FBR: Dynamic Memory-Aware Fast Rerouting

Nicklas S. Johansen, *Aalborg University, Denmark*
Lasse B. Kær, *Aalborg University, Denmark*
Andreas L. Madsen, *Aalborg University, Denmark*
Kristian Ø. Nielsen, *Aalborg University, Denmark*
Stefan Schmid, *TU Berlin, Germany and Univ. of Vienna, Austria*
Jiří Srba, *Aalborg University, Denmark*
Rasmus G. Tollund, *Aalborg University, Denmark*

***Abstract*—Modern internet communication networks, such as MPLS networks, provide fast rerouting mechanisms in the data plane in order to quickly react to link failures and hence become more dependable. However, fast rerouting requires additional memory for the conditional failover rules—a scarce and expensive resource. We initiate the study of memory-aware fast rerouting mechanisms and present Forward-Backward Routing (FBR), a dynamic rerouting approach with a provably high resilience to multiple link failures, which accounts for memory constraints on the routers. FBR relies on backtracking search along the routing paths via packet header modifications, and in our experimental evaluation on a wide range of ISP topologies, it outperforms state-of-the-art solutions while using less memory.**

***Index Terms*—memory-aware routing, fast reroute, MPLS**

I. Introduction

With the increasing scale of communication networks, link failures become more frequent [1], [2], [3]. Accordingly, modern communication networks provide fast rerouting (FRR) mechanisms to quickly mask such failures and ensure connectivity along alternative routes.

Fast rerouting mechanisms are implemented in the data plane, which typically allows to react to failures orders of magnitude faster compared to the control plane: fast rerouting in the data plane relies only on local failure information and avoids reconvergence [4], [5], [6]. At the same time, the limited local information also renders the design of fast rerouting mechanisms challenging, especially under multiple link failures, and the underlying algorithmic challenges have recently been intensively studied [7], [8], [9].

Fast rerouting mechanisms in the data plane also inherently come with overheads: routers need to store conditional failover rules which define the alternative forwarding behavior in case incident links are failed. This is problematic, as the forwarding rules of a router are typically stored in its Ternary Content Addressable Memory (TCAM), which is an expensive and power-hungry resource [10]. Interestingly, the memory overheads of fast rerouting mechanisms has received much less attention in the literature so far.

This paper initiates the study of fast rerouting mechanisms based exclusively on header modifications while accounting for the router memory constraints. Our main contribution is a new memory-aware algorithm, called Forward-Backward Routing (FBR), that generates highly resilient data planes without exceeding the given memory limit of the routers. FBR relies on header rewriting (a.k.a. dynamic rerouting, which is necessary for high resilience [11]). The input to FBR is a prioritized list of primary and backup paths, which FBR encodes using smart backtracking rules in case of link failures, allowing it to efficiently explore multiple paths to the destination (without necessarily backtracking to the source as in the state-of-the-art [12], [13]). We also propose two methods to generate the backup paths as an input to FBR and prove that one of these methods provides $(k-1)$-resilience on any k-connected network, while the other one achieves a remarkable performance on real-world network topologies. Our empirical benchmark-comparison on over 200 ISP topologies shows that FBR is attractive in terms of failure resilience, memory usage as well as the expected number of hops.

II. Context and Related Work

Fast rerouting mechanisms have already been studied extensively (see e.g. the survey [14]). We now provide an overview of the existing approaches, including those based on header-modification and the MPLS [15] technology. We also introduce abbreviations for ease of reference later in this paper.

RSVP-FN [16] is the industry standard fast rerouting mechanism used in MPLS networks. In particular we will compare to the RSVP facility node protection mechanism [16] utilizing MPLS label stacking to create bypass tunnels for protecting failed links and routers. **R-MPLS** [17] (Recursive MPLS protection) is a recent link protection extension to RSVP-FN, providing additional resilience through recursion. It can be employed on top of any existing MPLS data plane.

Plinko [18] is a data plane generation approach that achieves *perfect resilience*: packets always reach their destination as long as the underlying network remains connected. Plinko relies on dynamic rerouting guided by header rewriting because static perfect resilience [14] (without header modification) is not achievable [11]. Plinko stores link failure information in the packet header which causes an exponential explosion in the number of forwarding rules.

KF stands for the keep forwarding approach [6] without header modification. Routers are grouped based on their distance to destination and KF-traversal aims to bring the packet closer to the destination, unless this is impossible due to a failed link, in which case the packet circulates among the routers in the same group.

