The Quest for Resilient (Static) Forwarding Tables

Marco Chiesa*, Ilya Nikolaevskiy†, Slobodan Mitrović‡, Aurojit Panda§, Andrei Gurtov†¶,
Aleksander Mądry‖, Michael Schapira**, Scott Shenker§††
*Université Catholique de Louvain, Belgium
†Department of Computer Science, Aalto University, Finland
‡École Polytechnique Fédérale de Lausanne, Switzerland
§University of California, Berkeley, US
¶Helsinki Institute for Information Technology HIIT, Finland
‖Massachusetts Institute of Technology, US
**Hebrew University of Jerusalem, Israel
††International Computer Science Institute, US

***Abstract*—Fast Reroute (FRR) and other forms of immediate failover have long been used to recover from certain classes of failures without invoking the network control plane. While the set of such techniques is growing, the level of resiliency to failures that this approach can provide is not adequately understood. We embark upon a systematic algorithmic study of the resiliency of immediate failover in a variety of models (with/without packet marking/duplication, etc.). We leverage our findings to devise new schemes for immediate failover and show, both theoretically and experimentally, that these outperform existing approaches.**

I. Introduction

Routing on the Internet (both within an organizational network and between such networks) typically involves computing a set of *destination-based* routing tables (i.e., tables that map the destination IP address of a packet to an outgoing link). Whenever a link or node fails, routing tables are recomputed by invoking the routing protocol to run again (or having it run periodically, independent of failures). This produces well-formed routing tables, but results in relatively long outages after failures as the protocol is recomputing routes.

As critical applications began to rely on the Internet, such outages became unacceptable. As a result, "fast failover" techniques have been employed to facilitate immediate recovery from failures.[1] The most well-known of these is Fast Reroute in MPLS where, upon a link failure, packets are sent along a precomputed alternate path without waiting for the global recomputation of routes [1]. This, and other similar forms of fast failover thus enable rapid response to failures but are limited to the set of precomputed alternate paths.

The fundamental question is, then, how resilient can forwarding tables be? That is, how many link failures can failover routing tolerate before connectivity is interrupted (i.e., packets are trapped in a forwarding loop, or hit a dead end) without invoking the control plane? The answer to this question depends on (1) the number of failures the forwarding scheme should withstand (e.g., resiliency to multiple simultaneous link failures is crucial in overlay networks over optical backbone networks [2], [3]), (2) the structural properties of the network graph (e.g., in terms of connectivity), and (3) the limitations imposed on the routing scheme (e.g., with/without packet marking).

The goal of this paper is to shed light on the theoretical guarantees that are achievable by *any* failover forwarding table and leverage these insights to devise better, more resilient, immediate failover schemes. We only consider link failures, not vertex failures (which are not always detectable by neighboring routers, so such fast resilient routing techniques may not apply).

We distinguish between *static* routing tables and *dynamic* routing tables. While dynamically and adaptively changing the forwarding tables at a router in response to link failures can achieve high resiliency (see, e.g., link reversal [6] and [7], [8]), current routing protocols and infrastructure do not support such stateful failover routing. Our focus is hence on static, OpenFlow-like [9], failover routing, where a router/switch matches packet headers to forwarding rules.

We first observe that designing static routing tables that are robust to multiple failures is a relatively simple task when the forwarding decisions can rely on both the source and destination of the packet [10]. Unfortunately, the number of forwarding entries grows quadratically with the size of the network. We seek "scalable" static failover schemes that rely on limited, locally-available information, specifically: the destination address, the packet's incoming link, and the set of non-failed links incident to the router. We note that per-incoming-link destination-based forwarding tables are a necessity as destination-based routing alone is unable to achieve robustness against even a single link failure [11], (and, moreover, entails computationally hard challenges [3], [11]–[13]).

We investigate three models of "scalable" static failover routing: basic routing, routing with packet-header rewriting, and routing with packet-duplication. We present, for each of these three models, new immediate failover schemes with

[1]By "immediately", we mean that there is no control plane delay, but the fast failover can only happen after (a) the failure is detected and (b) the router can update its routing table to use the backup route. These delays depend on the technology used for failure detection and table management, so the resulting delays can vary substantially, but typically are much less than the time it takes for the control plane to reconverge.

TABLE I: Summary of the resiliency of routing tables for arbitrary topologies.

	Per-destination	Per-incoming port	Packet header rewriting	Packet duplication	Resiliency
Static Routing Tables	X				$\times$ impossible for any $k \geq 2$
	X	X			✓for any $k \leq 5$
	X	X	X		✓with $> k$ bits [4], [5], k-bits [2], $\lg k$-bits, and 3-bits
	X	X		X	✓at most $2k-1$ packets
Dynamic Routing Tables	X	X	X		✓1-bit in the packet header [6]–[8]

provably improved resiliency over past approaches. Our results are summarized in Table I. We experimentally compare these schemes. Our findings show that a high level of resiliency is achievable even with no/little rewriting of packet headers.

Basic (`BSC`) failover routing (2nd row): each packet is matched to an outgoing port only on the destination address, the incoming link, and the set of non-failed links. Past work [14], [15] (1) achieved guaranteed robustness against *only* a single link/node failure [16]–[21], (2) achieved robustness against $\lfloor \frac{k}{2} - 1 \rfloor$ link failures for k-connected networks [2], and (3) proved that resiliency to any set of link failures [17] cannot be achieved.

