# A Tight Characterization of Fast Failover Routing: Resiliency to Two Link Failures is Possible

Wenkai Dai
Faculty of Computer Science and UniVie Doctoral School Computer Science DoCS, University of Vienna

Klaus-Tycho Foerster
TU Dortmund

Stefan Schmid
TU Berlin and University of Vienna

## ABSTRACT

To achieve fast recovery from link failures, most modern communication networks feature local fast failover mechanisms in the data plane. These failover mechanisms typically rely on pre-installed static rerouting rules which can depend only on local failure information. The locally limited failure information renders the problem of providing a high resilience algorithmically challenging. In this paper, we are interested in algorithms which tolerate a maximal number of $k$ simultaneous link failures to guarantee packet delivery, as long as source and destination remain connected afterwards. Prior work showed that $k$=1 link failure can always be tolerated in general networks, but already the question of $k$=2 remained an unresolved problem.

This paper closes this gap by presenting a tight characterization of fast failover routing: We show that 2-resiliency is possible with pre-installed static routing rules on general topologies, but that 3-resiliency is impossible already in relatively simple networks. Our 2-resilient routing scheme can be computed efficiently and relies on a careful kernelization and subdivision of the network topology into sparse subgraphs.

## CCS CONCEPTS

• **Theory of computation** → **Routing and network design problems**; • **Networks** → Network reliability.

## KEYWORDS

Network resilience, local failover, routing, fast reroute, date plane

**ACM Reference Format:**
Wenkai Dai, Klaus-Tycho Foerster, and Stefan Schmid. 2023. A Tight Characterization of Fast Failover Routing: Resiliency to Two Link Failures is Possible. In *Proceedings of the 35th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA '23), June 17–19, 2023, Orlando, FL, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3558481.3591080

## 1 INTRODUCTION

Communication networks are a critical infrastructure of our digital society. To meet their stringent dependability requirements, networks need to be able to quickly deal with link failures, which are unavoidable and likely to become more frequent with the increasing scale of networks [21]. It has been shown that even short disruptions of connectivity can cause severe degradation in service quality [1, 36, 38].

Traditionally, when confronting a failure, routing protocols such as OSPF [30] and IS-IS [23] are invoked to recompute routing tables. Such reactions to link failures in the control plane are usually unacceptably slow for critical applications [1, 20, 21, 36, 38]. For a more rapid response to failures, most modern communication networks additionally feature local fast failover techniques in the data plane, where, upon a link failure, packets are *locally* rerouted to pre-installed alternative paths without waiting for global route re-computation. This can be orders of magnitude faster [13, 29]. For example, many networks rely on *IP Fast Reroute* [24, 25, 32] or *MPLS Fast Reroute* [31] to deal with failures on the data plane, SDNs provide FRR functionality in terms of *OpenFlow fast-failover groups* [34], BGP relies on BGP-PIC [9] for quickly rerouting flows, to name a few.

Fast re-routing of flows in the data plane however introduces an algorithmic challenge: How to pre-define the static failover rules such that reachability on the routing level is preserved, even under failures? In particular, routing rules can only depend on local failure information, and not on possible further failures downstream.

*The fundamental question is*: Can we design a *$k$-resilient failover routing*, which tolerates any $k$ simultaneous link failures, as long as the underlying topology remains connected?

The question of how to design robust failover mechanisms has received much attention over the last years [6]. While randomization [5, 8] (e.g., with a random walk) can always overcome failures in a graph as long as it is connected, similar to graph exploration, standard routers and switches do not support efficient randomized forwarding [19, 28]; randomization usually also results in long routes and may lead to packet reorderings. Similar disadvantages hold for packet duplication algorithms, such as, e.g., flooding, which also impose heavy additional load on the network.

Regarding deterministic algorithms, Feigenbaum et al. [13] gave a DAG-based construction s.t. 1-resilient failover routing is always possible, however, resilience to arbitrarily many link failures is already impossible on graphs with eight nodes [14]. While Chiesa et al. [7] have shown that 2-resilience is impossible when disregarding source information, standard IP-headers contain both the source and destination address, raising the question of how much more resilience fast failover routing can provide. Most further existing solutions either only provide heuristic guarantees, rely on packet header modifications, or require densely connected networks [5, 6, 18, 37]. Hence, even the question of 2- or 3-resilience is unresolved so far, except for weaker routing models [7] or dense connectivity

**Table 1: Summary of related previous and new results**

| **Resiliency results for static & deterministic routing without packet header modification** | **Assuming graph $k$-edge-connected** | **Local information matched for routing at each node** *Per-source* | *Per-destination* | *Per-incoming Port* | *Incident Links* |
|---|---|---|---|---|---|
| 1-resilience impossible [27] | no | | × | | × |
| 1-resilience possible [13], 2-resilience impossible [7] | no | | × | × | × |
| $(k-1)$-resilience possible for $k \leq 5$ [7] | yes | | × | × | × |
| $(\lfloor k/2 \rfloor)$-resilience possible for $k \geq 6$ [7] | yes | | × | × | × |
| $(k-1)$-resilience [19] | yes | × | × | × | × |
| arbitrary $k$-resilience impossible [18] | no | × | × | × | × |
| 2-resilience possible, 3-resilience imposs. [this paper] | no | × | × | × | × |

assumptions [7, 19], e.g., $k$-edge-connectivity against $k-1$ link failures. However, many real-world communication networks have relatively low all-to-all connectivity, with some dense parts [11, 15], and hence fast failover routing algorithms without connectivity assumptions are desirable.

Hence, in this paper, we are interested in approaches giving provable and deterministic worst-case resilience guarantees on general topologies, which do not require packet header rewriting or convergence mechanisms. In this sense, our work is most closely related to Feigenbaum et al. [13], Chiesa et al. [7], and Foerster et al. [16–19, 33], which consider deterministic algorithms for static failover routing. We give an overview of the directly related results in Table 1. A main conceptual difference in our work to the above is the choice and computation of the underlying topological routing structure. Previous algorithms which focused on special graph classes could leverage well-understood graph structures, namely the classic right-hand rule on outerplanar graphs for face routing [3, 26], disjoint paths or trees [2], respectively routing on a directed acyclic graph created by, e.g., depth-first search. We on the other hand need to devise a novel subgraph structure in §4-§6 that retains topological resilience, but also allows for efficient local failover routing.

*Contributions.* This paper presents a tight characterization of $k$-resilient local failover routing, by showing that on the one hand, a 3-resilient or better failover routing is impossible in general, but that on the other hand, a 2-resilient failover routing scheme always exists and is in fact, polynomial-time computable on every network topology. The main technical challenge is to find a representative topological structure that is sufficient to retain 2-resilience. To this end, we show that the problem can be reduced to subgraphs with virtual sources $s$ and destinations $t$ connected by two node-disjoint paths, in which we can derive further component subdivisions that preserve the same level of connectivity between $s$ and $t$. These subdivisions can be combined into one subgraph of the original topology, for which we prove that the same level of connectivity against two link failures is retained, resulting in an algorithm for 2-resilient failover routing problem.

*Organization.* We present a formal model in §2 and prove the impossibility of 3-resilience in §3. We then present intuitions, challenges, and reductions for solving the 2-resilient failover routing problem on general graphs in §4, followed by showing how these reduced components can be computed and present their connectivity properties in §5. The proof that failure resilience is maintained in the resulting subgraph is given in §6, along with a corresponding 2-resilient failover routing algorithm. We conclude in §7 with some open questions.

## 2 PRELIMINARIES

We represent the given network as *an undirected (multigraph)* $G = (V, E)$, where each router in the network is modeled by a node in $V$ and each bi-directed link between two routers $u$ and $v$ is modeled by an undirected edge $\{u, v\} \in E$. Henceforth, we assume that *multi-edges* between any two nodes in $V$ are *distinguishable.* Let $F \subseteq E$ denote failed links in $G$, which fail in transferring packets in both directions. For a set of edges $E' \subset E$, let $G \setminus E'$ denote a subgraph of $G$ obtained by removing edges $E'$ in $G$. For a set of nodes $V' \subset V$, let $G \setminus V'$ be a subgraph of $G$ generated by removing $V'$ and all incident edges of $V'$ in $G$. For a graph $G' \subseteq G$ and a node $v \in V(G')$, we use $N_{G'}(v)$, $E_{G'}(v)$, $\Delta_{G'}(v)$ to denote the *(open) neighbors* (excluding $v$), *incident edges*, and *the degree* of $v$ respectively in $G'$, where $G'$ can be omitted when the context is clear.

**Failover Routing:** For failover routing, each $v \in V$ stores a *pre-defined and static forwarding (routing) function*, which makes a forwarding decision deterministically for each incoming packet only relying on the local information at $v$, i.e.,

- the source $s$ and the destination $t$ of the incoming packet,
- the incoming port (*in-port*) of the packet at node $v$,
- and the set of non-failed links incident on $v$.

More specifically, given a graph $G$, a source $s \in V$, and a destination $t \in V$, a *forwarding function* for a *source-destination pair* $(s, t)$ at a node $v \in V$ is defined as $\pi^{(s,t)}_{G,v} : N_G(v) \cup \{\perp\} \times 2^{E_G(v)} \mapsto E_G(v)$, where $\perp$ represents the empty in-port, i.e. sending a packet originated at $v$ (for multigraphs, the function can be extended appropriately). When $G$ and the source-destination pair $(s, t)$ are clear, $\pi^{(s,t)}_{G,v}\left(u, E_{G\setminus F}(v)\right)$ can be abbreviated as $\pi_v\left(u, E_{G\setminus F}(v)\right)$, where $u \in N_{G\setminus F}(v) \cup \{\perp\}$. Let $F_v$ denote the failed edges incident on a node $v \in V$. With a slight abuse of notation, the forwarding function $\pi^{(s,t)}_{G,v}\left(u, E_{G\setminus F}(v)\right)$ can be also denoted by the form $\pi^{(s,t)}_{G,v}(u, F_v)$ since $E_{G\setminus F}(v) = E_G(v) \setminus F_v$. Especially, when $v$ does not lose any link under $F$, i.e., $E_{G\setminus F}(v) = E_G(v)$, its forwarding function is simplified as $\pi_v(u)$. The collection of forwarding functions: $\Pi^{(s,t)} = \bigcup_{v \in V\setminus\{t\}} \left(\pi^{(s,t)}_v\right)$ is called a *forwarding scheme for* $(s, t)$.

*Definition 2.1 (k-Resilient Failover Routing Problem).* Given a graph $G = (V, E)$, the *$k$-resilient failover routing problem* is to compute a *$k$-resilient* forwarding scheme for each source-destination pair $(s, t)$ in $G$. A forwarding scheme for $(s, t)$ is called *$k$-resilient*, if this scheme can route a packet originated at $s \in V$ to its destination $t \in V$ as long as the set of failed links $F \subseteq E$ is of cardinality at most $k$, i.e., $|F| \leq k$, and $s - t$ remains connected in $G \setminus F$.

We will focus on computing $k$-resilient forwarding scheme for a given pair $(s, t)$, as our algorithm for $(s, t)$ can be applied to any two nodes $u, v \in V$.

