x_i}}, n_{\overline{x_i}})$ or $(n_{\overline{x_i}}, c)$ fails, then the packet is sent back from $p_{\overline{x_i}}$ to c and then sent to t. Hence, neither of these



(a) Construction of a graph for a formula ϕ with three variables and two clauses; all edges are bi-directional.

node	in-port	priority list
p_{x_i}	c	n_{x_i}, c
p_{x_i}	n_{x_i}, p_{x_i}	c, n_{x_i}
n_{x_i}	c	p_{x_i}, c
n_{x_i}	p_{x_i}, n_{x_i}	c, p_{x_i}
$p_{\overline{x_i}}$	c	$n_{\overline{x_i}}, c$
$p_{\overline{x_i}}$	$n_{\overline{x_i}}, p_{\overline{x_i}}$	$c, n_{\overline{x_i}}$
$n_{\overline{x_i}}$	c	$p_{\overline{x_i}}, c$
$n_{\overline{x_i}}$	$p_{\overline{x_i}}, n_{\overline{x_i}}$	$c, p_{\overline{x_i}}$
u_j	$*$	c
v_j	$*$	c
w_j	$*$	c
c	c	$p_{x_1}, p_{\overline{x_1}}, t$
c	$p_{x_i}, p_{\overline{x_i}}$	t
c	n_{x_i}	$n_{\overline{x_i}}, p_{x_{i+1}}, p_{\overline{x_{i+1}}}, t$
c	$n_{\overline{x_i}}$	$n_{x_i}, p_{x_{i+1}}, p_{\overline{x_{i+1}}}, t$
c	n_{x_n}	$n_{\overline{x_n}}, u_1, v_1, w_1, t$
c	$n_{\overline{x_n}}$	$n_{x_n}, u_1, v_1, w_1, t$
c	u_m	$n_{q_m}, p_{x_1}, p_{\overline{x_1}}, t$
c	v_m	$n_{r_m}, p_{x_1}, p_{\overline{x_1}}, t$
c	w_m	$n_{s_m}, p_{x_1}, p_{\overline{x_1}}, t$
c	u_j	$n_{q_j}, u_{j+1}, v_{j+1}, w_{j+1}, t$
c	v_j	$n_{r_j}, u_{j+1}, v_{j+1}, w_{j+1}, t$
c	w_j	$n_{s_j}, u_{j+1}, v_{j+1}, w_{j+1}, t$

Figure 5 Table of the priority lists for all nodes (except for the target t). The symbol $*$ stands for the case that the priority list does not depend on the in-port. We use q_j, r_j, and s_j to denote the three literals in clause C_j, respectively. As an example, if clause $C_0 = (x \vee y \vee \overline{z})$, then $q_0 = x$, $r_0 = y$, and $s_0 = \overline{z}$. Priority lists for node c are only shown up to the target t.

links fail and the packet is forwarded over these links. Now assume the link (c, n_{x_i}) does not fail. Then, the packet is sent through that link. Now at least one of the links (n_{x_i}, p_{x_i}) and (p_{x_i}, c) has to fail or c receives the packet from p_{x_i}. Hence, the packet is sent back to c from n_{x_i}. However, by construction the packet is now sent to $n_{\overline{x_i}}$ and consequently to $p_{\overline{x_i}}$, back to c, and then to t (and since we assume that if a link fails, then also the link in the other direction fails, it holds that none of these links fail as we assumed or showed before). Thus, exactly one of the links (c, n_{x_i}) and $(c, n_{\overline{x_i}})$ fails in any solution and whichever does not fail, the entire respective loop also does not fail. We say that if the link (c, n_{x_i}) fails, then x_i is set to true and if the link $(c, n_{\overline{x_i}})$ fails, then x_i is set to false.

We now show that the reduction is correct, that is, ϕ is satisfiable if and only if the constructed routing is not perfectly resilient. In the forward direction, assume that a satisfying assignment β for ϕ exists. We let the links (c, p_{x_i}) and (c, n_{x_i}) fail for each variable x_i set to true by β and we let the links $(c, p_{\overline{x_i}})$ and $(c, n_{\overline{x_i}})$ fail whenever x_i is set to false by β. Moreover, for each clause $C_j = (q_j \vee r_j \vee s_j)$, we pick one variable that satisfies the clause under β. If q_j is picked, then we let links (c, v_j) and (c, w_j) fail. If r_j is picked, then we let links (c, u_j) and (c, w_j) fail. If s_j is picked, then we let links (c, u_j) and (c, v_j) fail. Now, the packet goes through p_{x_i} and n_{x_i} or $p_{\overline{x_i}}$ and $n_{\overline{x_i}}$ for all i in increasing order. Then, for each clause C_j in increasing order, it goes to the previously chosen node (u_j, v_j, or w_j) and immediately returns to c. Then, it goes to the next clause as we have a satisfying assignment and hence the link to $n_{q_j}/n_{r_j}/n_{s_j}$ fails. Afterwards it again goes through all variables and this cycle continues indefinitely. Thus, the constructed routing is not perfectly resilient.