We show that resiliency to multiple link failures can be accomplished even through basic failover routing. We show how to generate static forwarding tables that are resilient to any $k-1$ failures for a broad variety of k-connected networks, including full-meshes, generalized hypercubes (proposed as datacenter topologies in [22]), and Clos networks (today's datacenter topologies). Motivated by these results, we make the following general conjecture:

Conjecture: For any k-connected graph, basic failover routing can be resilient to any $k-1$ failures.

We take a first step in this direction by proving that for any network that is k-connected for $k \leq 5$, the above statement holds. We present several negative results, e.g., for natural forms of basic failover routing.

Failover routing with packet-header rewriting (`HDR`, 3rd row): a node has an ability to rewrite any bit in the packet header. Clearly, if it is possible to store an arbitrary amount of information in the packet header, perfect resiliency can be achieved by collecting information about every failed link that a packet encounters [4], [5]. Such approaches are not feasibly deployable in modern-day networks as the header of a packet may be too large for today's routing tables. More recent results show that for any k-connected network, k bits are sufficient to compute forwarding tables that are robust to $(k-1)$ link failures [2]. Our focus is thus on failover routing schemes that involve only minimal rewriting of bits in the packet header, or even no rewriting whatsoever. We show that the ability to modify at most *three* bits suffices to provide the same level of resiliency.

Failover routing with packet duplication (`DPL`, 4th row): a node has an ability to duplicate a packet (without rewriting its header) and send the copies through deterministically chosen outgoing links. We show how to compute, for any k-connected network, resilient routing tables that do not create more than $2f$ duplicates of a packet, where f is the number of failed links hit by a packet. (So, in particular, if there is no link failure, no packet duplication occurs.)

Comparing the three schemes: We experimentally evaluate the three proposed schemes both in terms of resiliency and in terms of path lengths (stretch). Our main conclusions are that (1) our positive results for the basic failover technique (which does not involve bit rewriting in the packet header) come with an average stretch of only 10%, and (2) for any k-connected network, the ability to rewrite only $\log(k)$ bits is sufficient to be resilient against $k-1$ link failures with only a small stretch compared to the technique that uses k bits. Hence, a high level of resiliency is achievable with little/no bit rewriting and without the overheads associated with packet duplication.

A. Organization

In Section II, we introduce our routing model and formally state the STATIC-ROUTING-RESILIENCY problem. In Section III, we summarize routing techniques that will be leveraged throughout the whole paper. Section IV is devoted to our main resiliency results for basic routing. Then, in Section V and Section VI, we show that robustness to $(k-1)$ link failures, where k is the connectivity of a graph, can be achieved through the rewriting of 3 bits in the packet header, or if the packet can be duplicated, respectively. We present the experimental evaluation of our failover scheme in Section VII. We present certain impossibility results in Section VIII and discuss related work in Section IX. We conclude in Section X. All missing details appear in [10].

II. MODEL

We represent the network as an undirected multigraph $G = (V, E)$, where each router in the network is modeled by a vertex in V and each link between two routers is modeled by an undirected edge in the multiset E. We denote an (undirected) edge between x and y by $\{x, y\}$. Each vertex v routes packets according to a *forwarding function* that matches an incoming packet to a sequence of forwarding actions.

Packet *matching* is performed according to the set of active (non-failed) edges incident at v, the incoming edge, and any information stored in the packet header (e.g., destination label, extra bits), which are all information that are *locally* available at a vertex. Since our focus is on per-destination forwarding functions, we assume that there exists a unique destination $d \in V$ to which every other vertex wishes to send packets and, therefore, that the destination label is not included is the header of a packet. Forwarding *actions* consist of routing packets through an outgoing edge, rewriting some bits in the packet header, and creating duplicates of a packet.

In this paper we consider three different types of forwarding functions. We first explore a particularly simple forwarding function, which we call *basic* routing (`BSC`). In basic routing (Section IV) a packet is forwarded to a specific outgoing edge based only on the incoming port and the set of active outgoing edges. The other two forwarding functions, which are generalization of `BSC` are the following ones: *header-rewriting* routing, in which a vertex rewrites the header of a packet, and *duplication* routing, in which a vertex creates copies of a packet. Basic routing is a special case of each of these forwarding functions. We present the formal definitions of the header-rewriting and duplication routing models in Section V and Section VI, respectively.

The STATIC-ROUTING-RESILIENCY problem. Given a graph G, a forwarding function f is *c-resilient* if, for each vertex $v \in V$, a packet originated at v and routed according to f reaches its destination d as long as at most c edges fail and there still exists a path between v and d. The input of the SRR problem is a graph G, a destination $d \in V$, and an integer $c > 0$, and the goal is to compute a set of resilient forwarding functions that is c-resilient. In this paper we investigate the relationship between the resiliency that can be achieved by static routing tables and the connectivity of a graph. We say a graph is *k-connected* if there exist k edge-disjoint paths between any pair of vertices in the graph. We now intuitively introduce our main routing techniques.

III. GENERAL ROUTING TECHNIQUES

As in [2], we leverage a well-known result from graph theory [23], which allows us to decompose any k-connected graph in a set of k directed spanning trees (rooted at the same vertex) such that no pair of spanning trees shares an edge in the same direction. As an example, consider Fig.1, in which each pair of vertices is connected by two edges (ignore the red crosses) and four arc-disjoint arborescences `Blue`, `Orange`, `Red`, and `Green` are depicted by colored arrows. Efficient fast algorithms to compute such arborescences can be found in [24]. We now show the main techniques that we use to route along these arborescences.