**Dead-Ends, Loops, and Circular Routing:** Next, we introduce some commonly-used concepts of forwarding function in failover routing. We say that a node $v$ *bounces back* a packet $p$, if $v$ sends the incoming packet $p$ back via its incoming port. Given a graph $G' \subseteq G$, if a node $v \in V(G')$ has only one neighbor, i.e., $\Delta_{G'}(v) = 1$, it is called a *dead-end*, any forwarding function at $v$ must bounce back packets received from its unique neighbor, otherwise packets must be *stuck* after arriving at $v$. A *forwarding loop* arises when the same direction of an undirected link is repeated in a path of a packet. Both directions of an undirected link can be traversed once without generating a loop. A packet cannot be sent from $u$ to $v$ anymore, if its path contains a forwarding loop or if the packet is stuck at a node. A forwarding function at a node $v \in V$ is called *link-circular* if packets routed by $v$ are based on an *ordered circular sequence* $\langle u_1, \dots, u_l \rangle$ of the *open neighbors* $\{u_1, \dots, u_l\}$ of $v$ as follows: a packet $p$ received from a node $u_i$ is forwarded to $u_{i+1}$; if the link $\{v, u_{i+1}\}$ is failed, then $p$ is forwarded to $u_{i+2}$ and so on, with $u_1$ following $u_l$ [7]. Obviously, for circular routing, bouncing back is only allowed on dead-ends.

**Further Notations and Graph Theory Concepts:** In the following, we also state some related concepts in graph theory and notations that we will use. A path $P$ from $u \in V$ to $v \in V$ in $G$ is called an $u - v$ path in $G$. Two paths are *edge-disjoint* if they do not have any joint edge, but they may have common (joint) nodes. Two edge-disjoint $u - v$ paths $P_1$ and $P_2$ are *node-disjoint* if $V(P_1) \cap V(P_2) \setminus \{u, v\} = \emptyset$. In this paper, we are often interested in the *edge-connectivity*, henceforth simply denoted by *connectivity*. In a graph $G = (V, E)$, two nodes $u \in V$ and $v \in V$ are *$k$-connected* (interchangeably, $u - v$ is *$k$-connected*) if there exist $k$ *edge-disjoint $u - v$ paths* in $G$, and $G$ is called *$k$-connected* if any two nodes in $V$ are *$k$-connected*. Particularly, $u$ and $v$ are called *exactly $k$-connected* if $u - v$ is $k$-connected but not $(k + 1)$-connected. An *$u - v$ articulation point* of a graph $G = (V, E)$, where $u, v \in V$, is a node $w \in V \setminus \{u, v\}$ whose removal breaks all $u$–$v$ paths of $G$. Given a path $P = (x_0, x_1, \dots, x_k)$, where $x_i \in V(P)$ for $0 \leq i \leq k$, then $x_i P x_j \subseteq P$ denotes a sub-path of $P$ from $x_i$ to $x_j$, where $0 \leq i < j \leq k$. Let $P = (x_0, \dots, u, \dots, x_k)$ and $Q = (y_0, \dots, u, \dots, y_\ell)$ be two paths with a joint node $u$ , where $x_{i'} = u$ and $y_{j'} = u$, then $x_i P u Q y_j$, where $0 \leq i < i'$ and $j' < j \leq \ell$, denotes a path from $x_i$ to $y_j$ by joining sub-paths $x_i P u \subset P$ and $u Q y_j \subset Q$ on the node $u$. Given $V' \subseteq V$, then $G[V']$ denotes a *subgraph* of $G$ *induced* by $V'$, where an edge $\{u, v\} \in E$ is contained in $G[V']$ iff $u, v \in V'$.

## 3 IMPOSSIBILITY OF 3-RESILIENCE

In this section, we start our characterization of the feasibility of fast failover routing by showing the impossibility of 3-resiliency. We give a counter-example in Fig. 1, where no 3-resilient forwarding scheme exists that can guarantee to route from the source $s$ to the destination $t$ after three link failures.

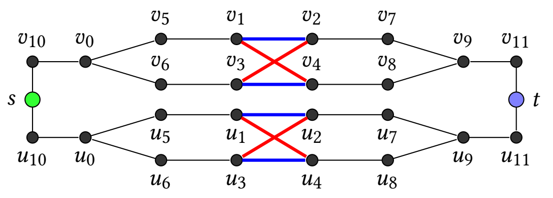


**Figure 1: Counter-example topology $G$ for 3-resilient forwarding schemes, where $s$ is the source and $t$ is the destination. We can show that each node must use a link-circular forwarding function, which has only two possible orderings for its neighbors, i.e., clockwise and anti-clockwise for the shown drawing. For instance, the clockwise and anticlockwise orderings for $v_0$ are $\langle v_{10}, v_5, v_6 \rangle$ and $\langle v_{10}, v_6, v_5 \rangle$, respectively. Setting $\{s, u_{10}\} \in F$, let $\{v_1, v_2\}, \{v_3, v_4\} \in F$ (both top blue links failed) if $v_0$ and $v_9$ have the same type of orderings (clockwise or anti-clockwise), otherwise let $\{v_1, v_4\}, \{v_2, v_3\} \in F$ (both top red links failed). Then a forwarding loop, which does not traverse $\{v_9, v_{11}\}$, always occurs in $G[V^*]$ even if $s - t$ is connected, where $V^* = \{s, t, v_0, v_1, \dots, v_{11}\}$. For example, if $v_0$ and $v_9$ use forwarding functions of clockwise and anti-clockwise orderings, respectively, and $F = \{\{s, u_{10}\}, \{v_1, v_4\}, \{v_2, v_3\}\}$, then a forwarding loop: $(s, v_{10}, v_0, v_5, v_1, v_2, v_7, v_9, v_8, v_4, v_3, v_6, v_0, v_{10}, s, v_{10})$ appears. Analogous arguments can be given for $\{s, v_{10}\} \in F$ by symmetry.**

Theorem 3.1. *The network in Fig. 1 does not permit a 3-resilient forwarding scheme.*

Proof. We first assume that a 3-resilient forwarding scheme $\Pi^{(s,t)}$ exists in the graph $G$ shown in Fig. 1. Then, for contradiction, we show that a packet originated at $s$ cannot be routed to the destination $t$ anymore, but is forwarded in a loop after removing at most 3 edges $F$ in $G$, even if $s$ has a path to $t$ in $G \setminus F$.

Let $V' = \{v_i : i \in \{0, \dots, 11\}\}$, $V^* = V' \cup \{s, t\}$, and $U' = \{u_i : i \in \{0, \dots, 11\}\}$. For each $v \in V(G)$, we define a forwarding function $\pi_v(u, F_v)$ at the node $v$, where $F_v$ denotes the failed edges incident on $v$ and the source-destination $(s, t)$ is used implicitly in this proof. Clearly, the induced graphs $G[U']$ and $G[V']$ are symmetric. By symmetry, when $F_s = \emptyset$, an arbitrary node in $\{v_{10}, u_{10}\}$ can be chosen as the outgoing port for the packet originated at $s$. Without loss of generality, we assume that $v_{10}$ is chosen.

Let $F \subseteq E(G)$ denote a set of arbitrary links, s.t., $|F| \leq 3$ and $F$ can be empty. Next, we claim that for each node $v \in V(G) \setminus \{t\}$, its forwarding function must be *link-circular*. If $v \in V(G) \setminus \{t\}$ has $\Delta_{G \setminus F}(v) = 1$, then $\pi_v(u, F_v) = u$, where $\pi_v \in \Pi^{(s,t)}$ and $u \in E_{G \setminus F}(v)$ denotes its unique neighbor in $G \setminus F$, otherwise packets get stuck at $v$. This case can be thought as a special case of the link-circular forwarding. If $v \in V(G) \setminus \{t\}$ has $\Delta_{G \setminus F}(v) = 3$, i.e., $\Delta_G(v) = \Delta_{G \setminus F}(v)$ and $F_v = \emptyset$, a *non-link-circular* forwarding function at $v$ must imply $\exists x, y \in N_{G \setminus F}(v) : \pi_v(x) = y$ and $\pi_v(y) = x$, where $N_{G\setminus F}(v) = \{x, y, z\}$. However, a non-link-circular forwarding function cannot be 3-resilient if the only $s - t$ path remained in $G \setminus F$ has to go through the link $\{v, z\}$. For example, when $F = \{\{s, u_{10}\}, \{v_5, v_1\}\}$ and $\Delta_{G\setminus F}(v_0) = 3$, if a non-link-circular forwarding function has $\pi_{v_0}(v_{10}) = v_5$ and $\pi_{v_0}(v_5) = v_{10}$, then a packet starting at $s$ cannot approach $t$ anymore even if $s - t$ is connected via $\{v_0, v_6\}$. A similar argument can be established if $\pi_{v_0}(v_{10}) = v_6$ and $\pi_{v_0}(v_6) = v_{10}$, and $F = \{\{s, u_{10}\}, \{v_6, v_3\}\}$. Moreover, for each $v \in V(G) \setminus \{t\}$ having $\Delta_{G\setminus F}(v) = 2$, a non-link-circular forwarding function at $v$ must imply $\exists x \in N_{G\setminus F}(v) : \pi_v(x) = x$ for $N_{G\setminus F}(v) = \{x, y\}$, which can make $v$ become a dead-end node, i.e., a packet cannot traverse from one neighbor of $v$ to the other neighbor to approach $t$ anymore. Therefore, each $v \in V(G) \setminus \{t\}$ must have a link-circular forwarding function.

If $v \in V(G) \setminus \{t\}$ has $\Delta_{G\setminus F}(v) = 2$, then its link-circular forwarding function is unique, i.e., from one neighbor to the other neighbor. If $v \in V(G) \setminus \{t\}$ has $\Delta_{G\setminus F}(v) = 3$, where $F_v = \emptyset$, then there are two possible circular orderings for its neighbors $N_{G\setminus F}(v)$, i.e., one clockwise and the other anti-clockwise based on their geometric locations in Fig. 1. For example, at $v_0$, the clockwise ordering of $N_G(v_0)$ is $\langle v_{10}, v_5, v_6\rangle$ and the anti-clockwise ordering of $N_G(v_0)$ is $\langle v_{10}, v_6, v_5\rangle$. Thus, for each $v \in V(G) \setminus \{t\}$ that has $\Delta_{G\setminus F}(v) = 3$, its link-circular forwarding function can choose one of two options: clockwise and anti-clockwise, arbitrarily.