A general and powerful set of techniques to design resilient data planes is to decompose the network into arc-

disjoint arborescences [9]. **B-CA** stands for basic circular arborescence approach [9] and relies on circular switching among the arborescences: it starts by routing along the first arborescence and in case of a failed link it bounces to reroute along the next arborescence etc. **GFT-CA** [7] presents an improved static circular arborescence algorithm called *grafting* which first creates partial edge-disjoint arborescences. These arborescences are extended to directed acyclic graphs (DAGs) by adding additional edges; an idea proposed in [19], [20]. GFT-CA forwards along an outgoing edge that reduces the distance to the egress router the most. We also implemented a memory-aware improvement to the CA strategy (referred to as **E-CA**) by applying dynamic header modifications and employing non-disjoint arborescences, and hence improving the resilience while at the same time being able to limit the required memory.

Finally, there are approaches like [21] that store the failure information in the header and locally recompute new routes for each failure scenario, or run a dataplane re-convergence algorithm by reversing the directions of links upon failures [22], while modifying the routing tables. These approaches are orthogonal to ours as they require path recomputations and/or modification of the routing tables. In case of link failures, our method instead allows us to switch between a number of predefined paths (similarly as in MPLS fast reroute), which simplifies network operations and allows us to support traffic engineering goals (e.g., congestion avoidance, latency minimization, etc.).

III. Network Model and Metrics

In this section we formalize the network model, assuming that each packet has a header that can be matched and modified by forwarding rules at the routers.

Definition 1: A *network topology* is a graph $G = (V, E)$ where V is a finite set of *routers* and $E \subseteq V \times V$ is a finite set of *links*, assuming that all links are bidirectional, i.e. if $(v, v') \in E$ then also $(v', v) \in E$.

Figure 1a shows a network topology where the links are depicted by solid undirected edges. A data plane forwarding function for each router and an incoming label returns the set of forwarding rules, each with a priority, next-hop router as well as the outgoing label that replaces the incoming label.

Definition 2: For a network topology $G = (V, E)$, a *data plane* is a tuple (τ, L) where L is a finite set of header *labels* and $\tau : V \times L \to 2^{\mathbb{N} \times V \times L}$ is the global forwarding table.

Figure 1c provides an example of a forwarding table. For every router $v \in V$ and an incoming label $\ell \in L$, the function $\tau(v, \ell)$ returns a (possibly empty) set of *forwarding rules*. A forwarding rule is a triple (p, v', ℓ') where p is the priority of the entry, v' is the next-hop s.t. either $(v, v') \in E$ or $v = v'$ (loopback interface), and $\ell' \in L$ is the outgoing header label.

A *failure scenario* $F \subseteq E$ is a subset of the edges that represents the failed links; we assume that if $(v, v') \in F$ then also $(v', v) \in F$ and by $|F|$ we denote the number of bidirections links in F. A forwarding rule is *active* if the link to the next-hop is not failed (does not belong to F). For a given failure scenario, we define the active forwarding table as a forwarding table that only contains highest-priority rules with an active next-hop.

Definition 3: For a given data plane (τ, L) and a failure scenario $F \subseteq E$, we define the *active forwarding table* $\tau_F : V \times L \to 2^{V \times L}$ as $\tau_F(v, \ell) = \{(v', \ell') \mid (p, v', \ell') \in \tau(v, \ell), (v, v') \notin F, p = p_{min}\}$, where $p_{min} = \min\{p \mid (p, v', \ell') \in \tau(v, \ell), (v, v') \notin F\}$.

Consider the forwarding table from Figure 1c. In case of no link failures, we have $\tau_\emptyset(in, \ell_1) = \{(a, \ell_1)\}$, meaning that a packet with the label ℓ_1 arriving to the router in is forwarded to the router a without modifying the label. In case that the link between in and a fails, i.e. $F = \{(in, a), (a, in)\}$, we get that $\tau_F(in, \ell_1) = \{(in, \ell_2)\}$, implying that the packet is recirculated at the router in with the modified label ℓ_2. As the next step, the packet is forwarded to the router b with the label ℓ_2 as $\tau_F(in, \ell_2) = \{(b, \ell_2)\}$.

Definition 4: A *demand* in a network is a triple $(v_{in}, v_{out}, \ell) \in V \times V \times L$ describing a packet flow from an ingress router v_{in} to the egress router v_{out} with an initial label ℓ comprising the packet header. We denote by D the set of all demands in a given network.