Arborescence-based routing. Throughout the paper, unless specified otherwise, we let $\mathcal{T} = \{T_1, \ldots, T_k\}$ denote a set of k arc-disjoint spanning arborescences of G rooted at a common destination vertex. All our routing techniques are based on a decomposition of G into $\mathcal{T}$. We say that a packet is routed

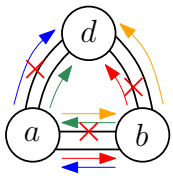

Fig. 1: A 4-connected graph with 4 arc-disjoint arborescences.

in *canonical* mode along an arborescence T if a packet is routed through the unique directed path of T towards the destination. If packet hits a failed edge at vertex v along T, it is processed by v (e.g., duplication, header-rewriting) according to the capabilities of a specific forwarding function and it is rerouted along a different arborescence. We call such routing technique *arborescence-based* routing. One crucial decision that must be taken is the next arborescence to be used after a packet hits a failed edge. In this paper, we propose two natural choices that represent the building blocks of all our forwarding functions. When a packet is routed along T_i and it hits a failed arc (v, u), we consider the following two possible actions:

- **Reroute along the next available arborescence**, e.g., reroute along $T_{next} = T_{(i+1) \mod k}$. Observe that, if the outgoing arc belonging to T_{next} failed, we forward along the next arborescence, i.e. $T_{(i+2) \mod k}$, and so on.
- **Bounce on the reversed arborescence**, i.e., we reroute along the arborescence T_{next} that contains arc (u, v).

We say that a forwarding function is a *circular-arborescence* routing if each vertex can arbitrarily choose the first arborescence to route a packet and, for each $T_i \in \mathcal{T}$, we use canonical routing until a packet hits a failed edge, in which case we reroute along the next available arborescence. We will show an example in Section IV-A.

In the next sections, we show how it is possible to achieve different degrees of resiliency by using our general routing techniques and different forwarding functions (i.e., basic, packet-header-rewriting, and packet-duplication).

IV. BASIC ROUTING

In this section we show how to achieve $(k-1)$-resiliency for any arbitrary k-connected graph, with $k \leq 5$, using basic forwarding functions (`BSC`), which map an incoming edge and the set of active edges incident at v to an outgoing edge. This striking result demonstrates that resiliency to multiple failures can surprisingly be achieved even without invoking the control plane and without adding any additional information in the header of a packet. We also show that for several k-connected graphs (e.g., Clos networks, full-mesh), with arbitrary positive k, there exist sets of $(k-1)$-resilient forwarding functions.

A. Arbitrary Graphs

We first show that circular-arborescence routing is not sufficient to achieve 3-resiliency. Consider the example in Fig. 1 with 3 vertices a, b, and c and 6 edges (depicted as black lines) $e^A_{a,b} = \{a, b\}$, $e^F_{a,b} = \{a, b\}$, $e^A_{a,d} = \{a, d\}$, $e^F_{a,d} = \{a, d\}$, $e^A_{b,d} = \{b, d\}$, and $e^F_{b,d} = \{b, d\}$, where A

stands for "active" edge and F for "failed" edge (depicted with a red cross over them). Four arc-disjoint arborescences $\mathcal{T} = \{\texttt{Blue}, \texttt{Orange}, \texttt{Red}, \texttt{Green}\}$ are depicted by colored arrows. Let $< \texttt{Blue}, \texttt{Orange}, \texttt{Red}, \texttt{Green} >$ be a circular ordering of the arborescences in $\mathcal{T}$. We now describe how a packet p originated at a is forwarded throughout the graph using a circular-arborescence routing. Since $e^F_{a,d}$ is failed, p cannot be routed along the Blue arborescence. It is then rerouted through Orange, which also contains a failed edge $e^F_{a,b}$ incident at a. As a consequence, p is forwarded to b through the Red arborescence. At this point, p cannot be forwarded to d because $e^F_{b,d}$, which belongs to Red, failed. It is then rerouted through Green, which also contains a failed edge $e^F_{a,b}$ incident at b. Hence, p is rerouted again through Blue, which leads p to the initial state—a forwarding loop.

An intuitive explanation is the following one. Since an edge might be shared by two distinct arborescences, a packet may hit the same failed edge both when it is routed along the first arborescence and when it is routed along the second arborescence. As a consequence, even $\frac{k}{2}$ failed edges may suffice to let a packet be rerouted along the same initial vertex and initial arborescence, creating a forwarding loop. Our first positive result shows that a forwarding loop cannot arise in 2- and 3-connected graphs if circular-arborescence routing is adopted.

Theorem 1. *For any k-connected graph, with $k = 2, 3$, any circular-arborescence routing is $(k-1)$-resilient. In addition, the number of switches between trees is at most 4.*

Proof sketch. Consider a 2-connected graph $G = (V, E)$, two arc-disjoint arborescences T_1 and T_2 of G, and an arbitrary failed edge $e = \{u, v\} \in E$. W.l.o.g, T_1 is the first arborescence that is used to route a packet p. When p hits e (w.l.o.g, at u), p cannot hit e in the opposite direction along T_2. In fact, this would mean that there exists a directed path from u to v that belongs to T_2 and that (v, u) is contained in T_2—a directed cycle. A similar, but more involved argument, holds for the 3-connected case (see [10]). □

4-connected graphs. Let us look again at the graph in Fig. 1. It is not hard to see that a different circular ordering of the arborescences (i.e., $< \texttt{Blue}, \texttt{Green}, \texttt{Orange}, \texttt{Red} >$) would be robust to any three failures. However, our first result shows that in general circular-arborescence routing is not sufficient to achieve $(k-1)$-resiliency, for any $k \geq 4$.

Theorem 2. *There exists a 4-connected graph such that, given a set of k arc-disjoint arborescences, there does not exist any 3-resilient circular-arborescence forwarding function.*

In the experimental section we show that a naive choice of the arc-disjoint arborescences will very likely cause a forwarding loop.