Fixing $\{s, u_{10}\} \in F$, let $\{v_1, v_2\}, \{v_3, v_4\} \in F$ (blue links failed) if $v_0$ and $v_9$ have the same type (clockwise or anti-clockwise) of link-circular forwarding function, otherwise let $\{v_1, v_4\}, \{v_2, v_3\} \in F$ (red links failed). In this case, even if $s, t$ are connected in $G[V^*]$, a packet originated at $s$ will enter a forwarding loop but never traverse the link $\{v_9, v_{11}\}$ to arrive at $t$. For example, if $v_0$ and $v_9$ take forwarding functions of clockwise and anti-clockwise orderings respectively and $\{s, u_{10}\}, \{v_1, v_4\}, \{v_2, v_3\} \in F$, then a forwarding loop: $(s, v_{10}, v_0, v_5, v_1, v_2, v_7, v_9, v_8, v_4, v_3, v_6, v_0, v_{10}, s, v_{10})$ occurs. Moreover, a similar discussion can be applied when $\{s, v_{10}\} \in F$. Thus, no 3-resilient forwarding scheme for $(s, t)$ exists in Fig. 1. □

To complement Theorem 3.1, we will apply our algorithm to compute a 2-resilient forwarding scheme for the graph $G$ of Fig. 1 in §4.1.

## 4 FIRST ALGORITHMIC INSIGHTS OF 2-RESILIENCE

Given our impossibility result, the question arises, whether at least 2-resilience is feasible—previous work only showed that 1-resilience is possible [13]. We answer this question positively by presenting an algorithm, which operates on *the minimum topologies* (subgraphs) that can preserve the same s-t connectivity as the original network against two failures, but will not be prone to forwarding loops. We will call these subgraphs *gadgets (Def. 4.4)*, and a process we denote as kernelization in the following. In particular, our algorithm designs a 2-resilient forwarding scheme on a *kernel graph*, i.e., a subgraph of the original network, consisting of gadgets. We summarize these algorithmic procedures in Algorithm 1.

### 4.1 An Intuition of 2-Resiliency for Fig. 1

The challenge of our algorithm is to find the appropriate minimum topology (subgraph), which can guarantee enough $s-t$ connectivity against two failures, but also avoid forwarding loops. To better understand the inherent challenge, we will compute a 2-resilient forwarding scheme for the network $G$ of Fig. 1 by Algorithm 1, see Fig. 2.

```
Algorithm 1: Compute a 2-resilient forwarding scheme
  Input: A graph $G$ with a source-destination pair $(s, t)$
  Output: A 2-resilient forwarding scheme $\Pi^{(s,t)}$ for $G$
1 Reductions on $G$ by Claims 1 and 2, s.t., $s - t$ is 2-connected
    in $G$ and $G$ contains elementary paths $P_1$ and $P_2$ (Def. 4.1);
2 Let $E^c := \{\{e, e'\} : \{e, e'\}$ is an $s - t$ cut in $G\}$;
3 Compute a gadget (Def. 4.4) for each component $C_\ell \in G \setminus E^c$
    according to its type (Def. 4.3) (see §5);
4 Obtain a kernel graph $\mathcal{G} \subseteq G$ of $G$ by combining the gadget
    of each $C_\ell \in G \setminus E^c$, e.g., Fig., 9;
5 return a 2-resilient forwarding scheme $\Pi^{(s,t)}$ computed on
    the kernel graph $\mathcal{G}$ according to Def. 6.2 (see §6);
```

For the graph $G = (V, E)$ as shown in Fig. 1, Algorithm 1 will return its kernel graph (subgraph) $\mathcal{G} \subseteq G$ (details in Fig. 2), where the *induced* subgraph $\mathcal{G}[\{v_1, v_2, v_3, v_4\}] \subset \mathcal{G}$ contains either the red (crossing) links $\{\{v_1, v_4\}, \{v_3, v_2\}\}$ or the blue (parallel) links $\{\{v_1, v_2\}, \{v_3, v_4\}\}$, but not both, and similarly in the induced subgraph $\mathcal{G}[\{u_1, u_2, u_3, u_4\}] \subseteq \mathcal{G}$. Note that if $s - t$ is connected in $G \setminus F$, then $s - t$ is also connected in $\mathcal{G} \setminus F$ for any $F \subseteq E$ that has $|F| \leq 2$. It moreover holds that a forwarding scheme $\Pi^{(s,t)}$ by Def. 6.2, which defines a link-circular forwarding function on each node in $\mathcal{G}$, is 2-resilient in $\mathcal{G}$. Thus, $\Pi^{(s,t)}$ is also a 2-resilient forwarding scheme for $G$ since the $s - t$ connectivity in $G \setminus F$ remains in $\mathcal{G} \setminus F$ for any $|F| \leq 2$.

This is in contrast to the situation for *three* link failures, which we discussed in the previous section: For three link failures $F \subseteq E$, no subgraph $G' \subset G$ ($G$ shown as Fig. 1) can guarantee the same $s - t$ connectivity in $G' \setminus F$ as in $G \setminus F$. Thus, a kernel graph of $G$ must be $G$ itself, whose topology was shown to be not resilient against three failures in Theorem 3.1, no matter what local fast failover routing is used.

### 4.2 Challenges in General Graphs and Problem Reductions

As we saw in the impossibility proof against three failures in Fig. 1, we are posed with the dilemma of either maintaining the $s - t$ connectivity against failures, which prefers to include as many nodes and links as possible for resilient routing, or of yielding to the limited power of local forwarding functions, which easily induce forwarding loops under failures. Given a specific and small graph, it might be possible to determine an appropriate subgraph and its 2-resilient routing functions by means of brute force. However, in general, such an approach is not practical due to the combinatorial explosion of possibilities, which both makes the analysis cumbersome and the algorithmic runtime high. Moreover, a brute force approach also does not solve the underlying fundamental question if 2-resiliency is *always* possible. Given the heterogeneity of general graphs, we must thus *reduce the problem complexity* to provide a tractable solution, which is one of our main building blocks to

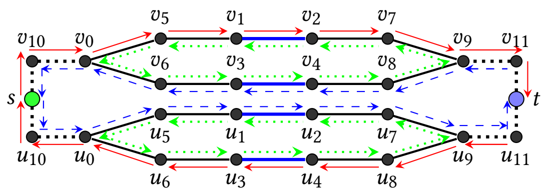


**Figure 2: Example of applying Algorithm 1 to the graph $G$ shown as Fig. 1 to obtain its kernel graph $\mathcal{G}$ (shown as this figure). The dotted lines denote the links in the set $E^c$. Given nodes $V_{0,9} = \{v_0, \ldots, v_9\}$ and $U_{0,9} = \{u_0, \ldots, u_9\}$, the induced subgraphs $G[V_{0,9}] \subset G$ and $G[U_{0,9}] \subset G$ are two components in $G \setminus E^c$ . Given components $G[V_{0,9}] \subset G$ and $G[U_{0,9}] \subset G$ of Type-4 (see Def. 4.3), we can compute two gadgets: two induced (chordless) cycles $\mathcal{G}[V_{0,9}] \subset \mathcal{G}$ and $\mathcal{G}[U_{0,9}] \subset \mathcal{G}$ respectively (see Fig. 5), s.t., each cycle contains either parallel links, i.e., $\{\{v_1, v_2\}, \{v_3, v_4\}\}$ (resp., $\{\{u_1, u_2\}, \{u_3, u_4\}\}$) or crossing links, i.e., $\{\{v_1, v_4\}, \{v_3, v_2\}\}$ (resp., $\{\{u_1, u_4\}, \{u_3, u_2\}\}$) in $E(G)$ but not both. By Def. 6.2, we obtain a forwarding scheme $\Pi^{(s,t)}$, which defines a link-circular forwarding function at each node of $\mathcal{G}$, and we can easily verify $\Pi^{(s,t)}$ as 2-resilient. In this figure, $\Pi^{(s,t)}$ is illustrated by solid (red) arcs, dotted (green) arcs, and dashed (blue) arcs respectively, s.t., at a node $v$, a packet from an incoming arc $(u, v)$ is forwarded to an outgoing arc $(v, w)$ that has the same dash pattern (color) as $(u, v)$ If an outgoing arc $(v, w)$ is failed, then the arc $(w, v)$ is considered as an incoming arc to continue forwarding on the dash pattern (color) of $(w, v)$, while a packet originated at $s$ can select either the solid (red) arc $(s, v_{10})$ or the dashed (blue) arc $(s, u_{10})$ arbitrarily to start. For example, if the link $\{v_1, v_2\}$ has failed, we might route from $s$ to $v_1$ along the solid (red) arcs, then use dotted (green) arcs until we hit $v_2$, and then use solid (red) arcs again to reach $t$.**

obtain 2-resilience. To this end, in the following, we first give some insights that the 2-resilient failover routing problem on general graphs can be reduced to a restricted problem, where $s-t$ is exactly 2-connected in $G$.

Claim 1. *The 2-resilient fast failover routing problem with a given source-destination pair $(s, t)$ can be reduced to versions, where source and destination are exactly 2-connected.*

Proof. Consider a graph $G = (V, E)$ with $s, t \in V$. If $s - t$ is at least 3-connected in $G$, then we can find three edge-disjoint $s - t$ paths $\mathcal{P} = \{P_1, P_2, P_3\}$ in $G$. Since there are $|F| \leq 2$ failed links, there must be a path $P \in \mathcal{P}$ always connected in $G \setminus F$. Let $\Pi^{(s,t)}$ be a forwarding scheme working as follows: first send packets along $P_1$ starting at $s$, and when encountering a link failure, then bounce back to $s$ and try the next path $P_2$ and then $P_3$ until arriving at $t$. Hence, $\Pi^{(s,t)}$ is 2-resilient, as observed in [19].

When $s - t$ is 1-connected in $G$, let $P$ be an $s - t$ path in $G$. Let $E' \subseteq E$ be a set of $s - t$ *bridges*, s.t., $\forall e \in E'$, $s - t$ is disconnected in $G \setminus \{e\}$. Hence, for each $e \in E'$, it implies $e \in E(P)$, otherwise $s - t$ is still connected by $P$ in $G \setminus \{e\}$. For each connected component $G_i \subseteq G \setminus E'$, let $s_i \in V(G_i)$ (resp., $t_i \in V(G_i)$) be the node in $V(P)$

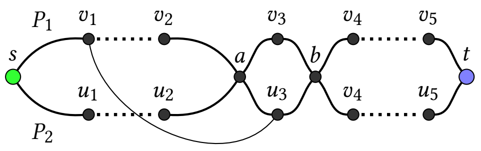


**Figure 3: An example graph $G$ for Claim 2, where $b$ is an $s - t$ articulation point of $G$ but $a$ is not: as $s - t$ is connected after removing $a$ in $G$, there are two edge-disjoint $s - t$ paths $P_1$ and $P_2$, that do not both contain $a$, by including the edge $\{v_1, u_3\}$ in $P_1$. But, since $s - t$ is disconnected in $G \setminus \{b\}$, any two $s - t$ edge-disjoint paths must both contain $b$. Thus, the problem is split into two sub-problems with source-destination pairs $(s, b)$ and $(b, t)$ respectively, s.t., each sub-problem has two edge-disjoint paths $P_1$ and $P_2$, which are node-disjoint except for source and destination.**

that is closest to the source $s$ (resp., the destination $t$) in $P$. If each component $G_i$ has a 2-resilient forwarding *scheme* $\Pi^{(s_i,t_i)}$, then a 2-resilient forwarding scheme $\Pi^{(s,t)}$ exists in $G$. Now, the problem becomes to find $\Pi^{(s_i,t_i)}$ for each $G_i$. If $s_i$ and $t_i$ are 3-connected in $G_i$, $\Pi^{(s_i,t_i)}$ can be computed as mentioned above by 3 edge-disjoint paths already.