The demands determine where packets enter and exit the network. A packet enters the network at a router via a link from outside the network. We denote this outside link as $(\hat{v}, v_{in})$, where $\hat{v}$ is a placeholder for the outside router. Additionally, each demand has an egress router v_{out}. If the packet reaches this router it is delivered and leaves the network. We can now formalize the notion of a trace for a given demand.

Definition 5: Let $(v_0, v_n, \ell_0) \in D$ be a demand, F be a failure scenario, and τ_F be the active forwarding table. A *trace* is a (finite or infinite) maximal sequence of link-label pairs $((\hat{v}, v_0), \ell_0)((v_0, v_1), \ell_1) \cdots ((v_{i-1}, v_i), \ell_i) \cdots$, where $((v_{i-1}, v_i), \ell_i) \in V \times V \times L$ and $(v_i, \ell_i) \in \tau_F(v_{i-1}, \ell_{i-1})$ for each $i > 0$.

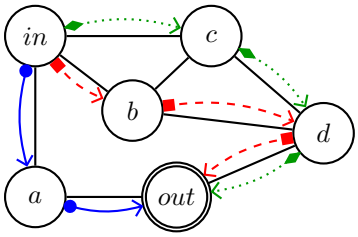
Figure 1d shows a trace for the failure scenario $F = \{(a, out), (out, a), (in, b), (b, in)\}$.

Given a network $G = (V, E)$, a failure scenario F and a demand (v_{in}, v_{out}, ℓ), we say that v_{in} *delivers* to v_{out} if for *all* possible traces, $t = ((\hat{v}, v_{in}), \ell) \cdots$, there exists a link-label pair $((v, v_{out}), \ell') \in t$, i.e. for all nondeterministic choices (rules with equal priority) the packet arrives to v_{out}.

a) Data Plane Metrics: Our aim is to construct a memory-bounded data plane with fast rerouting that maximizes the possibility of delivering packets for all demands in the network. We say that a router is *connected* to another router if there exists a path between them.

The *connectivity* metric returns for a given failure scenario the number of successfully delivered demands divided by the number of demands that are connected under the failure scenario. For a network topology $G = (V, E)$, a failure scenario F and a set of demands D, the measure $\mathit{connectivity}(\tau_F, D)$ is hence given as

$$\frac{\#(v_1, v_2, \ell) \in D \text{ where } v_1 \text{ delivers to } v_2 \text{ in } \tau_F}{\#(v_1, v_2, \ell) \in D \text{ where } v_1 \text{ is connected to } v_2 \text{ in } G_F}$$



(a) Network with three forwarding paths from in to out

iteration	path	label	visualisation
1	in-a-out	ℓ_1	solid/●
2	in-b-d-out	ℓ_2	dashed/■
3	in-c-d-out	ℓ_3	dotted/♦

(b) Three forwarding paths found by Algorithm 2

Router	Label	Priority	Router	Label
in	ℓ_1	1	a	ℓ_1
		2	in	ℓ_2
	ℓ_2	1	b	ℓ_2
		2	in	ℓ_3
	ℓ_3	1	c	ℓ_3
a	ℓ_1	1	out	ℓ_1
		2	a	ℓ_2
	ℓ_2	1	in	ℓ_2
b	ℓ_2	1	d	ℓ_2
		2	b	ℓ_3
	ℓ_3	1	in	ℓ_3
c	ℓ_3	1	d	ℓ_3
d	ℓ_2	1	out	ℓ_2
		2	d	ℓ_3
	ℓ_3	1	out	ℓ_3

(c) Data plane created by Algorithm 1 with paths from Figure 1b

Step	Router	Label	Forwarding rules
1	in	ℓ_1	$(1, a, \ell_1)$, $(2, in, \ell_2)$
2	a	ℓ_1	~~$(1, out, \ell_1)$~~, $(2, a, \ell_2)$
3	a	ℓ_2	$(1, in, \ell_2)$
4	in	ℓ_2	~~$(1, b, \ell_2)$~~, $(2, in, \ell_3)$
5	in	ℓ_3	$(1, c, \ell_3)$
6	c	ℓ_3	$(1, d, \ell_3)$
7	d	ℓ_3	$(1, out, \ell_3)$
8	out	ℓ_3	