To overcome this, we first introduce the following key lemma, in which we show how to construct four arc-disjoint arborescences such that some of them do not share edges with each other. Then, we compute a circular-arborescence routing that is 3-resilient based on these arborescences.

Lemma 3. *For any $2k$-connected graph G, with $k \geq 1$, and any vertex $d \in V$, there exist $2k$ arc-disjoint arborescences $T_1, \ldots, T_{2k}$ rooted at d such that $T_1, \ldots, T_k$ do not share edges with each other and $T_{k+1}, \ldots, T_{2k}$ do not share edges with each other.*

A similar lemma holds also for any $(2k+1)$-connected graph, where $k \geq 0$ (see [10]). The following theorem states that a circular ordering $< T_1, \ldots, T_4 >$ of the arborescences constructed as in Lemma 3 is a 3-resilient circular-arborescence routing. We will make use of the general case of Lemma 3 in Sect. VI.

Theorem 4. *For any 4-connected graph, there exists a circular-arborescence routing that is 3-resilient. In addition, the number of switches between trees is at most $2f$, where f is the number of failed edges.*

Constrained topologies and 5-connected graphs. For several graph topologies that are common in Internet routing or datacenter networks, we show that $(k-1)$-resilient forwarding functions can be computed in polynomial time. The list of graphs that admit $(k-1)$-resilient forwarding functions encompasses cliques, complete bipartite graphs, generalized hypercubes, Clos networks, and grids [22], [25], [26]. We refer the reader to [10] for further details. We also show how to compute 4-resilient forwarding functions in polynomial time for 5-connected graphs (cf. [10]).

V. Packet header rewriting

We devote this section to algorithms that rewrite a very limited number of bits in the packet header in order to achieve $(k-1)$-resiliency, and present two such algorithms, approached in mutually different ways. The first algorithm uses $\lceil \log k \rceil$ and the second one uses only 3 bits in the packet header. As depicted in Table I, concerning the number of bits allocated in the packet header, both algorithms substantially improve upon the previous work. Our experiments, that we present in Section VII, suggest that the algorithm that uses only $\lceil \log k \rceil$ bits is of a high practical relevance as well.

The good arborescence property. Before we delve into the details of how the first algorithm works, we introduce the notion of *good arborescences*. This concept might be of an independent interest as it provides a novel insight into the structure of failed links and the corresponding set of arc-disjoint arborescences. Intuitively, given a set of failed edges, an arborescence T is a good arborescence if for every packet that is routed along it, either it reaches the destination vertex along T or, when it is bounced on a failed edge, it will reach the destination vertex without any further interruption.

More formally, let $a = (u, v)$ be a link such that (u, v) belongs to arborescences T_i and (v, u) belongs to T_j. We say that a is a *well-bouncing* arc if by bouncing from T_i to T_j on the failed link $\{u, v\}$ the packet will reach d via routing along T_j without any further interruption. Having well-bouncing

notion in hands, we say that an arborescence T_i is a good arborescence if every failed arc of T_i is well-bouncing. Note that by the definition if T_i does not contain any failed link then it is still good.

Each non-shared failed link affects only one arborescence. So, as long as there are at most $k-1$ failed links, non-shared failed links do not fully disrupt the connectivity of the arborescences. On the other hand, each shared failed link affects two arborescences, but it allows the packet to bounce and hence very efficiently cope with failures. A careful analysis of the both properties allows us to show that there always exists a good arborescence. For the sake of brevity we omit the proof of that fact and refer the reader to [10] for more details.

Circular routing, bouncing and $\lceil \log k \rceil$ bits for $(k-1)$-resiliency. Algorithm HDR-LOG-K-BITS is an example of how the circular routing can be mixed with the properties of good-arborecences to obtain $(k-1)$-resiliency. We make this bond possible via bouncing and $\lceil \log k \rceil$ bits, stored as variable $currcirc$, maintained in the packet header.

Algorithm 1 Definition of HDR-LOG-K-BITS.

HDR-LOG-K-BITS: Given $\mathcal{T} = \{T_1, \ldots, T_k\}$ and d

1. Let T_i be the first tree that is used to route a packet.
2. Set $currcirc := i$.
3. Repeat until the packet is delivered to d
   a. Route along T_i until d is reached or the routing hits a failed edge.
   b. If the routing hits a failed edge a and a is shared with arborescence T_j.
      (i) If $currcirc \neq i$, let $currcirc := (currcirc + 1) \mod k + 1$ and $i := currcirc$.
      (ii) Otherwise, let $i := j$.

Recall that a handy property of a good arborescence is that by bouncing on it the packet will be delivered to the destination without any further interruption. As before, when the delivery encounter a failed link our main goal is to discover whether the arborescence it is routed along is a good one or not. As a reminder, as long as there is at least one and at most $k-1$ failed links it can be shown that there is at least one good arborescence. But, how to find such one? We employ the circular routing (the loop starting at line 3 along with the condition at line 3b(i)) to keep on looking for a good arborescence. So, once the algorithm encounter a failed link on the currently considered arborescence T_i in the circular ordering (at line 3b) there are two actions it performs. First, it checks whether the current arborescence is good, i.e. it bounces (expressed by the assignment at line 3b(ii)) and routes along the new arborescence. If T_i is good, then the bouncing will deliver the packet. Otherwise, the algorithm performs the second action – it routes along the arborescence following T_i in the circular order (as done at line 3b(i) if the corresponding condition is satisfied). Which of the two actions is taken, and the information on how to retreat back to the circular ordering after a bouncing is performed, is kept in variable $currcirc$. Name $currcirc$ stands for "current circular" and represents the index of the arborescence which in the circular ordering is currently considered. Observe that since the number of arborescences is k, we need only $\lceil \log k \rceil$ bits to store $currcirc$.