In the other direction, assume that the constructed routing is not perfectly resilient and let F be a failure scenario where the packet does not reach t from some source node. We have already shown that exactly one of the links (c, n_{x_i}) and $(c, n_{\overline{x_i}})$ fails for each variable x_i and how this corresponds to an assignment. If the packet reaches some clause node u_j, v_j, or w_j (or starts there and there exists some path to c and t), then it is sent to c. Then, c tries to send the packet to a node n_{x_i} or $n_{\overline{x_i}}$ for some x_i. If this link does not fail (which corresponds to the case where the clause C_j is not satisfied by the respective assignment of the corresponding variable), then the packet is sent to the respective node n_y and subsequently to p_y and back to c from p_y as the entire loop is not failing whenever the link (c, n_y) is not failing (as shown above). If all three links to a clause gadget fail, then the packet is directly sent to t, so we may assume that at least one of them is not failing. The first one that is not failing will encode the satisfying assignment for the respective clause. The packet is then sent to one of the three nodes for the next clause and the cycle continues. This cycle can only continue indefinitely, if all clauses are satisfied by the chosen assignment, that is, the formula ϕ is satisfiable. This concludes the proof. We note that the hardness result holds even in case that we know the source of the packet as, e.g., injecting the packet at c is already enough to argue for coNP hardness. ◀

Note that the size of the network constructed in the previous proof is linear in the number of variables and clauses of the input formula. Assuming a complexity hypothesis called the exponential time hypothesis (ETH), 3-Sat cannot be solved in $2^{o(n+m)}$ time [19], where n and m are the number of variables and clauses in the input formula. Hence, both verification problems cannot be solved in $2^{o(n+m)}$ time. This contrasts with the typical results that many problems on planar graphs can be solved in subexponential time (usually $2^{\sqrt{n}}$ time) [7, 22, 27]. However, as we just proved, this is surprisingly not the case for the perfect resilience verification problem (assuming $\mathsf{P} \neq \mathsf{NP}$).

5 Conclusion and Open Questions

Fast re-route mechanisms that allow for a local and immediate reaction to link failures are essential in present-day dependable computer networks. Several approaches for fast re-routing have been implemented and deployed in essentially every type of computer network. However, the question for which graphs we can efficiently construct perfectly resilient protection mechanisms, meaning that they should guarantee packet delivery in any failure scenario as long as the source and target nodes remain physically connected, has not been solved yet. The exact computational complexity of deciding whether a given network topology together with a target node allows for a perfectly resilient routing protection is showed to be polynomial by the application of Robertson and Seymour theorem [33] together with the minor-stability result [12] for the existence of a perfectly resilient routing to all possible target nodes as well as to a given target node (via the rooted variant of this result [31, 32]). This contrasts to our result that the verification problem is computationally hard – we showed by a nontrivial reduction from 3-Sat that the question whether a given set of routing tables for a fixed target node is perfectly resilient is coNP-complete.

As a sufficient condition that no perfectly resilient routings exist, we provided a general theorem that allows us to efficiently identify network topologies where such a routing is not possible. We have also studied a variant of a local fast re-route that is in-port oblivious (i.e., the packet's incoming interface is not part of the routing tables). This potentially enables a faster failover protection as well as a more memory-efficient way of storing the forwarding tables. In this restricted case, we were able to provide a fast linear-time algorithm that

determines whether a given in-port oblivious routing is perfectly resilient and we showed that the synthesis problem is also decidable in linear time, allowing us to efficiently construct in-port oblivious routing tables for the topologies where this is indeed possible.

A major open problem is to design a concrete algorithm that for a given network constructs in polynomial time perfectly resilient routing tables whenever this is possible.