Corresponding trace:
$((\hat{v}, in), \ell_1)$ $((in, a), \ell_1)$ $((a, a), \ell_2)$ $((a, in), \ell_2)$ $((in, in), \ell_3)$ $((in, c), \ell_3)$ $((c, d), \ell_3)$ $((d, out), \ell_3)$

(d) Forwarding steps for the demand (in, out, ℓ_1) in the failure scenario $F = \{(a, out), (out, a), (in, b), (b, in)\}$; the striked-out rules are not active in F

Fig. 1: Example network, protection paths, generated data plane and a trace in a given routing scenario

where G_F is the subgraph induced by removing the edges corresponding to the links in F. Clearly, $connectivity(\tau_F, D)$ is a rational number between 0 and 1, where 1 means that the data plane can for the failure scenario F deliver every demand where it is physically possible.

We assume that each failure scenario F has a probability p_F of occurring (which allows us to account for independent link and node failures as well as shared risk group failures) such that $\sum_{F \subseteq E} p_F = 1$. We now introduce the notion of connectedness of a data plane under failures, defined by

$$connectedness(\tau, D) = \sum_{F \subseteq E} p_F \cdot connectivity(\tau_F, D)$$

which aggregates connectivity over all failure scenarios and weights them by their probability. Hence the influence of failure scenarios that are more common is higher. The *connectedness* measure is again a number between 0 and 1 and a higher connectedness implies a more reliable data plane.

Finally, we define the *memory usage* of a data plane τ at a router v by

$$M(\tau, v) = \sum_{\ell \in L} |\tau(v, \ell)|$$

as the number of forwarding rules stored at the router v for each of the incomming labels.

A *memory limit* is then a function $m : V \to \mathbb{N}$ that determines the maximum number of forwarding rules that each router can store. Our objective is to synthetise a data plane with the highest connectedness within the memory limit.

Problem 1: Given a network topology $G = (V, E)$, a set of demands D and a memory limit m, create a data plane (τ, L) that maximises $connectedness(\tau, D)$ while at the same time $M(\tau, v) \leq m(v)$ for all $v \in V$.

IV. Forward-Backward Routing

We now present our memory-aware data plane generation algorithm, Forward-Backward Routing (FBR). The core idea of FBR is to utilize the observation that edges that were just used for forwarding are active. FBR assumes as an input a prioritized sequence of alternative paths that can deliver packets for each demand. FBR then follows the first path in the sequence and in case that at a certain router v a link on the path is down, it tries (using the loopback interface) to switch to an alternative path (assuming that there is an alternative path that intersects at the router v). If no alternative path is found at the router v, the packet is returned back to where it came from (knowing that this link is active) and the same process recursively repeats at the source router of the link. In the worst case, the packet can arrive all the way back to the ingress router where an alternative path is guaranteed to

Algorithm 1: Forward-Backward Routing (FBR)

input : A network topology $G = (V, E)$, a set of demands D, a path function $P : D \to \mathcal{P}_G$, and a memory limit $m : V \to \mathbb{N}$.
output: A data plane (τ, L).

```
1  Initialize τ to an empty forwarding table
   // Create a label for each path and
   demand
2  L := {ℓ_i^d | d ∈ D, 1 ≤ i ≤ |P(d)|}
3  for i := 1 to max{|P(d)| | d ∈ D} do
4      foreach d = (v_0, v_n, ℓ_0) ∈ D do
5          if i ≤ |P(d)| then
6              p := P(d)(i)
7              if MEM(τ, p, v) ≤ m(v) for all v ∈ V then
8                  ENCODE-PATH(d, p, i)
9  return τ, L
10 Function ENCODE-PATH(d, p, i)
11     foreach (v, v') ∈ p do
12         τ(v, ℓ_i^d) := {(1, v', ℓ_i^d)}
13         if i < |P(d)| then
14             p' := P(d)(i + 1)
15             τ(v, ℓ_i^d) := τ(v, ℓ_i^d) ∪ {(2, v, ℓ_{i+1}^d)}
16             if v does not appear in p' and ∃(v'', v) ∈ p
                 then
17                 τ(v, ℓ_{i+1}^d) := τ(v, ℓ_{i+1}^d) ∪ {(1, v'', ℓ_{i+1}^d)}
18 Function MEM(τ, p, v)
       // Computes memory usage at router
       v after path p is encoded into τ
```

be selected. It is important to evenly distribute diverse paths among all the demands and FBR iteratively attempts to add a path for each demand in a round robin fashion as long as it does not exceed the routers' memory. The algorithm relies hence on a centralized controller.