Employing even fewer bits in the packet header. Next, we show how to construct a set of $(k-1)$-resilient forwarding functions that requires only three extra bits in the packet header.

Consider the circular routing algorithm with the following twist, which we call HDR-3-BITS. If in the circular routing the packet hits a failed edge a of an arborescence T_i, then the packet bounces to arborescence T_j, if there is any, and continues routing along T_j. Now, if the packet hits a failed edge of T_j, then the packet is routed back to the edge a and the circular routing continues. The corresponding algorithm follows. We stress the fact that variable i is not stored in the packet header but is inferred from the incoming arc on which the packet is received.

Algorithm 2 Definition of HDR-3-BITS.

HDR-3-BITS: Given $\mathcal{T} = \{T_1, \ldots, T_k\}$ and d

1. Set $i := 1$.
2. Repeat until the packet is delivered to d
   1. Route along T_i until d is reached or the routing hits a failed edge.
   2. If the routing hits a failed edge a and a is shared with arborescence T_j, $i \neq j$.
      (a) Bounce and route along T_j. (As we discuss in the sequel, the routing scheme employed after bouncing deviates from the one used before the bouncing.)
      (b) If the routing hits a failed edge in T_j, route back to the edge a.
   3. Set $i := (i+1) \mod k + 1$

As we show in the sequel, in case there are at most $k-1$ failed edges then the described routing scheme delivers the packet to d. However, there are a few questions that we should resolve in order to implement this scheme in our routing model: first, after bouncing on a failed edge a and hitting a new failed edge, how one can route the packet back to a; and, second, how we keep track of whether the circular routing or the one after bouncing is in use. Now, both questions could be easily answered if the packet stores the path it is routed over, which in the worst case could require "many" extra bits. On the other hand, as we have been discussing in the introduction, our aim is to provide a routing scheme that uses a very few bits, which we do in this section.

Backtracking: A routing and its inverse. Essentially, the first question can be cast as a task of devising a routing scheme $R(T)$, for a given arborescence T, which has its inverse. Let our hypothetical scheme $R(T)$ route the packet along edges $a_1, a_2, \ldots, a_t, a_{t+1}$ in that order. Then, the inverse routing

scheme $R^{-1}(T)$ would route a packet received along a_{t+1} through edges $a_t, a_{t-1}, \ldots, a_1$ in that order. We choose $R(T)$ to be a DFS traversal of T starting at d. For the sake of the traversal, we disregard the orientation of the edges of T, as shown in Fig. 2.

Note that we use canonical mode (which does not have an inverse) for routing packets along the arborescences that are chosen in the circular order. Only once the packet bounces to arborescence T, we route the packet following scheme $R(T)$, and then follow its inverse $R^{-1}(T)$ if a new failed edge is hit, as explained above.

Three extra bits suffices for $(k-1)$-resiliency. So, to put into action our routing algorithm, we use three different routing schemes. In order to distinguish which one is currently used, we store extra bits in the packet header. Those bits are used to keep the information needed to decide which routing scheme should be used. To keep track of which routing scheme is being used, out of the three aforementioned, we need two bits. Let RM be a two-bit word with the following meaning: $RM = 0$ for canonical mode; $RM = 1$ for scheme $R(T)$; and $RM = 2$ for scheme $R^{-1}(T)$.

We now motivate the usage of the third bit. Let a be the last arc the packet is routed over. Then in canonical mode, i.e. if $RM = 0$, a uniquely determines the arborescence along which the packet is routed. However, if T_i and T_j, for $i < j$, share an edge $\{x, y\}$, then the arcs (x, y) and (y, x) are in both $R(T_i)$ and in $R(T_j)$. Therefore, if $RM \neq 0$ then the information stored in RM along with a is not sufficient to determine whether the arborescence the packet is routed along is T_i or T_j. So, to keep track of whether the packet is routed along T_i or T_j we use another extra bit H. We set $H = 1$ if the packet is routed along the arborescence with higher index, i.e. along T_j, and set $H = 0$ otherwise.

Therefore, in total we need three additional bits (two for RM and one for H) to keep track of which routing scheme and which arborescence is currently in the use.

Putting good arborescences into the setting we have just developed, we can show that indeed HDR-3-BITS computes a $(k-1)$-resilient routing.

Theorem 5. *For any k-connected graph,* HDR-3-BITS *computes a set of $(k-1)$-resilient forwarding functions.*

VI. PACKET DUPLICATION

In this section we show that, for any k-connected graph G, it is always possible to compute duplication forwarding functions (DPL) that are $(k-1)$-resilient. DPL maps an incoming edge and the set of active edges incident at v to a subset of the outgoing edges at v. A packet is duplicated at v and one copy is sent to each of the edges in that set.

A naive approach would flood the whole network with copies of the same packets, i.e., each vertex creates a copy of a packet for each outgoing edge and forwards it through that edge. There are two drawbacks to this approach. First, marking packets is necessary to avoid forwarding loops. Second, at least a copy of the packet will be routed through each edge, wasting routing resources. In the following, we present an algorithm that creates a very limited number of copies of a packet and guarantees robustness against any $k-1$ edge failures.

Algorithm 3 Definition of DPL-ALGO.