Thus, the 2-resilient failover routing problem can be reduced to a simplified version, where $s - t$ is exactly 2-connected in $G$. □

For a graph $G$ that has its $s - t$ edge-connectivity at exactly 2, there must be two edge-disjoint $s - t$ paths, but it is not mandatory to have two node-disjoint $s - t$ paths. However, node-disjoint $s - t$ paths can effectively simplify our algorithms. Thus, by Claim 2, we show that the reduced problem can be further simplified into subproblems, s.t., source and destination are 2-node-connected.

Claim 2. *The 2-resilient fast failover routing problem, where $s-t$ is exactly 2-connected in $G$, can be reduced to versions, where there exist two node-disjoint $s-t$ paths $P_1$ and $P_2$, s.t., $V(P_1) \cap V(P_2) \setminus \{s, t\} = \emptyset$, implying that $s - t$ is at least 2-node-connected in $G$.*

Proof. If $s - t$ is already 2-node-connected in $G$, then we can directly compute two node-disjoint $s-t$ paths $P_1$ and $P_2$ in $G$ within $O(|V|)$ [22]. In the following, we consider that $s-t$ is exactly 2-edge-connected but 1-node-connected in $G$, e.g., Fig 3. An *$s-t$ articulation point* of $G$ is a node $v \in V \setminus \{s, t\}$, s.t., $s-t$ is disconnected in $G \setminus \{v\}$, and we can compute the set $J$ of all $s - t$ articulation points of $G$ within $O(|V| + |E|)$ [4]. Clearly, any two edge-disjoint $s - t$ paths $P$ and $P'$ in $G$ must have $J \subseteq V(P) \cap V(P') \setminus \{s, t\}$ (see Fig 3), otherwise there must be a node $u \in J$ but $u \notin V(P) \cap V(P')$, contradicting that $s - t$ is still connected by $P \cup P'$ in $G \setminus \{u\}$.

Let $J = \{v_1, \ldots, v_j\}$ be sorted s.t., every two consecutive nodes $v_i, v_{i+1} \in J$ satisfy that $v_{i+1}$ is connected with $t$ in the graph $G \setminus v_i$. We note that any $s-t$ path $P$ in $G$ must traverse all $s-t$ articulation points in the same order of $J$. Now, our 2-resilient failover routing problem with the given pair $(s, t)$ can be divided into a set of sub-problems for different source-destination pairs: $(s, v_1)$, $(v_j, t)$ and $(v_i, v_{i+1})$ for $i \in \{1, \ldots, j-1\}$, s.t., in each sub-problem, its source $s$ and destination $t$ are 2-node-connected and we can compute two node-disjoint $s - t$ paths $P_1$ and $P_2$ within $O(|V|)$, e.g., Fig 3. □

Henceforth, by Claim 2, we will focus on the 2-resilient failover routing problem on graphs, where $s-t$ is 2-node-connected in $G$.

*Definition 4.1 (Elementary Paths).* Given a graph $G=(V,E)$, where $s-t$ is exactly 2-connected and at least 2-node-connected, let $P_1$ and $P_2$ denote two arbitrary node-disjoint $s-t$ paths in $G$. From now on, we call $P_1$ and $P_2$ as elementary paths for $(s,t)$ in $G$.

## 4.3 Understanding Connected Components After Two Failures

After applying the above reductions on the 2-resilient failover problem, with Lemma 4.2 we can show that any two failures that can disconnect $s$ and $t$ in $G$ must be on *elementary paths* $P_1$ and $P_2$ (Definition. 4.1). However, two arbitrary failures $F$ on $P_1$ and $P_2$, which can disconnect $P_1$ and $P_2$, do not necessarily disconnect $s$ and $t$ in $G \setminus F$. Thus, devising 2-resilient failover forwarding relies on understanding these structures in addition to $P_1$ and $P_2$, which can continue connecting $s$ and $t$ in $G \setminus F$ when $P_1$ and $P_2$ are disconnected by $F$.

Lemma 4.2. *Given a graph $G=(V,E)$, where $s,t \in V$ are exactly 2-connected, let $P_1, P_2$ be elementary paths for $(s,t)$. Then, for any two distinct edges $e_1, e_2 \in E$, s.t., $s-t$ is disconnected in $G \setminus \{e_1,e_2\}$, then one edge must be on $P_1$ and the other on $P_2$, denoted by $e_1 \in E(P_1)$ and $e_2 \in E(P_2)$.*

Proof. We prove the lemma by contradiction. If $\{e_1,e_2\}$ are not on $P_1$ and $P_2$ respectively, there must be a path $P \in \{P_1,P_2\}$ that is still connected in the graph $G \setminus \{e_1,e_2\}$, which contradicts that $s$ and $t$ are disconnected in $G \setminus \{e_1,e_2\}$. □

For every two distinct edges $e_1,e_2 \in E$, if $s-t$ is disconnected in $G \setminus \{e_1,e_2\}$ then $\{e_1,e_2\}$ is called a 2-$(s,t)$*-cut*. After removing all 2-$(s,t)$-cuts in $G$, let the remaining subgraph be $G'$. Then, for any two failures $F \subseteq E(G)$, if $F \cap E(G') \neq \emptyset$, then $s-t$ is connected in $G \setminus F$, and we need to design a 2-resilient scheme for this case. If $F$ is a 2-$(s,t)$-cut, i.e., $F \cap E(G') = \emptyset$, then we do not need to retain resilience due to $s-t$ being disconnected. Thus, the subgraph $G'$, which might contain multiple connected components, provides the topological information for routing.

With Definition 4.3, all possible connected components in $G'$ are characterized and classified by their topological features.

*Definition 4.3 (Types of Connected Components).* Given a graph $G=(V,E)$, where $s \in V$ and $t \in V$ are exactly 2-connected, let $P_1$ and $P_2$ be elementary paths for $(s,t)$. Let $E^c(G)$ denote the set of all 2-$(s,t)$-cuts in $G$, and $E^c(G)$ can be abbreviated as $E^c$ when $G$ is clear. By removing all edges in $E^c$ from $G$, we obtain a graph $G \setminus E^c$, which consists of a set of *connected components* (E.g., see Fig. 4). Let $C_\ell$ denote each connected component in $G \setminus E^c$, and then we define a set $C$ as follows: $C = \{C_\ell : |V(C_\ell) \cap V(P_1 \cup P_2)| \geq 2\}$. If $C_\ell$ contains at most one node from either $P_1$ or $P_2$, then $C_\ell \notin C$. If $C_\ell \in C$ has $s \in V(C_\ell)$ (resp., $t \in V(C_\ell)$), then it is called the *head (resp., tail) component*, otherwise it is called a *normal component*. Note that it is not necessary to have a head/tail component in $C$.

Moreover, a function $\partial(C_\ell, P')$ is defined as follows:

$$\forall C_\ell \in C, \forall P' \in \{P_1,P_2\} : \partial(C_\ell, P') = V(C_\ell) \cap V(P') \setminus \{s,t\}.$$

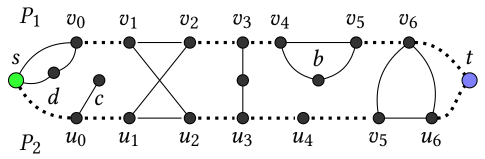


**Figure 4: Illustration of Definition 4.3: each connected component in $G \setminus E^c$ is shown by solid lines and $E^c$ by dotted lines.**

Let $P,\ Q \in \{P_1,P_2\}$ and $Q \neq P$. If $C \neq \emptyset$, for each $C_\ell \in C$, when $C_\ell$ is head or tail, then its type can be defined as follows:

- **Type-1:** $|\partial(C_\ell,P)| \geq 1$ and $\partial(C_\ell,Q) = \emptyset$,
- **Type-2:** $|\partial(C_\ell,P)| \geq 1$ and $|\partial(C_\ell,Q)| \geq 1$,

and when $C_\ell$ is normal, then its types can be defined as follows:

- **Type-3:** $|\partial(C_\ell,P)| = 1$ and $|\partial(C_\ell,Q)| = 1$,
- **Type-4:** $|\partial(C_\ell,P)| \geq 2$ and $\partial(C_\ell,Q) = \emptyset$,
- **Type-5:** $|\partial(C_\ell,P)| \geq 2$ and $|\partial(C_\ell,Q)| = 1$,
- **Type-6:** $|\partial(C_\ell,P)| \geq 2$ and $|\partial(C_\ell,Q)| \geq 2$.

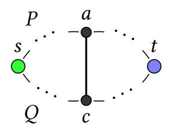
In Definition 4.3, the given types characterize all possible cases for connected components. If a connected component $C_\ell \notin C$, it must have $|V(C_\ell) \cap V(P_1 \cup P_2)| \leq 1$, which further implies $E(C_\ell) \cap E(P_1 \cup P_2) = \emptyset$. Thus, if a connected component $C_\ell \notin C$ has $F \cap E(C_\ell) \neq \emptyset$, then $s$ and $t$ can still communicate in $G \setminus F$ by a path $P \in \{P_1,P_2\}$. Therefore, any $C_\ell \notin C$ can be ignored in our further discussion.

To proceed, we define some concepts for each $C_\ell \in C$.

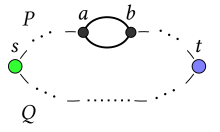
**Ports and Key-Nodes of Components:** In Definition 4.3, for each $C_\ell \in C$ and each $P \in \{P_1,P_2\}$, if $\partial(C_\ell,P) \neq \emptyset$, *the left-$P$ port* (resp., *right-$P$ port*) of $C_\ell$ is defined as the *first-traversed* node in $\partial(C_\ell,P)$ when walking along the path $P$ from $s$ to $t$ (resp., from $t$ to $s$). If the left-$P$ port and the right-$P$ port of $C_\ell$ are identical, then it is called *the $P$ port* of $C_\ell$. Moreover, if $C_\ell$ is the head (resp., tail) component, we only consider the right-$P$ (resp., the left-$P$) port of $C_\ell$. For each $C_\ell \in C$, all ports of $C_\ell$ and the source $s$ (resp., destination $t$) if it is contained in $C_\ell$ are called the *key-nodes* of $C_\ell$. Even when two connected components have the same type, they can be quite different. However, we are only interested in the minimum subgraph of each component $C_\ell$ for devising forwarding functions, s.t., these subgraphs are enough to guarantee the same $s-t$ connectivity after two failures $F$. In terms of the $s-t$ connectivity against failures, different components of the same type can be uniformly described, which are called *gadgets* and defined in Definition 4.4. Thanks to the higher-level abstraction given by gadgets, we can design forwarding functions by focusing on a handful of simple topologies instead of devising them for components.

*Definition 4.4 (Gadgets of Connected Components).* For each component $C_\ell \in C$, let $G_\ell \subseteq C_\ell$ be a (connected) subgraph of $C_\ell$, s.t., $G_\ell$ contains all *key-nodes* of $C_\ell$.