FBR is formalized in the pseudocode of Algorithm 1. The input to the algorithm is, apart from the topology $G = (V, E)$, set of demands D and a memory limit m, also a function P that for every demand $d \in D$ returns a prioritized sequence of paths in G so that these paths can deliver the demand d. The set of all such prioritized sequences is denoted by $\mathcal{P}_G$ and formally for each demand d we have a function $P(d) : \{1, 2, \ldots, n\} \to E^*$ that encodes n alternative paths such that $P(d)(i)$ for $1 \leq i \leq n$ is the sequence of edges in the i'th path for the demand d. We require that each router occurs in each path at most once. By $|P(d)| = n$ we denote the number of alternative paths for the demand d and we assume that $|P(d)| \geq 1$ for all $d \in D$. We will later discuss how these paths can be effectively generated in order to optimize the performance of the data plane.

Algorithm 1 adds paths for the demands in a round robin fashion, starting with the first path of each demand, then the second path of each demand etc. Paths are encoded into the data plane only if their addition does not cause any router to exceed its memory limit (see line 7). The function `Mem` simulates the encoding of a path to the routing table (according to the function `Encode-Path`) in order to compute the number of rules for each router after the path gets installed.

If the path fits into the memory, each edge in the path is added to the routing table at line 12. The switch to the next path, in case of failure, is encoded at line 15 using the loop-back interface. Finally, a backtracking to the parent node is added at line 17. To avoid forwarding loops, a backtracking rule is added only if the router is not part of the next path.

Example 1: Figure 1 shows how FBR encoded three paths (depicted in Figure 1b) in a network with a single demand (in, out, ℓ_1). We assume here a sufficient memory limit to install all three paths. The first processed path (encoded using the label ℓ_1) is *in-a-out*. The forwarding rules from in to out through a are added with priority 1. For all routers on the path, except the egress router, a bounce rule to the next path is added with priority 2 (meaning that it is active only if the outgoing interface on the first path is down). Because the next path, *in-b-d-out* (encoded using the label ℓ_2), does not use the router a, FBR creates a backtracking rule from a to in using the label ℓ_2. FBR then creates rules for the second path *in-b-d-out*. Similarly, the third path is encoded using the label ℓ_3. As the third path *in-c-d-out* is the last one, no further bounce/backtracking rules are added. Figure 1d shows the forwarding behaviour and a trace for the failure scenario with failed links (a, out) and (in, b) (in both directions).

A. Theoretical Properties of FBR

We shall first argue that FBR does not create any forwarding loops, meaning that after finitely many steps every packet is either delivered or dropped.

Theorem 1: FBR algorithm generates a loop-free data plane.

Proof (sketch): A packet with a given label can use at most twice as many hops (first moving forward and then backward) as the length of the path it encodes. Moreover, labels can be swapped only in acyclic way and if the last path cannot deliver, the packet is dropped. ■

Next, we shall argue that FBR can achieve high resilience.

Definition 6: Let $G = (V, E)$ be a network topology, D be a set of demands. A forwarding table τ is k-resilient if $\mathit{connectivity}(\tau_F, D) = 1$ for all $F \subseteq E$ where $|F| \leq k$.

Theorem 2: Let G be a k-connected network topology with demands D and memory limit $m(v) \geq (3k - 2) \cdot |D|$ for all $v \in V$. There is a path generation method for which the FBR algorithm achieves $(k - 1)$-resilience.

Proof (sketch): As the network is k-connected, we can find k edge-disjoint paths between all pairs of routers. If Algorithm 1 is called with such a sequence of paths, interleaved with the empty paths (this guarantees a full backtracking to the ingress router in case a failed edge on a path), we know that the data plane eventually delivers any packet as we assume

Algorithm 2: Semi-disjoint path generation

input : A network topology $G = (V, E)$, a set of demands D, and a number of iterations N
output: A path function $P : D \to \mathcal{P}_G$ used by Alg. 1

```
1  Let W : D × E → ℕ, initially set to 0 for all entries
2  Let P : D → P_G, initialized to empty path sequences
3  for N iterations do
4      for d = (v_0, v_n, l_0) ∈ D do
5          p := ShortestPath(G, v_0, v_n, W)
6          for e ∈ p do
7              W(d, e) := W(d, e) · 2 + 1
8          if p is not in P(d) then
9              j := |P(d)| + 1    // Get next index
10             P(d)(j) := p       // Add new path
11 return P
```

that only $k-1$ edges can fail (implying that there is at least one connected path left). The number of installed entries for the k paths of each demand is $2(k-1)+k = 3k-2$. ■