1) p is first routed along T_1.
2) p is routed along the same arborescence towards the destination, unless a failed edge is hit.
3) if p hits a failed edge (x, y) along T_i, then:
   a) if $i < s$: one copy of p is created; the original packet is forwarded along T_{i+1}; the copy is forwarded along T_l, where T_l is the arborescence that contains arc (y, x).
   b) if $i = s$: $s-1$ copies of p are created; the original packet is forwarded along T_{s+1}; the j'th copy, with $1 \leq j \leq s-1$, is routed along T_{s+j+1}.
   c) if $i > s$: p is destroyed.

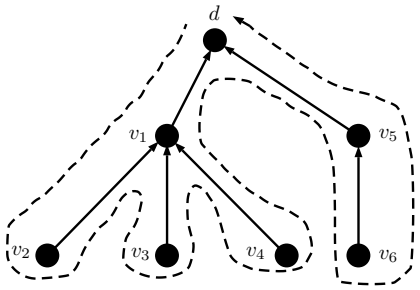

Fig. 2: Let T denote the arborescence on the figure. A DFS traversal is illustrated by the dashed line, i.e. $R(T) = dv_1v_2v_1v_3v_1v_4v_1dv_5v_6v_5d$ and $R^{-1}(T) = dv_5v_6v_5dv_1v_4v_1v_3v_1v_2v_1d$.

The general idea is to carefully combine the benefits of both circular-arborescence and bounce routing (as for HDR routing in Sect. V). Circular-arborescence routing allows us to visit each arborescence, while bouncing a packet allows us to discover well-bouncing arcs (see Sect. V for the definition of well-bouncing arcs). Bouncing packets comes at the risk of easily introducing forwarding loops as packets may be bounced between just two arborescences. Hence, we leverage our construction of arborescences from Lemma 3, which helps us to eventually hit $k-1$ distinct failed edges, and we forbid any bouncing that may create a forwarding loop. For simplicity, we assume that $k = 2s$ is even.

Let G be a $2s$-connected graph and $T_1, \ldots, T_{2s}$ be $2s$ arc-disjoint arborescences such that $T_1, \ldots, T_s$ ($T_{s+1}, \ldots, T_{2s}$) do not share edges with each other (as in Lemma 3). We define the DPL-ALGO in Alg. 3 and in the following show that it provides a set of $(2s-1)$-resilient forwarding functions.

We start by observing that each failed edge hit along the first s arborescences cannot be a well-bouncing arc, otherwise this would mean that at least a copy of a packet will reach d.

Lemma 6. *Let T_i be a good arborescence. If* DPL-ALGO *fails to deliver a packet to d, then $i > s$.*

By a counting argument (see [10]), we can leverage Lemma 6 to prove the resiliency of DPL-ALGO to $(2s-1)$ link failures and bound the amount of duplicated packets that are created.

Theorem 7. *For any $2s$-connected graph and $s \geq 1$, DPL-ALGO computes $(2s-1)$-resilient forwarding functions. In addition, the number of copies of a packet created by the algorithm is f, if $f < s$, and $2s-1$ otherwise, where f is the number of failed edges.*

Routing from any arbitrary initial arborescence. In contrast to the BSC and HDR forwarding techniques, in DPL-ALGO each vertex is forced to start routing packets on the same initial arborescence T_1. This approach has one main drawback: even in the absence of link failures, routing is constrained along an arborescence, which may not even be a shortest path arborescence. This leads to unacceptable high path lengths. We solve this issue by adding a counter with $\lceil \log(s) \rceil$ bits in the packet header. We initially set the counter to 0 and we increase it every time a packet hits a failed edge. If the counter is smaller than s, we apply step 3a from Alg. 3, otherwise, if the counter is equal to s, we apply step 3b. We do not use the counter when we route along any arborescence T_i, with $i > s$. Evaluating this version of the algorithm in Section VII, we show that it achieves high resiliency with very limited stretch. We call this algorithm DPL-LOG-K-BITS.

VII. EXPERIMENTS

We experimentally evaluate the four proposed schemes both in terms of resiliency and in terms of path lengths (stretch). Our main conclusions are that (1) our positive results for basic failover technique (which does not involve marking packets) come with an average stretch of only 10%, and (2) for any k-connected network, we show that the ability to rewrite only $\lceil \log k \rceil$ bits is sufficient to be resilient against $k-1$ link failures with only small stretch compared to the technique that uses k bits. Hence, a high level of resiliency is achievable with little/no packet rewriting of bits in the packet header and without the overheads associated with packet duplication.

First, we assess the effectiveness of the BSC-ALGO, which is based on a circular-arborescence forwarding function. Recall that BSC-ALGO is based on a special construction of a set of arc-disjoint arborescences. We show that an arbitrary set of arbitrary arc-disjoint arborescences would very likely be prone to forwarding loops.

Arbitrary arc-disjoint arborescences are not 3-resilient. We experimentally quantify the amount of routers that are no longer able to send packets to a destination vertex when circular-arborescence is used on an arbitrary set of arc-disjoint arborescences, Fig. 3. We generate 1000 different 4-connected random networks with sizes ranging from 10 to 40 vertices. For each network with N routers, we consider $320 \cdot N$ random 3-link failures scenarios. We then count the number of routers that are no longer able to reach the destination router (i.e., are trapped in a forwarding loop) in at least one failure scenario. As shown in Fig. 3, roughly 65% of the routers lost connectivity to the destination vertex in at least one failure scenario. We point out that we are only providing a lower bound as an exploration of all possible 3-link failures in large networks is computationally unfeasible. In contrast, our construction of arc-disjoint arborescences described in Lemma 3 guarantees that no pair of vertices is disconnected in any 4-connected network for any 3-link failures.