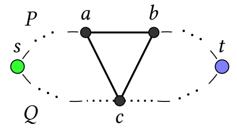
Let $P_1 \cup P_2$ be a graph consisting of two *elementary paths* $P_1$ and $P_2$ in $G$, and then let $T_\ell = P_1 \cup P_2 \setminus E(C_\ell)$ be a subgraph of $P_1 \cup P_2$ without edges from $C_\ell$. *The test graph $\mathcal{T}(G_\ell)$ of $G_\ell$* is defined as $\mathcal{T}(G_\ell) = G_\ell \cup T_\ell$, i.e., a graph obtained by combining $G_\ell$ and $T_\ell$. Moreover, *the simple test graph $\mathcal{T}'(G_\ell)$ of $G_\ell$* is generated by connecting the left-$P$ (resp., right-$P$), where $P \in \{P_1,P_2\}$, port $u$ of



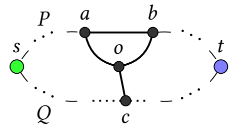
(a) **Type**-3



(b) **Type**-4



(c) **Type**-5



(d) **Type**-5

**Figure 5: Path-contracted patterns of gadgets in components of Type-3, Type-4 and Type-5, where $P, Q \in \{P_1, P_2\}$ and $P \neq Q$, the nodes $a$ and $b$ denote the left-$P$ and the right-$P$ ports respectively, and the node $c$ indicates the $Q$ port. We also note that the node $o$ in Fig. 5d can be $a$ or $b$.**

$G_\ell$ to $s$ (resp., $t$) by an edge $\{s, u\}$ (resp., $\{t, u\}$) if that corresponding port $u$ exists in $G_\ell$. SPAA Then, the subgraph $G_\ell \subseteq C_\ell$ is called *a gadget* of $C_\ell$, denoted by $\mathcal{G}(C_\ell)$, if $s - t$ can be disconnected by removing two edges $\{e, e'\}$ in $\mathcal{T}(G_\ell)$ (resp., $\mathcal{T}'(G_\ell)$), then it implies $\{e, e'\} \cap E(G_\ell) = \emptyset$.

A component might contain multiple gadgets of essentially different topologies. To better illustrate variant topologies of gadgets by a handful of figures, the gadgets are further abstracted into so-called *patterns*. We remark that patterns are introduced only for presenting this paper, which are unnecessary in the practical executions of our algorithms.

**Pattern of Gadgets:** If two sub-paths $(x, y, z)$ and $(x', y, z')$ have a joint node $y$, then we can split $y$ into two duplicates: $y$ and $y'$, s.t., $(x, y, z)$ and $(x', y', z')$ are not overlapped at $y$ anymore. Given a gadget $G_\ell$ of a component $C_\ell$, by recursively splitting joint nodes on $G_\ell$, we can obtain a graph $G'_\ell$ as a *pattern* of $G_\ell$, s.t., any *induced cycle* in $G'_\ell$ must contain at least two key-nodes of $C_\ell$ (exceptionally, if $G_\ell$ is a case shown as Fig. 6a, then any induced cycle in its pattern $G'_\ell$ should contain all four ports). The topology of a pattern $G'_\ell$ can be further abstracted by using *path-contraction*, s.t., each node $v$ that is not a key-node and has a degree of 2 in $G'_\ell$ is not explicitly shown, namely *a path-contracted pattern.* In the following, the patterns presented in our figures are all path-contracted.

## 5 COMPUTING GADGETS

In the previous section, we presented some first insights that allowed us to only consider restricted problem instances. This section is now dedicated to the computation of these subdivisions and proving their desired properties. We will develop methods to compute a gadget for each possible type of components, as introduced in Definition 4.3.

Lemma 5.1. *For a normal component $C_\ell \in C$ of **Type**-3, let $a = V(C_\ell) \cap V(P)$ and $c = V(C_\ell) \cap V(Q)$, where $P, Q \in \{P_1, P_2\}$ and $P \neq Q$, then a gadget of $C_\ell$ can be a path (edge) from $a$ to $c$, and its pattern is shown as Fig. 5a.*

Proof. If there is no path between $a$ and $c$ in $C_\ell$, then $C_\ell$ is not a connected component of **Type**-3 according to Def. 4.3. By contradiction, $a$ and $c$ are connected by at least one path $P_{a,c}$ in $C_\ell$. Hence, by Def. 4.4, we can verify that $P_{a,c}$ is a gadget of $C_\ell$. □

Lemma 5.2. *For a component $C_\ell \in C$ of **Type**-1 or **Type**-4, let the left-$P$ port and the right-$P$ port of $C_\ell$ be $a \in V(P)$ and $b \in V(P)$ respectively, where $P \in \{P_1, P_2\}$. Note that $a = s$ when $C_\ell$ is head and $b = t$ when $C_\ell$ is tail. Then $C_\ell$ contains at least two edge-disjoint paths $P_{a,b}$ and $P'_{a,b}$ between $a$ and $b$, and the graph $P_{a,b} \cup P'_{a,b}$ is a gadget of $C_\ell$. The corresponding path-contracted patterns are shown in Fig. 8a (resp., Fig. 8d) when $C_\ell$ is head (resp., tail) of **Type**-1, and in Fig. 5b when $C_\ell$ is of **Type**-4.*

Proof. Let $Q = \{P_1, P_2\} \setminus P$. By Definition 4.3, $C_\ell$ (**Type**-1 or **Type**-4) has $V(C_\ell) \cap (V(Q) \setminus \{s, t\}) = \emptyset$, implying $E(C_\ell) \cap E(Q) = \emptyset$. Clearly, the sub-path $aPb$ of $P$ is included in $C_\ell$. If $a$ and $b$ are 1-connected in $C_\ell$, then there must be an edge $e^* \in E(aPb)$, whose removal disconnects $a$ and $b$ in $C_\ell$. Then, $\{e^*, e'\}$, where $e' \in E(Q)$, must be a 2-$(s, t)$-cut in $E^c$, which further implies that the connected component $C_\ell$ containing both $a$ and $b$ cannot exist in $G \setminus E^c$. Therefore, $a - b$ must be 2-connected in $C_\ell$, implying at least two edge-disjoint paths $P_{a,b}$ and $P'_{a,b}$ from $a$ to $b$ in $C_\ell$. By Definition 4.4, $P_{a,b} \cup P'_{a,b}$ is hence a gadget in $C_\ell$. When $C_\ell$ is head (resp. tail) of **Type**-1 or **Type**-4, its path-contracted patterns can be shown in Fig. 8a (resp., Fig. 8d) or in Fig. 5b accordingly. □

After showing the simpler cases, **Type**-1, **Type**-3, and **Type**-4, we will focus on the most complicated case, **Type**-6, which has the maximum number of variants for gadgets. To exhaust different gadgets of **Type**-6 caused by the varying number of edge-disjoint $s - t$ paths in $C_\ell$, we need to introduce a notion called *augmented component.* Comparing to the test graphs defined in Def. 4.4, the graph defined by augmented component aids in computing at least 3 edge-disjoint paths between $s$ and $t$.

*Definition 5.3 (Augmented Component $\mathcal{A}(C_\ell)$).* For each component $C_\ell \in C$, the *augmented component* $\mathcal{A}(C_\ell)$ of $C_\ell$ is a multigraph having $C_\ell \subseteq \mathcal{A}(C_\ell)$ such that

- if $C_\ell \in C$ is head, then $V(\mathcal{A}(C_\ell)) = V(C_\ell) \cup \{t\}$ and $t$ has two parallel edges connecting each right-$P$ port of $C_\ell$, where $P \in \{P_1, P_2\}$.
- if $C_\ell \in C$ is tail, then $V(\mathcal{A}(C_\ell)) = V(C_\ell) \cup \{s\}$ and $s$ has two parallel edges connecting each left-$P$ port of $C_\ell$, where $P \in \{P_1, P_2\}$.
- if $C_\ell \in C$ is normal, then $V(\mathcal{A}(C_\ell)) = V(C_\ell) \cup \{s, t\}$ and $s$ has two parallel edges connecting each left-$P$ port of $C_\ell$ and $t$ has two parallel edges connecting each right-$P$ port of $C_\ell$, where $P \in \{P_1, P_2\}$.

Lemma 5.4. *Given a connected component $C_\ell \in C$ of **Type**-6 or **Type**-2, then $s$ and $t$ are at least 3-connected in the graph (augmented component (Def. 5.3)) $\mathcal{A}(C_\ell)$.*

Proof. We assume that $s$ and $t$ are at most 2-connected in $\mathcal{A}(C_\ell)$. There must be two edges $\{e_1, e_2\}$ in $\mathcal{A}(C_\ell)$ s.t., $s - t$ is disconnected in $\mathcal{A}(C_\ell) \setminus \{e_1, e_2\}$. By observation on $\mathcal{A}(C_\ell)$, $s$ and $t$ cannot be disconnected in $\mathcal{A}(C_\ell) \setminus \{e_1, e_2\}$ if $\{e_1, e_2\} \cap E(C_\ell) = \emptyset$ due to these four parallel edges from $s$ (resp., $t$) to the left-$P$ (resp.,

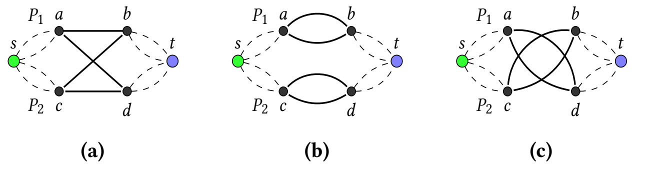
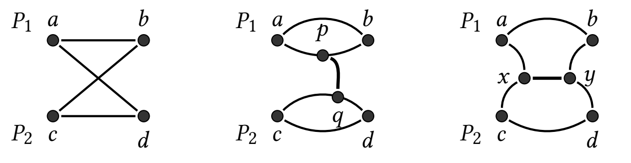


**Figure 6: Three possible outputs when computing 4 edge-disjoint paths from $s$ to $t$ in $\mathcal{A}(C_\ell)$, where $C_\ell \in C$ has Type-6.**