B. Semi-Disjoint Paths Generation

We now propose a practically efficient path generation method for the use in the FBR algorithm, also used in the experimental evaluation. We produce semi-disjoint paths for all demands in a network topology, providing a good balance between the disjointness of the paths while minimizing the number of hops. The pseudocode is given in Algorithm 2. In each iteration starting at line 3, we add a new path to each demand. The path is identified as the shortest path between v_0 and v_n with respect to the dynamic edge weight function W. In case of multiple shortest paths w.r.t. W, we select one with the fewest number of hops. Initially, all weights are set to 0, and when a path p is selected, the weights of the edges on this path are increased. Updating the weights penalizes paths that use edges already in other paths, thus increasing the diversity of the paths.

V. Implementation and Evaluation

We implement our FBR algorithm in the data plane toolkit *MPLS-Kit* [23], which is also used to generate failure scenarios, to simulate the packet forwarding, and to compute the connectedness of the generated data planes. The data plane generation for Plinko (with resilience level 4), RSVP-FN and R-MPLS are already available in *MPLS-Kit*. The algorithms KF, B-CA, E-CA and GFT-CA are implemented by ourselves. The source code and reproducibility package is available on GitHub at https://github.com/Ragusaen/p8.

In our benchmark we use 259 ISP topologies from the Topology Zoo [24]. For each topology, we create a demand that starts in each router and ends in a random egress router. We assume that all routers have the same memory limit and we consider a uniform link failure probability of $p_f = 0.001$. Hence the probability of a failure scenario $F \subseteq E$ is given by $p_F = p_f^{|F|} \cdot (1-p_f)^{|E|-|F|}$. where $|E|$ and $|F|$ is the number bidirectional links in E and F, respectively. As the simulation of all possible failure scenarios is infeasible (the number grows exponentially in the number of links), we run our experiments with randomly selected 1000 failure scenarios with up to four failed links and normalize the sum of probabilities of the selected failure scenarios. All experiments are executed on a compute-cluster running Ubuntu version 18.04.5 on an AMD Opteron(tm) Processor 6376.

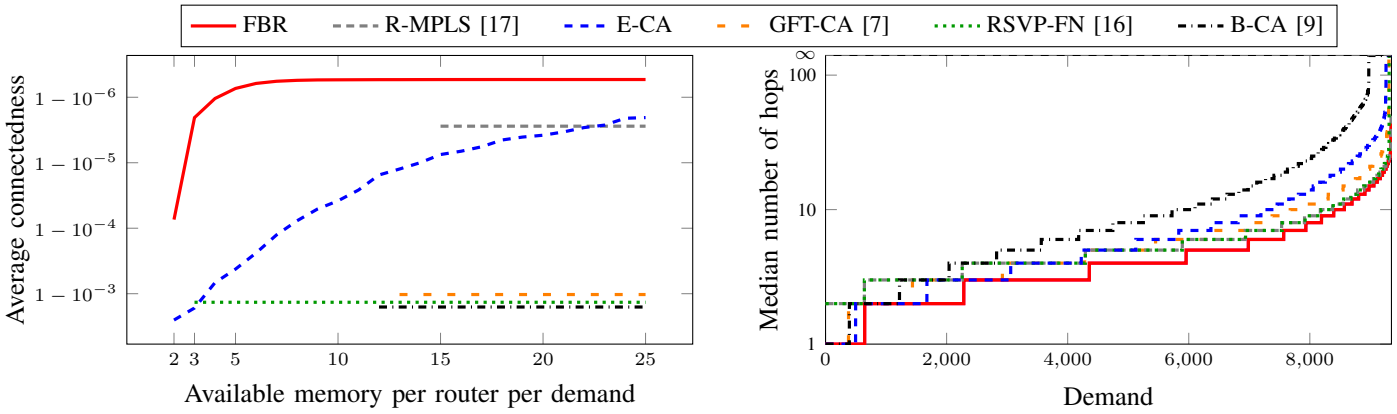
Experimental Evaluation. KF and Plinko (that achieves the optimal resilience by construction) use over 500 and 7500 forwarding rules on each router per demand, respectively. As these numbers are unreasonably high for any real-world application (e.g. the industry standard RSVP-FN uses only up to 3 rules per router and demand), we exclude these approaches from the comparison plots.