Path stretch in the absence of failures. In [2] it was shown that arc-disjoint arborescences have limited stretch with respect to shortest paths in the absence of link failures. The authors also observe that, if packet header marking is allowed by the forwarding function, a single extra bit can be used to switch to failover routing only when a packet hits a failed link. Otherwise, a packet is forwarded according to any arbitrary scheme defined by a network operator (e.g., shortest paths). We omit the results for the path stretch in the absence of failures as they are similar to the ones already obtained in [2].

Little/no bit rewriting in packet header is sufficient for high resiliency and low stretches. We use as a point of reference for our evaluation the algorithm presented in [2], which uses k bits in the packet header. We define the stretch of a routing function R as the ratio between the number of links traversed by algorithm and the number of links traversed by the algorithm in [2]. We generated 1000 different 4-connected random networks with 100 routers. For each network we look at 3200 random 3-link failures scenarios. In Fig. 4, Fig. 5, and Fig. 6, we show the cumulative distribution function of the path stretch from each source vertex to a specific destination using our four 3-resilient algorithms, i.e., BSC-ALGO, which routes packets based on a circular-arborescence forwarding function, HDR-LOG-K-BITS, which rewrites $\log(k)$ bits in the packet header, HDR-3-BITS, which rewrites only 3 bits in the packet header, and DPL-LOG-K-BITS, which rewrites $\log(k)$ bits in the packet header and possibly creates duplicates of a packet. We stress the fact that in all the depicted graphs, we only compute the stretch for those packets that actually hit at least one failed link. We first observe no rewriting of bits in the packet header (i.e., BSC-ALGO) leads to surprisingly limited average stretch, i.e., 90% of the packets have stretch smaller than 1.2. HDR-LOG-K-BITS reduces the average stretch to roughly 1.1. Not surprisingly, DPL-LOG-K-BITS can reduce stretch further as it can explore different paths in the network at the same time[2]. Finally, we observe that the path stretch in HDR-3-BITS is unacceptable when compared to the other approaches. The main reason is that packets are not directly routed through the destination vertex along an arborescence (see Sect. VI). We finally compare in Fig. 7 and Fig. 8 the performance of the three other $(k-1)$-resilient algorithms on 8-connected networks. We observe a similar trend to the one observed for 4-connected networks.

VIII. IMPOSSIBILITY RESULTS FOR BASIC ROUTING

We now show that simplified forms of failover forwarding functions are not sufficiently powerful. It is well-known that

[2]The stretch of DPL-LOG-K-BITS is computed on the first copy of a packet that reaches the destination.

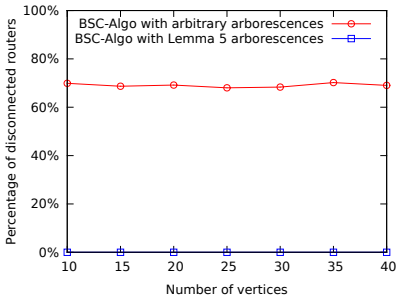

Fig. 3: 4-connected, 3 link failures.

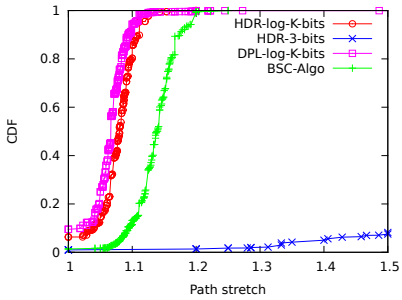

Fig. 4: 4-connected, 1 link failure.

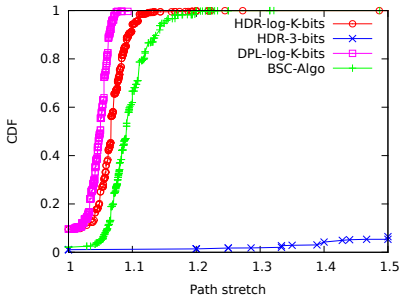

Fig. 5: 4-connected, 2 link failures.

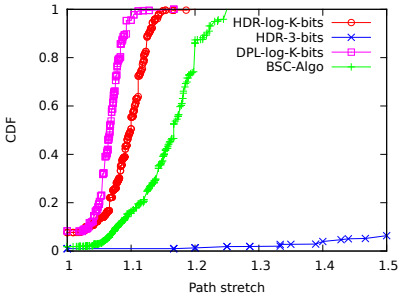

Fig. 6: 4-connected, 3 link failures.

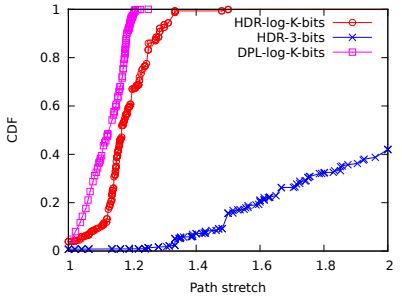

Fig. 7: 8-connected, 4 link failures.

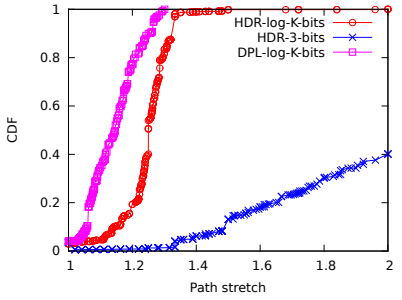

Fig. 8: 8-connected, 7 link failures.

without matching the incoming-edge it is not even possible to construct 1-resilient static forwarding functions [11]. To overcome this, [15] suggests to route packets based on a circular ordering of the edges incident at each vertex. Namely, a set of forwarding functions is *link-circular* if each vertex v routes packets based on an ordered circular sequence $< e_1, \ldots, e_l >$ of its incident edges as follows. If a packet p is received from an edge e_i, then v forwards it along e_{i+1}. If the outgoing edge e_{i+1} failed, v forwards p through e_{i+2}, and so on. We prove in [10] that this simplified forwarding functions cannot provably guarantee $(k-1)$-resiliency even for 3-connected graphs.