References

1. Ahmed Bashandy, Clarence Filsfils, B Decraene, S Litkowski, P Francois, D Voyer, F Clad, and P Camarillo. Topology independent fast reroute using segment routing. *Working Draft*, 2018.
2. Marco Chiesa, Andrzej Kamisiński, Jacek Rak, Gábor Rétvári, and Stefan Schmid. A survey of fast-recovery mechanisms in packet-switched networks. *IEEE Communications Surveys & Tutorials*, 23(2):1253–1301, 2021. doi:10.1109/COMST.2021.3063980.
3. Marco Chiesa, Ilya Nikolaevskiy, Slobodan Mitrovic, Andrei V. Gurtov, Aleksander Madry, Michael Schapira, and Scott Shenker. On the resiliency of static forwarding tables. *IEEE/ACM Transactions on Networking*, 25(2):1133–1146, 2017. doi:10.1109/TNET.2016.2619398.
4. Marco Chiesa, Ilya Nikolaevskiy, Slobodan Mitrovic, Aurojit Panda, Andrei V. Gurtov, Aleksander Madry, Michael Schapira, and Scott Shenker. The quest for resilient (static) forwarding tables. In *Proceedings of the 35th Annual IEEE International Conference on Computer Communications (INFOCOM)*, pages 1–9. IEEE, 2016. doi:10.1109/INFOCOM.2016.7524552.
5. Marco Chiesa, Roshan Sedar, Gianni Antichi, Michael Borokhovich, Andrzej Kamisinski, Georgios Nikolaidis, and Stefan Schmid. PURR: a primitive for reconfigurable fast reroute: hope for the best and program for the worst. In *CoNEXT'19*, pages 1–14. ACM, 2019. doi:10.1145/3359989.3365410.
6. Wenkai Dai, Klaus-Tycho Foerster, and Stefan Schmid. A tight characterization of fast failover routing: Resiliency to two link failures is possible. In *Proc. ACM SPAA*, pages 153–163, 2023. doi:10.1145/3558481.3591080.
7. Erik D. Demaine and MohammadTaghi Hajiaghayi. The bidimensionality theory and its algorithmic applications. *The Computer Journal*, 51(3):292–302, 2008. doi:10.1093/COMJNL/BXM033.
8. J. Edmonds. Edge-disjoint branchings. *Combinatorial Algorithms*, pages 91–96, 1973. URL: https://cir.nii.ac.jp/crid/1573387451139139584.
9. Theodore Elhourani, Abishek Gopalan, and Srinivasan Ramasubramanian. IP fast rerouting for multi-link failures. *IEEE/ACM Transactions on Networking*, 24(5):3014–3025, 2016. doi:10.1109/TNET.2016.2516442.
10. Joan Feigenbaum, Brighten Godfrey, Aurojit Panda, Michael Schapira, Scott Shenker, and Ankit Singla. Brief announcement: on the resilience of routing tables. In *PODC*, pages 237–238. ACM, 2012. doi:10.1145/2332432.2332478.
11. Klaus-Tycho Foerster, Juho Hirvonen, Yvonne-Anne Pignolet, Stefan Schmid, and Gilles Trédan. On the feasibility of perfect resilience with local fast failover. In *Proceedings of the 2nd SIAM Symposium on Algorithmic Principles of Computer Systems (APOCS)*, pages 55–69. SIAM, 2021. doi:10.1137/1.9781611976489.5.
12. Klaus-Tycho Foerster, Juho Hirvonen, Yvonne-Anne Pignolet, Stefan Schmid, and Gilles Trédan. On the price of locality in static fast rerouting. In *52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN*, pages 215–226. IEEE, 2022. doi:10.1109/DSN53405.2022.00032.
13. Klaus-Tycho Foerster, Andrzej Kamisinski, Yvonne-Anne Pignolet, Stefan Schmid, and Gilles Trédan. Grafting arborescences for extra resilience of fast rerouting schemes. In *INFOCOM*, pages 1–10. IEEE, 2021. doi:10.1109/INFOCOM42981.2021.9488782.

14 Klaus-Tycho Foerster, Mahmoud Parham, Marco Chiesa, and Stefan Schmid. TI-MFA: Keep calm and reroute segments fast. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 415–420. IEEE, 2018. doi:10.1109/INFCOMW.2018.8406885.

15 Eli Gafni and Dimitri P. Bertsekas. Distributed algorithms for generating loop-free routes in networks with frequently changing topology. *IEEE Transactions on Communications*, 29(1):11–18, 1981. doi:10.1109/TCOM.1981.1094876.

16 Csaba Györgyi, Kim G. Larsen, Stefan Schmid, and Jirí Srba. Syrep: Efficient synthesis and repair of fast re-route forwarding tables for resilient networks. In *Proceedings of the 54th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 483–494. IEEE, 2024. doi:10.1109/DSN58291.2024.00053.