**(a) Pattern for gadgets in components of Type-6 when $s-t$ is 4-connected in $\mathcal{A}(C_\ell)$**

**(b) Pattern for gadgets in components of Type-6 when $s-t$ is 4-connected in $\mathcal{A}(C_\ell)$**

**(c) Pattern for gadgets in components of Type-6 when $s-t$ is 3-connected in $\mathcal{A}(C_\ell)$**

**Figure 7: The path-contracted patterns for all gadgets in components of Type-6.**

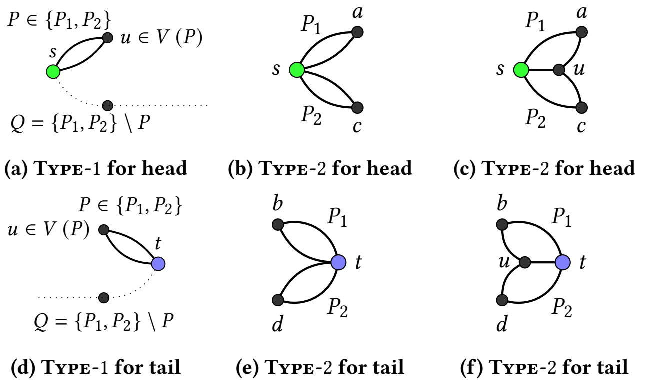
the right-$P$) ports of $C_\ell$ respectively, where $P \in \{P_1, P_2\}$. Moreover, $s-t$ cannot be disconnected in $\mathcal{A}(C_\ell) \setminus \{e_1, e_2\}$ if $\{e_1, e_2\} \cap E(C_\ell) = e$, otherwise $s-t$ is disconnected in $G \setminus \{e\}$, contradicting that $s-t$ is 2-connected in $G$. Thus, it must have $\{e_1, e_2\} \subseteq E(C_\ell)$, which further implies that $s-t$ is disconnected in $G \setminus \{e_1, e_2\}$, and then $C_\ell$ cannot exist in $G \setminus E^c$. Thus, this contradiction shows at least 3-connectivity of $s-t$ in $\mathcal{A}(C_\ell)$. □

If applying a maximum flow algorithm on $\mathcal{A}(C_\ell)$, where $C_\ell \in C$ is of **Type**-6, returns four edge-disjoint $s-t$ paths in $\mathcal{A}(C_\ell)$, then all possible outputs can be classified to three types, whose patterns are shown in Fig. 6 respectively. We can easily verify that the output with its pattern of Fig. 6a is a gadget directly, whose pattern is shown in Fig. 7a. If a output has its pattern as shown in Fig. 6b (resp., Fig. 6c), then by interconnecting these independent connected components contained in the output strategically, we can obtain a gadget of the pattern as shown in Fig. 7b (resp., Fig. 7a). We summarize these results into Lemma 5.5 and defer its formal proof to the full version of this paper due to space constraints.

Lemma 5.5. *For a component $C_\ell \in C$ of* ***Type****-6, if $s-t$ is at least 4-connected in the augmented component $\mathcal{A}(C_\ell)$, then $C_\ell$ has a gadget with a path-contracted pattern shown in Fig. 7a or Fig. 7b.*

The ideas used to prove the Lemmas 5.1–5.5, for computing gadgets of **Type**-1, **Type**-3, **Type**-4 and **Type**-6, can be applied analogously to remaining cases for other types of the normal, head and tail components. Due to space limitations, we omit repeating the proof details and cast the statement as Lemma 5.6. The proofs of these cases in Lemma 5.6, which were unmentioned before, are deferred to the full version of this paper.

Lemma 5.6. *For each connected component $C_\ell \in C$, by its type defined in Def. 4.3 and the $s-t$ edge-connectivity in $\mathcal{A}(C_\ell)$ (Def. 5.3),*



**(a) Type-1 for head** **(b) Type-2 for head** **(c) Type-2 for head**

**(d) Type-1 for tail** **(e) Type-2 for tail** **(f) Type-2 for tail**

**Figure 8: The path-contracted patterns of gadgets contained in head and tail components classified by types, where $P, Q \in \{P_1, P_2\}$ and $P \neq Q$.**

*we can compute a gadget in $C_\ell$, which has a (path-contracted) pattern illustrated by a figure indicated in Table 2.*

**Table 2: Figures for patterns sorted by types of $C_\ell$**

| *The type of $C_\ell \in C$* | *The edge-connectivity of $s-t$ in $\mathcal{A}(C_\ell)$* | *Figures for patterns of gadgets in $C_\ell$* |
|---|---|---|
| ***Type**-1* | | *Fig. 8a or Fig. 8d* |
| ***Type**-2* | *4-connected* | *Fig. 8b or Fig. 8e* |
| ***Type**-2* | *3-connected* | *Fig. 8c or Fig. 8f* |
| ***Type**-3* | | *Fig. 5a* |
| ***Type**-4* | | *Fig. 5b* |
| ***Type**-5* | | *Fig. 5c or Fig. 5d* |
| ***Type**-6* | *3-connected* | *Fig. 7c* |
| ***Type**-6* | *4-connected* | *Fig. 7a or Fig. 7b* |

## 6 DETERMINING A 2-RESILIENT FORWARDING SCHEME

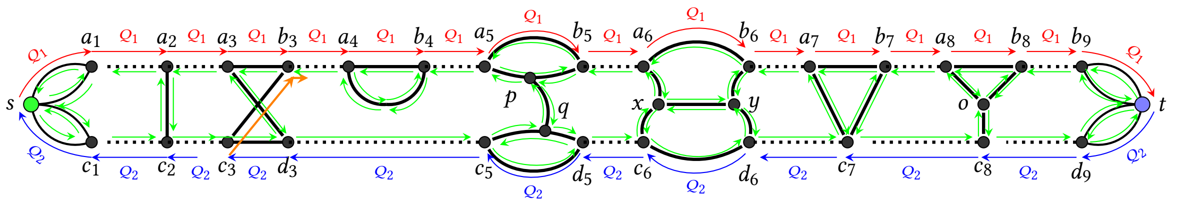
In this section, we combine these gadgets into a new graph, henceforth called *kernel graph*, which is a *subgraph* of the given network $G$. We then devise a 2-resilient forwarding scheme for this kernel graph.

### 6.1 Computing the Kernel Graph of a Given Network $G$

We define a *kernel graph* $\mathcal{G} \subseteq G$ of the given network $G$, which is constructed based on gadgets computed in each component $C_\ell \in C$ of $G$ by Lemmas 5.1–5.6, e.g., Fig. 9.

**Kernel Graph $\mathcal{G}$:** Given a graph $G = (V, E)$, where $s-t$ is exactly 2-connected, then the graph $\mathcal{G} = \left(\bigcup_{C_\ell \in C} \mathcal{G}(C_\ell)\right) \cup E^c$ is called a *kernel graph* (e.g., Fig. 9), where $E^c$ denotes the set of all 2-$(s,t)$-cuts in $G$ and $\mathcal{G}(C_\ell) \subseteq C_\ell$ denotes a gadget computed in each connected component $C_\ell \in C$.

Recall the definition of elementary paths $P_1$ and $P_2$. As the kernel graph $\mathcal{G} \subseteq G$, $P_1$ and $P_2$ might not remain in $\mathcal{G}$. Thus, we introduce the concepts of *primary path* $Q_1$ and *secondary path* $Q_2$ based on a



$s$
$t$
$a_1$ $a_2$ $a_3$ $b_3$ $a_4$ $b_4$ $a_5$ $b_5$ $a_6$ $b_6$ $a_7$ $b_7$ $a_8$ $b_8$ $b_9$
$c_1$ $c_2$ $c_3$ $d_3$ $c_5$ $d_5$ $c_6$ $d_6$ $c_7$ $c_8$ $d_9$
$p$ $q$ $x$ $y$ $o$
$Q_1$
$Q_2$

**Figure 9: Example of a kernel graph, where each gadget $\mathcal{G}(C_\ell)$ is a pattern drawn by bold lines and $E^c$ is denoted by dotted lines. The directed paths and cycles indicate the forwarding rules defined in Definition 6.2. In the case of no failures, the routing starts at $s$ and follows the directed path $\vec{Q}_1$ to $t$. When traversing a directed path/cycle and a failure (arc) $(v,u)$ is encountered at $v$, we then first route along another directed path/(cycle) that contains the arc $(u,v)$, to proceed. For example, consider the failed links as $\{a_3,b_3\}$ and $\{d_3,c_5\}$. After arriving at $a_3$ along $\vec{Q}_1$ from $s$, the packet encounters $(a_3,b_3)$ as a failure, and is routed along the detour $(a_3,d_3)$ to $d_3$, from where it encounters a failure $(d_3,c_5)$. Then, the packet is routed along the detour $(d_3,c_3)$, $(c_3,b_3)$ (an *additional rule* as defined in Definition 6.2) to $b_3$, where routing along $\vec{Q}_1$ to $t$ can resume. We give a second example: if $\{b_7,a_8\}$ fails, then, at $b_7$, the packet travels along $(b_7,c_7,c_8,o,a_8)$ to $a_8$ and then follows along $\vec{Q}_1$ to $t$.**

kernel graph $\mathcal{G}$, which will be used for computing our 2-resilient forwarding scheme.

**Primary Path $Q_1$ and Secondary Path $Q_2$:** Given a kernel graph $\mathcal{G} = \left(\bigcup_{C_\ell \in C} \mathcal{G}(C_\ell)\right) \cup E^c$, for each gadget $\mathcal{G}(C_\ell)$, let $aP_1(C_\ell)\, b \subseteq \mathcal{G}(C_\ell)$ be a path from the left-$P_1$ port $a$ to the right-$P_1$ $b$ in $\mathcal{G}(C_\ell)$ that corresponds to the edge $\{a,b\}$ in the (path-contracted) patterns, where $a = s$ if $C_\ell$ is head, and $b = t$ if $C_\ell$ is tail. Then *the primary path* defined as $Q_1 := \left(\bigcup_{C_\ell \in C} aP_1(C_\ell)\, b\right) \cup (E(P_1) \cap E^c)$ is a path from $s$ to $t$ including all $P_1$-Ports of each $\mathcal{G}(C_\ell)$. Similarly, for each gadget $\mathcal{G}(C_\ell)$, let $cP_2(C_\ell)\, d \subseteq \mathcal{G}(C_\ell)$ be a path from the left-$P_2$ port $c$ to the right-$P_2$ port $d$ in $\mathcal{G}(C_\ell)$ that corresponds to $\{c,d\}$ in the (path-contracted) patterns, where $c = s$ if $C_\ell$ is head, and $d = t$ if $C_\ell$ is tail. Then *the secondary path* defined as $Q_2 := \left(\bigcup_{C_\ell \in C} cP_2(C_\ell)\, d\right) \cup (E(P_2) \cap E^c)$ is a path from $s$ to $t$ including all $P_2$-Ports of each $\mathcal{G}(C_\ell)$. Clearly, $Q_1$ and $Q_2$ are two edge-disjoint paths from $s$ to $t$ in $\mathcal{G}$.

For the correctness, we first need to show the connectivity of $s-t$ is same in $\mathcal{G} \setminus F$ and $G \setminus F$ for any two failures $F \subseteq E$.