Figure 2a shows how the average connectedness of the algorithms (on y-axis) increases with the amount of available memory per router and demand (on x-axis). The approaches of B-CA, RSVP-FN and GFT-CA are not memory-aware (hence the horizontal lines in the plot) and achieve similar average connectedness of around $1 - 10^3$. Noticeably, RSVP-FN has the lowest memory requirements (installing at most three rules per router and demand) while the other two approaches install over 10 rules in order to achieve a similar performance. With a slightly higher memory requirements the R-MPLS protocol achieves a significantly higher connectedness but similarly to the other protocols it is not memory-aware and may not necessarily fit into the available memory (depending on the hardware). We extended the basic CA algorithms into a memory-aware variant, called E-CA, which considerably improves the connectedness while increasing the available memory (and eventually even exceeds R-MPLS). Finally, the plot shows that our FBR algorithm produces a data plane that provides the highest connectedness (very close to the one achieved by Plinko) for any considered memory limit and achieves its best performance already with 4-5 installed rules per router and demand. Already with the same memory requirement as the industry standard RSVP-FN, it provides the highest connectedness among all competing approaches.

Figure 2b displays the median number of hops (y-axis) for an algorithm to deliver a packet over all demands on all networks (x-axis). Here the memory limit for E-CA and FBR is set to 4. We can notice that the arborescence based approaches B-CA, GFT-CA and E-CA use the largest number of hops to deliver a packet. Both RSVP-FN and R-MPLS have a very similar performance in the number of hops but still they use one additional hop, in the median case, to deliver the packet compared to our FBR approach.

VI. Conclusion

We initiated the study of memory-aware fast rerouting and presented an efficient mechanism, Forward-Backward Routing (FBR), which guarantees a high resilience. On real-world ISP network topologies, FBR outperforms the state-of-the-art, with regard to failure resilience, while using only 2–3 rules per



(a) Average connectedness vs. maximum memory usage

(b) Median number of hops among all failure scenarios

Fig. 2: Experimental results (in Figure 2b E-CA and FBR have a memory limit of 4).

router per demand. Compared to related work, FBR further uses the fewest number of hops to deliver a packet.

In our future work, we plan to improve the path generation method for FBR to account for congestion and to include unused links more effectively in the paths.

Acknowledgements. Research supported by the Vienna Science and Technology Fund (WWTF) project ICT19-045 and by the DFF project QASNET.

References

[1] N. Shelly, B. Tschaen, K. Förster, M. A. Chang, T. Benson, and L. Vanbever, "Destroying networks for fun (and profit)," in *14th ACM Workshop on Hot Topics in Networks*, J. de Oliveira, J. Smith, K. J. Argyraki, and P. A. Levis, Eds. ACM, 2015, pp. 6:1–6:7.

[2] S. Iyer, S. Bhattacharyya, N. Taft, and C. Diot, "An approach to alleviate link overload as observed on an IP backbone," in *IEEE INFOCOM 2003*. IEEE Computer Society, 2003, pp. 406–416.

[3] P. Gill, N. Jain, and N. Nagappan, "Understanding network failures in data centers: measurement, analysis, and implications," in *ACM SIGCOMM 2011*, S. Keshav, J. Liebeherr, J. W. Byers, and J. C. Mogul, Eds. ACM, 2011, pp. 350–361.

[4] C. Jiang, S. G. Rao, and M. Tawarmalani, "PCF: provably resilient flexible routing," in *SIGCOMM '20*, H. Schulzrinne and V. Misra, Eds. ACM, 2020, pp. 139–153.

[5] H. H. Liu, S. Kandula, R. Mahajan, M. Zhang, and D. Gelernter, "Traffic engineering with forward fault correction," in *ACM SIGCOMM 2014*, F. E. Bustamante, Y. C. Hu, A. Krishnamurthy, and S. Ratnasamy, Eds. ACM, 2014, pp. 527–538.

[6] B. Yang, J. Liu, S. Shenker, J. Li, and K. Zheng, "Keep forwarding: Towards k-link failure resilient routing," in *INFOCOM 2014*. IEEE, 2014, pp. 1617–1625.

[7] K.-T. Foerster, A. Kamisiński, Y.-A. Pignolet, S. Schmid, and G. Tredan, "Grafting arborescences for extra resilience of fast rerouting schemes," in *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*, 2021, pp. 1–10.