17 Csaba Györgyi, Kim G. Larsen, Stefan Schmid, and Jiří Srba. Syper: Synthesis of perfectly resilient local fast re-routing rules for highly dependable networks. In *IEEE INFOCOM 2024 - IEEE Conference on Computer Communications*, pages 2398–2407, 2024. doi:10.1109/INFOCOM52122.2024.10621323.

18 Thomas Holterbach, Stefano Vissicchio, Alberto Dainotti, and Laurent Vanbever. Swift: Predictive fast reroute. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 460–473, 2017. doi:10.1145/3098822.3098856.

19 R. Impagliazzo, R. Paturi, and F. Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4):512–530, 2001. doi:10.1006/JCSS.2001.1774.

20 Peter Gjøl Jensen, Dan Kristiansen, Stefan Schmid, Morten Konggaard Schou, Bernhard Clemens Schrenk, and Jiří Srba. AalWiNes: A fast and quantitative what-if analysis tool for mpls networks. In *ACM CoNEXT*, pages 474–481, 2020.

21 Richard M. Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Plenum Press, 1972. doi:10.1007/978-1-4684-2001-2_9.

22 Philip N. Klein and Dániel Marx. A subexponential parameterized algorithm for subset TSP on planar graphs. In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1812–1830. SIAM, 2014. doi:10.1137/1.9781611973402.131.

23 Karthik Lakshminarayanan, Matthew Caesar, Murali Rangan, Tom Anderson, Scott Shenker, and Ion Stoica. Achieving convergence-free routing using failure-carrying packets. *ACM SIGCOMM*, 37(4):241–252, 2007. doi:10.1145/1282380.1282408.

24 Junda Liu, Aurojit Panda, Ankit Singla, Brighten Godfrey, Michael Schapira, and Scott Shenker. Ensuring connectivity via data plane mechanisms. In *Proceedings of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 113–126. USENIX Association, 2013. URL: https://www.usenix.org/conference/nsdi13/technical-sessions/presentation/liu_junda.

25 Junda Liu, Baohua Yang, Scott Shenker, and Michael Schapira. Data-driven network connectivity. In *Proceedings of the 10th ACM Workshop on Hot Topics in Networks (HotNets)*, page 8. ACM, 2011. doi:10.1145/2070562.2070570.

26 Grzegorz Malewicz, Alexander Russell, and Alexander A Shvartsman. Distributed scheduling for disconnected cooperation. *Distributed Computing*, 18:409–420, 2006. doi:10.1007/S00446-005-0149-0.

27 Jesper Nederlof. Detecting and counting small patterns in planar graphs in subexponential parameterized time. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 1293–1306. ACM, 2020. doi:10.1145/3357713.3384261.

28 Ping Pan, George Swallow, and Alia Atlas. Fast reroute extensions to RSVP-TE for LSP tunnels. *RFC*, 4090:1–38, 2005. doi:10.17487/RFC4090.

29 J. Papán, P. Segeč, M. Moravčík, M. Kontšek, L. Mikuš, and J. Uramová. Overview of ip fast reroute solutions. In *ICETA*, pages 417–424, 2018.

30 Yvonne-Anne Pignolet, Stefan Schmid, and Gilles Tredan. Load-optimal local fast rerouting for resilient networks. In *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 345–356. IEEE, 2017.

31 Neil Robertson and Paul D. Seymour. Graph minors IV. Tree-width and well-quasi-ordering. *Journal of Combinatorial Theory, Series B*, 48(2):227–254, 1990. doi:10.1016/0095-8956(90)90120-0.

32 Neil Robertson and Paul D. Seymour. Graph minors XIII. The disjoint paths problem. *Journal of Combinatorial Theory, Series B*, 63(1):65–110, 1995. doi:10.1006/JCTB.1995.1006.

33 Neil Robertson and P.D Seymour. Graph minors. VIII. A Kuratowski theorem for general surfaces. *Journal of Combinatorial Theory, Series B*, (48(2)):255–288, 1990.

34 Brent Stephens, Alan L Cox, and Scott Rixner. Plinko: Building provably resilient forwarding tables. In *HotTopics in Networks*, pages 1–7, 2013. doi:10.1145/2535771.2535774.

35 Switch Specification 1.3.1. OpenFlow. In *https://bit.ly/2VjOOTT*, 2013.

36 Baohua Yang, Junda Liu, Scott Shenker, Jun Li, and Kai Zheng. Keep forwarding: Towards k-link failure resilient routing. In *IEEE INFOCOM 2014 - IEEE Conference on Computer Communications*, pages 1617–1625, 2014. doi:10.1109/INFOCOM.2014.6848098.