Lemma 6.1. *Given $G = (V,E)$, where $s-t$ is exactly 2-connected, let $\mathcal{G} = \left(\bigcup_{C_\ell \in C} \mathcal{G}(C_\ell)\right) \cup E^c$ be a kernel graph obtained from $G$. For any two edges $F = \{e_1,e_2\} \subseteq E$, if $s-t$ is connected in $G \setminus F$, then $s-t$ is also connected in $\mathcal{G} \setminus F$.*

Proof. The kernel graph $\mathcal{G}$ is a subgraph of $G$. Moreover, we can find the primary path $Q_1$ and the secondary path $Q_2$ in $\mathcal{G}$. Let $F \subseteq E$ denote two arbitrary edges in $G$, s.t., $s-t$ is connected in $G \setminus F$. If $|F \cap E(\mathcal{G})| \le 1$, then $s-t$ must be connected in $\mathcal{G} \setminus F$ since at least one of $Q_1$ and $Q_2$ is still connecting $s$ and $t$. Now, we assume $|F \cap E(\mathcal{G})| = 2$. Recall the definition of gadgets (Definition 4.4). Please note that each $\mathcal{G}(C_\ell)$ is a gadget. Here, each subgraph $\mathcal{G}(C_\ell) \cup (Q_1 \cup Q_2 \setminus E(\mathcal{G}(C_\ell))) \subseteq \mathcal{G}$ actually becomes a test graph $\mathcal{T}(\mathcal{G}(C_\ell))$ for the gadget $\mathcal{G}(C_\ell)$. By the definition of gadgets, $s-t$ must be connected in $\mathcal{T}(\mathcal{G}(C_\ell)) \setminus F$, which further implies the connectivity of $s-t$ in $\mathcal{G} \setminus F$. □

## 6.2 Determination of a 2-Resilient Forwarding Scheme

We will now design a 2-resilient routing scheme for the kernel graph $\mathcal{G} \subseteq G$. We first consider a simplified version of $\mathcal{G}$, s.t., each gadget is replaced by its pattern, calling it a *kernel pattern*, denoted by $\mathcal{G}'$. Next, we develop a 2-resilient forwarding scheme for the kernel pattern $\mathcal{G}'$ and show that it implies a 2-resilient forwarding scheme for a kernel graph $\mathcal{G}$ of $G$. We remark that the kernel pattern is employed here to simplify illustrating our forwarding scheme and proving correctness, not necessary to be implemented in practice.

We first introduce the concept of a *reduced kernel pattern* $\mathcal{G}^*$ and its features in Observation 1.

Observation 1. *Given a kernel graph $\mathcal{G} = \left(\bigcup_{C_\ell \in C} \mathcal{G}(C_\ell)\right) \cup E^c$, we replace each $\mathcal{G}(C_\ell)$ by its (path-contracted) pattern $\mathcal{G}'(C_\ell)$ to obtain a graph $\mathcal{G}'$, called* a kernel pattern. *We can compute the primary path $Q_1$ and the secondary path $Q_2$ of $\mathcal{G}'$. If $\mathcal{G}'$ contains a pattern $\mathcal{G}'(C_\ell)$ as Fig. 7a, then we remove the edge (path) $\{c,b\}$ from $\mathcal{G}'$, where $c$ is the left-$P_2$ port and $b$ is the right-$P_1$ port. By removing these edges, $\mathcal{G}'$ is reduced to a new graph $\mathcal{G}^*$, henceforth called a* reduced kernel pattern. *Then, by observing all possible patterns in Lemma 5.6, we know that $\mathcal{G}^*$ is a planar graph having $Q_1 \cup Q_2$ as the* boundary of its outer face *[12] on a planar drawing (e.g., Fig. 9) and any* induced cycle (chordless cycle) [12] *$f_i$ in $\mathcal{G}^*$ must have $E(f_i) \cap E(Q_2 \cup Q_1) \neq \emptyset$.*

We now define a forwarding scheme for kernel pattern $\mathcal{G}'$:

*Definition 6.2 (2-Resilient Forwarding Scheme on a Kernel Pattern).* Given a kernel graph $\mathcal{G}$, we obtain its kernel pattern $\mathcal{G}'$, reduced kernel pattern $\mathcal{G}^*$, the primary path $Q_1$ of $\mathcal{G}'$, and the secondary path $Q_2$ of $\mathcal{G}'$, respectively, according to Observation 1. In $\mathcal{G}^*$, we define the followings:

- Directed path $\vec{Q}_1$: traverse $Q_1$ from $s$ to $t$;
- Directed path $\vec{Q}_2$: traverse $Q_2$ from $t$ to $s$;
- Directed cycles $\vec{f_i}$: for each *induced cycle (chordless cycle)* $f_i$ in $\mathcal{G}^*$, if it contains an edge of $Q_1$, then traverse $f_i$ in a direction reverse to $\vec{Q}_1$, otherwise traverse $f_i$ in a direction reverse to $\vec{Q}_2$.

Given a directed path (cycle) $\vec{P} \in \{\vec{Q}_1, \vec{Q}_2, \vec{f_i}\}$, for each $v \in V(\vec{P}) \setminus \{t\}$, define a forwarding rule at $v$ as follows: from its incoming neighbor in $\vec{P}$ to its outgoing neighbor in $\vec{P}$, and if $v = s$, the packet originated at $s$ first goes to the outgoing neighbor of $s$ in $\vec{Q}_1$.

**Additional Rules:** For each edge $\{c, b\}$ removed from $\mathcal{G}'$, we define additional rules as follows: for each arc $(u, c) \in \vec{Q}_2$, let $\pi_c(u) = b$ and for each arc $(b, v) \in \vec{Q}_1$, let $\pi_b(c) = v$.
**Failover Rules:** When applying all above rules, if the outgoing neighbor $u$ of the current rule at the node $v$ is not available due to failures, then use another rule at $v$ having $u$ as incoming neighbor.

By Theorem 6.3, we show that the forwarding scheme defined by Definition 6.2 for the kernel pattern $\mathcal{G}'$ is 2-resilient, and we provide an example in Fig. 9 to illustrate a sketch proof of Theorem 6.3.

Theorem 6.3. *Given a kernel graph $\mathcal{G} = \left(\bigcup_{C_\ell \in C} \mathcal{G}(C_\ell)\right) \cup E^c$, we obtain its kernel pattern $\mathcal{G}'$ by replacing each gadget with its pattern. Then the forwarding scheme in Def. 6.2 is 2-resilient on $\mathcal{G}'$.*

Proof. A sketch of this proof is illustrated in Fig. 9. First, we can obtain a *reduced* kernel pattern $\mathcal{G}^* \subseteq \mathcal{G}'$ according to Observation 1, which is a planar graph and does not need the additional rules defined in Def. 6.2. Thus, for two arbitrary failures $F \subseteq E(\mathcal{G}^*)$, which can be empty or $|F| = 1$, the graph $\mathcal{G}^* \setminus F \subseteq \mathcal{G}^*$ is also a planar graph. By using concepts of planar graphs [12], we define the *boundary* of $\mathcal{G}^* \setminus F$ as the boundary of the *outer (unbounded) face* on a planar drawing of $\mathcal{G}^* \setminus F$, which is a *closed walk* on the outer face from $s$ to $t$ and then back to $s$. For example, $\vec{Q}_1 \cup \vec{Q}_2$ in Fig. 9 denotes the boundary of $\mathcal{G}^*$. Now, we can verify that the forwarding scheme by Def. 6.2 implies a walk from $s$ to $t$ by following the boundary of $\mathcal{G}^* \setminus F$ for any two failures $F \subseteq E(\mathcal{G}^*)$, if $s-t$ is connected in $\mathcal{G}^* \setminus F$, which is hence 2-resilient on a reduced kernel pattern $\mathcal{G}^*$.

For each pattern $\mathcal{G}'(C_\ell)$ as Fig. 7a, its edge (path) $\{c, b\}$ (e.g., $\{c_3, b_3\}$ in Fig. 9) is omitted in $\mathcal{G}^*$. Next, we show that additional rules in Def. 6.2 can compensate missing $\{c, b\}$ in $\mathcal{G}^*$. Let $e_1$ be the edge $\{a, b\}$ (resp., $\{c, d\}$) in $\mathcal{G}'(C_\ell)$ and let $e_2 \notin E(\mathcal{G}'(C_\ell))$ be an edge of $Q_2$ on the right side of the port $d$ of $\mathcal{G}'(C_\ell)$ (resp., of $Q_1$ on the left side of the port $a$ of $\mathcal{G}'(C_\ell)$), but not approaching the adjacent pattern yet. After ignoring $\{c, b\}$ in $\mathcal{G}^*$, if $F = \{e_1, e_2\}$, then $s-t$ can be disconnected in $\mathcal{G}^* \setminus F$ but $s-t$ is still connected in $\mathcal{G}' \setminus F$. For example, in Fig. 9, if an edge $e_1 = \{a_3, b_3\}$ and another edge $e_2 \in E(d_3 Q_2 c_5)$ failed, then $s-t$ is not connected in $\mathcal{G}^* \setminus F$ anymore without employing additional forwarding rules, contradicting the $s-t$ connectivity in $\mathcal{G}' \setminus F$. After applying the additional rules for each missing edge $\{c, b\}$, which introduces a directed path (edge) from $c$ to $b$ in $\mathcal{G}'(C_\ell)$ (e.g., $(c_3, b_3)$ in Fig. 9), $s$ and $t$ can always communicate if $s-t$ is connected in $\mathcal{G}' \setminus F$. However, these directed $c-b$ paths introduced by additional rules are never used if $s-t$ is connected in $\mathcal{G}^* \setminus F$, and only one of them can be visited when $s-t$ is disconnected in $\mathcal{G}^* \setminus F$. Thus, additional rules cannot incur any loop in $\mathcal{G}' \setminus F$ when combined with other rules defined in $\mathcal{G}^*$. □

Finally, after obtaining a 2-resilient forwarding scheme for kernel pattern $\mathcal{G}'$, we show that it can be transferred to another 2-resilient forwarding scheme for the kernel graph $\mathcal{G}$ of $G$.

Theorem 6.4. *Given a kernel graph $\mathcal{G} = \left(\bigcup_{C_\ell \in C} \mathcal{G}(C_\ell)\right) \cup E^c$ and a kernel pattern $\mathcal{G}' = \left(\bigcup_{C_\ell \in C} \mathcal{G}'(C_\ell)\right) \cup E^c$, where each $\mathcal{G}'(C_\ell)$ is a pattern of $\mathcal{G}(C_\ell)$, then a 2-resilient forwarding scheme for $\mathcal{G}'$ computed according to Definition 6.2 can be mapped to a 2-resilient forwarding scheme for $\mathcal{G}$.*

Proof. We recall that a pattern is generated from its gadget by splitting nodes. Reversely, merging two sub-paths at a split node do not merge their edges. Given a forwarding scheme $\Pi'$ for $\mathcal{G}'$ defined by Definition 6.2, we can obtain a corresponding forwarding scheme $\Pi$ for $\mathcal{G}$ by one-to-one mapping these directed paths, e.g., $\vec{Q}_1, \vec{Q}_2$, and directed cycles $\vec{f}_i$ from $\mathcal{G}'$ to $\mathcal{G}$, s.t., a set of edge-disjoint (directed) paths (cycles) in $\mathcal{G}'$ also implies a set of edge-disjoint (directed) paths (cycles) in $\mathcal{G}$. Since $\Pi'$ is 2-resilient in $\mathcal{G}'$, $\Pi$ is also 2-resilient in $\mathcal{G}$. □

We can now cast our algorithmic result as the following theorem, along with its run-time:

Theorem 6.5. *Given a graph $G = (V, E)$ with the pair of source-destination $(s, t)$, where $|V| = n$ and $|E| = m$, a 2-resilient forwarding scheme can be computed correctly within a run-time of $O(n \cdot m)$.*

Proof. First, by Claims 1 and 2, the problem is restricted to a version, where $s-t$ is exactly 2-edge-connected and 2-node-connected in $G$. By Lemma 6.1, we can compute a kernel graph $\mathcal{G} \subseteq G$, which is a subgraph of $G$, s.t., if $s-t$ is connected in $G \setminus F$, then $s-t$ is also connected in $\mathcal{G} \setminus F$. Finally, Theorem 6.4 implies that a 2-resilient forwarding scheme for $(s, t)$ can be attained in the kernel graph $\mathcal{G}$.