[8] K. Foerster, J. Hirvonen, Y. Pignolet, S. Schmid, and G. Trédan, "On the feasibility of perfect resilience with local fast failover," in *2nd Symposium on Algorithmic Principles of Computer Systems, APOCS 2020, Virtual Conference, January 13, 2021*, M. Schapira, Ed. SIAM, 2021, pp. 55–69.

[9] M. Chiesa, I. Nikolaevskiy, S. Mitrovic, A. V. Gurtov, A. Madry, M. Schapira, and S. Shenker, "On the resiliency of static forwarding tables," *IEEE/ACM Trans. Netw.*, vol. 25, no. 2, pp. 1133–1146, 2017.

[10] A. X. Liu, C. R. Meiners, and E. Torng, "TCAM razor: a systematic approach towards minimizing packet classifiers in tcams," *IEEE/ACM Trans. Netw.*, vol. 18, no. 2, pp. 490–500, 2010.

[11] J. Feigenbaum, B. Godfrey, A. Panda, M. Schapira, S. Shenker, and A. Singla, "Brief announcement: on the resilience of routing tables," in *ACM Symposium on Principles of Distributed Computing, PODC '12, Funchal, Madeira, Portugal, July 16-18, 2012*, D. Kowalski and A. Panconesi, Eds. ACM, 2012, pp. 237–238.

[12] K.-T. Foerster, Y.-A. Pignolet, S. Schmid, and G. Tredan, "Casa: congestion and stretch aware static fast rerouting," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2019, pp. 469–477.

[13] P. M. Mohan, T. Truong-Huu, and M. Gurusamy, "Tcam-aware local rerouting for fast and efficient failure recovery in software defined networks," in *2015 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2015, pp. 1–6.

[14] M. Chiesa, A. Kamisinski, J. Rak, G. Rétvári, and S. Schmid, "A survey of fast-recovery mechanisms in packet-switched networks," *IEEE Commun. Surv. Tutorials*, vol. 23, no. 2, pp. 1253–1301, 2021.

[15] *Introduction to MPLS*, https://www.cisco.com/c/dam/global/fr_ca/training-events/pdfs/Intro_to_mpls.pdf, visited: 19/05/2020.

[16] P. Pan, G. Swallow, and A. Atlas, "Fast reroute extensions to RSVP-TE for LSP tunnels," in *Request for Comments (RFC) 4090*, 2005. [Online]. Available: https://doi.org/10.17487/RFC4090

[17] S. Schmid, M. K. Schou, J. Srba, and J. Vanerio, "Recursive Protection for Highly Dependable Networks," Patent application PA 202200385. Filed by Aalborg University, April, 2022.

[18] B. E. Stephens, A. L. Cox, and S. Rixner, "Scalable multi-failure fast failover via forwarding table compression," in *Symposium on SDN Research, SOSR 2016*, B. Godfrey and M. Casado, Eds. ACM, 2016, p. 9.

[19] S. Ray, R. Guérin, K. W. Kwong, and R. Sofia, "Always acyclic distributed path computation," *IEEE/ACM Trans. Netw.*, vol. 18, no. 1, pp. 307–319, 2010.

[20] J. Liu, B. Yang, S. Shenker, and M. Schapira, "Data-driven network connectivity," in *Tenth ACM Workshop on Hot Topics in Networks (HotNets-X), HOTNETS*, H. Balakrishnan, D. Katabi, A. Akella, and I. Stoica, Eds. ACM, 2011, p. 8.

[21] K. Lakshminarayanan, M. Caesar, M. Rangan, T. Anderson, S. Shenker, and I. Stoica, "Achieving convergence-free routing using failure-carrying packets," in *Proceedings of the 2007 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, ser. SIGCOMM '07. ACM, 2007, p. 241–252.

[22] J. Liu, A. Panda, A. Singla, B. Godfrey, M. Schapira, and S. Shenker, "Ensuring connectivity via data plane mechanisms," in *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*. USENIX Association, 2013, pp. 113–126.

[23] J. Vanerio, S. Schmid, M. Schou, and J. Srba, "MPLS-Kit: MPLS Data Plane Tooklit," 2022. [Online]. Available: https://github.com/juartinv/mplskit

[24] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The internet topology zoo," *IEEE Journal on Selected Areas in Communications*, vol. 29, no. 9, pp. 1765–1775, 2011.