In the following, we give a formal analysis of the run-time. We first note that the edge-connectivity, the edge-disjoint paths and the max-flow between $s$ and $t$ can be computed in $O(m)$ by the Ford–Fulkerson algorithm [10] when the $s-t$ edge-connectivity in $G$ is $O(1)$, and all $s-t$ bridges (resp., articulation points) of $G$ can be computed in $O(m+n)$ [4, 35]. The run-time of Claims 1 and 2 is $O(n \cdot m)$, dominated by finding two node-disjoint $s-t$ paths $P_1$ and $P_2$ for each subproblem. To compute $E^c$, for each edge $e$ of $P_1$, we can check the $s-t$ connectivity in $G \setminus \{e\}$ and then compute all $s-t$ bridges in $G \setminus \{e\}$, which needs $O\left(n \cdot m + n^2\right)$ $(m \geq n)$ in total, since each $\{e_1, e_2\} \in E^c$ must be on $P_1$ and $P_2$ respectively. There are at most $n$ components $C_\ell$ regardless of the $O(n)$ subproblems divided by Claim 2, and a gadget in each $C_\ell$ can be obtained in $O(m)$ by computing a max-flow, implying all gadgets being computable in $O(nm)$. The 2-resilient forwarding scheme can be obtained by traveling each directed path (cycle) according to Def. 6.2, bounded by $O(m)$. Thus, the total run-time is bounded by $O(n \cdot m)$. □

## 7 CONCLUSIONS AND FUTURE WORK

We present a tight characterization of fast failover routing, where 2-resilience is possible with pre-installed static routing rules on general graphs, but we show that 3-resilience is impossible.

We believe that our work opens several interesting avenues for future research. In particular, it remains to explore the resilience of our algorithms in more specific failure scenarios, both analytically and empirically. It would also be interesting to explore the use of our approach for randomized failover routing and to implement decentralized control planes that compute the corresponding failover tables.

## ACKNOWLEDGMENTS

Research supported by the Vienna Science and Technology Fund (WWTF), grant 10.47379/ICT19045 (WHATIF), 2020-2024.

## REFERENCES

[1] Mohammad Alizadeh, Albert G. Greenberg, David A. Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. 2010. Data center TCP (DCTCP). In *SIGCOMM*. ACM.

[2] Anand Bhalgat, Ramesh Hariharan, Telikepalli Kavitha, and Debmalya Panigrahi. 2008. Fast edge splitting and Edmonds' arborescence construction for unweighted graphs. In *SODA*.

[3] Prosenjit Bose, Pat Morin, Ivan Stojmenovic, and Jorge Urrutia. 1999. Routing with guaranteed delivery in ad hoc wireless networks. In *DIAL-M*. ACM, 48–55.

[4] Massimo Cairo, Shahbaz Khan, Romeo Rizzi, Sebastian S. Schmidt, Alexandru I. Tomescu, and Elia C. Zirondelli. 2021. A simplified algorithm computing all s-t bridges and articulation points. *Discret. Appl. Math.* 305 (2021), 103–108.

[5] Marco Chiesa, Andrei V. Gurtov, Aleksander Madry, Slobodan Mitrovic, Ilya Nikolaevskiy, Michael Schapira, and Scott Shenker. 2016. On the Resiliency of Randomized Routing Against Multiple Edge Failures. In *ICALP*.

[6] Marco Chiesa, Andrzej Kamisinski, Jacek Rak, Gábor Rétvári, and Stefan Schmid. 2021. A Survey of Fast-Recovery Mechanisms in Packet-Switched Networks. *IEEE Commun. Surv. Tutorials* 23, 2 (2021), 1253–1301.

[7] Marco Chiesa, Ilya Nikolaevskiy, Slobodan Mitrovic, Andrei V. Gurtov, Aleksander Madry, Michael Schapira, and Scott Shenker. 2017. On the Resiliency of Static Forwarding Tables. *IEEE/ACM Trans. Netw.* 25, 2 (2017), 1133–1146.

[8] Marco Chiesa, Roshan Sedar, Gianni Antichi, Michael Borokhovich, Andrzej Kamisinski, Georgios Nikolaidis, and Stefan Schmid. 2021. Fast ReRoute on Programmable Switches. *IEEE/ACM Trans. Netw.* 29, 2 (2021), 637–650.

[9] Cisco. 2017. Configuring BGP PIC Edge and Core for IP and MPLS.

[10] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. *Introduction to Algorithms, Third Edition* (3rd ed.). The MIT Press.

[11] Fabien de Montgolfier, Mauricio Soto, and Laurent Viennot. 2011. Treewidth and Hyperbolicity of the Internet. In *NCA*. IEEE Computer Society, 25–32.

[12] Reinhard Diestel. 2012. *Graph Theory, 4th Edition*. Graduate texts in mathematics, Vol. 173. Springer.

[13] Joan Feigenbaum, Brighten Godfrey, Aurojit Panda, Michael Schapira, Scott Shenker, and Ankit Singla. 2012. Brief announcement: on the resilience of routing tables. In *PODC*. ACM.

[14] Klaus-Tycho Foerster, Juho Hirvonen, Yvonne-Anne Pignolet, Stefan Schmid, and Gilles Trédan. 2020. Brief Announcement: What Can(Not) Be Perfectly Rerouted Locally. In *DISC*. 46:1–46:3.

[15] Klaus-Tycho Foerster, Andrzej Kamisinski, Yvonne-Anne Pignolet, Stefan Schmid, and Gilles Trédan. 2021. Grafting Arborescences for Extra Resilience of Fast Rerouting Schemes. In *INFOCOM*. IEEE, 1–10.

[16] Klaus-Tycho Foerster, Andrzej Kamisinski, Yvonne-Anne Pignolet, Stefan Schmid, and Gilles Trédan. 2022. Improved Fast Rerouting Using Postprocessing. *IEEE Trans. Dependable Secur. Comput.* 19, 1 (2022), 537–550.

[17] Klaus-Tycho Foerster, Juho Hirvonen Yvonne-Anne Pignolet, Stefan Schmid, and Gilles Trédan. 2022. On the Price of Locality in Static Fast Rerouting. In *DSN*. IEEE.

[18] Klaus-Tycho Foerster, Juho Hirvonen, Yvonne Anne Pignolet, Stefan Schmid, and Gilles Trédan. 2021. On the Feasibility of Perfect Resilience with Local Fast Failover. In *Symposium on Algorithmic Principles of Computer Systems (APOCS)*.

[19] Klaus-Tycho Foerster, Yvonne Anne Pignolet, Stefan Schmid, and Gilles Trédan. 2019. CASA: Congestion and Stretch Aware Static Fast Rerouting. In *INFOCOM*. IEEE.

[20] Pierre François, Clarence Filsfils, John Evans, and Olivier Bonaventure. 2005. Achieving sub-second IGP convergence in large IP networks. *ACM CCR* 35, 3 (2005), 35–44.

[21] Phillipa Gill, Navendu Jain, and Nachiappan Nagappan. 2011. Understanding network failures in data centers: measurement, analysis, and implications. In *SIGCOMM*. ACM.

[22] Roberto Grossi, Andrea Marino, and Luca Versari. 2018. Efficient Algorithms for Listing k Disjoint st-Paths in Graphs. In *LATIN 2018: Theoretical Informatics*, Michael A. Bender, Martín Farach-Colton, and Miguel A. Mosteiro (Eds.). Springer International Publishing, Cham, 544–557.

[23] ISO. 2002. Intermediate Ststem-to-Intermediate System (IS-IS) Routing Protocol. ISO/IEC 10589.

[24] Aubin Jarry. 2013. Fast reroute paths algorithms. *Telecommunication Systems* 52, 2 (2013), 881–888.

[25] Andrzej Kamisiński. 2018. Evolution of IP Fast-Reroute Strategies. In *RNDM*. IEEE.

[26] Evangelos Kranakis, Harvinder Singh, and Jorge Urrutia. 1999. Compass routing on geometric networks. In *CCCG*.

[27] Kin Wah Kwong, Lixin Gao, Roch Guérin, and Zhi-Li Zhang. 2011. On the Feasibility and Efficacy of Protection Routing in IP Networks. *IEEE/ACM Trans. Netw.* 19, 5 (2011), 1543–1556.

[28] Ka-Cheong Leung, Victor O. K. Li, and Daiqin Yang. 2007. An Overview of Packet Reordering in Transmission Control Protocol (TCP): Problems, Solutions, and Challenges. *IEEE Trans. Parallel Distributed Syst.* 18, 4 (2007), 522–535.

[29] Junda Liu, Aurojit Panda, Ankit Singla, Brighten Godfrey, Michael Schapira, and Scott Shenker. 2013. Ensuring Connectivity via Data Plane Mechanisms. In *NSDI*.

[30] John Moy. 1998. OSPF Version 2. *RFC* 2328 (1998), 1–244.

[31] Ping Pan, George Swallow, and Alia Atlas. 2005. Fast Reroute Extensions to RSVP-TE for LSP Tunnels. *RFC* 4090 (2005), 1–38.

[32] J. Papán, P. Segeč, M. Moravčík, M. Kontšek, L. Mikuš, and J. Uramová. 2018. Overview of IP Fast Reroute Solutions. In *ICETA*.

[33] Oliver Schweiger, Klaus-Tycho Foerster, and Stefan Schmid. 2021. Improving the Resilience of Fast Failover Routing: TREE (Tree Routing to Extend Edge disjoint paths). In *ANCS*. ACM, 1–7.

[34] Switch Specification 1.3.1. 2013. OpenFlow. In *https://opennetworking.org/wp-content/uploads/2013/04/openflow-spec-v1.3.1.pdf*.

[35] Robert Endre Tarjan. 1974. A Note on Finding the Bridges of a Graph. *Inf. Process. Lett.* 2, 6 (1974), 160–161.

[36] Balajee Vamanan, Jahangir Hasan, and T. N. Vijaykumar. 2012. Deadline-aware datacenter tcp (D2TCP). In *SIGCOMM*. ACM.

[37] Baohua Yang, Junda Liu, Scott Shenker, Jun Li, and Kai Zheng. 2014. Keep Forwarding: Towards k-link failure resilient routing. In *INFOCOM*. IEEE.

[38] David Zats, Tathagata Das, Prashanth Mohan, Dhruba Borthakur, and Randy H. Katz. 2012. DeTail: reducing the flow completion time tail in datacenter networks. In *SIGCOMM*. ACM.