Theorem 8. *There is a 3-connected graph G for which no 2-resilient link-circular forwarding function exists.*

We now exploit the previous theorem to state another impossibility result, which shows that the connectivity between two vertices, i.e., the maximum amount of disjoint paths between the two vertices, does not match the resiliency guarantee for these two vertices. In other words, even if a vertex v is k-connected to the destination (but *not* the entire graph), it is not possible to guarantee that a packet originated at v will reach d when $k-1$ edges fail. Clearly, if we want to protect against $k-1$ failures a single vertex that is k-connected to d, we can safely route along its k edge-disjoint paths one after the other until the packet reaches its destination. However, if there are more vertices to be protected, it may be not possible to protect all of them. We say that a forwarding function is *strong-connectivity-resilient* if each packet that is originated by a vertex v that is k-connected to the destination d, can be routed towards the destination as long as less than k edges fail. By leveraging Theorem 8 and using a simple graph transformation (see [10]) we can show that strong-connectivity-resilient is not achievable.

Theorem 9. *There are a graph G and a destination d for which no set of strong-connectivity-resilient forwarding functions exists.*

We show that there exists a limit on the resiliency that can be attained in a k-connected graph, in which each vertex is k-connected to the destination. It was proved in [17] that resiliency against any failures that do not disconnect a sender from d, cannot be guaranteed. We claim a stronger bound.

Theorem 10. *There is a 2-connected graph for which no set of 2-resilient forwarding functions exists.*

IX. Related Work

There is a huge body of literature on related topics, and here we give only a high-level overview. We make several distinctions among the studies satisfying these requirements; the first is whether the routing algorithm can rewrite packet headers (inserting/modifying additional state). This category includes [2], [4], [27]–[37] and the general thrust of these results (with some variation) is that adding one or a few additional bits (or tunnels) can achieve 1- or 2-resiliency, whereas one can achieve $k-1$ resiliency with k bits. When one allows an unlimited list of failed node/links in the packet header, [4] and [5] deliver packets as long as the network remains connected. The next category involves solutions that do not modify the packet header, and here we can further distinguish between solutions that modify the forwarding

tables based on packet arrivals, and those that have static tables. The dynamic approaches can deliver packets whenever the network remains connected [7], [8]. Among the static approaches, some depend only on the destination address, and some also depend on the incoming port. The former are guaranteed to deliver packets under any arbitrary non-disconnecting set of failures only if the routing tables are not deterministic, otherwise, for deterministic static routing tables, not only the problem of protecting against one single failure may not admit a solution, but it is even hard to compute routing tables that maximize the number of vertices that are protected [3], [11]–[13]. The latter (i.e., per-incoming port static deterministic routing tables) exploit the incoming port of a packet to infer what links have failed. Our work belongs to this category. Previous works are limited in several aspects: the proposed heuristics have no provable failover guarantees [14], [15]; failover mechanisms have limited guaranteed resilience against *only* one single link/node failure [16]–[21] or $\lfloor \frac{k}{2} - 1 \rfloor$-resiliency for k-connected graphs [2]; routing focus limited to shortest-path-IP [14], [34], [35]; impossibility of achieving ∞-resiliency [17]. For specific topologies, works [38], [39] achieve $k - 1$ resiliency but no general methodology is described. In contrast, we show how to compute $(k - 1)$-resilient routing tables for arbitrary k-connected graphs, with $k \leq 5$ and we show that k-resiliency cannot be guaranteed for any k-connected graph with static routing tables.

X. Conclusions

We presented the Static-Routing-Resiliency problem and explored the power of static fast failover routing in a variety of models: deterministic routing, routing with packet-duplication, and routing with packet-header-rewriting. Our results suggest that even under severe restrictions on forwarding (no/little rewriting of bits in the packet header) a high-level of resiliency is achievable with negligible stretch.

XI. Acknowledgements

This research is (in part) supported by European Union's Horizon 2020 research and innovation programme under the ENDEAVOUR project (grant agreement 644960) and by Swiss National Science Foundation (grant number P1ELP2_161820).

References

[1] P. Pan, G. Swallow, and A. Atlas, "Fast Reroute Extensions to RSVP-TE for LSP Tunnels," *RFC 4090, IETF, 2005*.

[2] T. Elhourani, A. Gopalan, and S. Ramasubramanian, "IP Fast Rerouting for Multi-Link Failures," in *Proc. IEEE INFOCOM*, 2014.

[3] A. Atlas and A. Zinin, "Basic Specification for IP Fast Reroute: Loop-Free Alternates," *RFC 5286, IETF, 2008*.

[4] K. Lakshminarayanan, M. Caesar, M. Rangan, T. Anderson, S. Shenker, and I. Stoica, "Achieving convergence-free routing using failure-carrying packets," in *SIGCOMM*, 2007.

[5] B. Stephens, A. L. Cox, and S. Rixner, "Plinko: Building Provably Resilient Forwarding Tables," in *Proc. of HotNets*. ACM, 2013.

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

[7] J. Liu, A. Panda, A. Singla, B. Godfrey, M. Schapira, and S. Shenker, "Ensuring Connectivity via Data Plane Mechanisms," in *Proc. of NSDI*, 2013, pp. 113–126.

[8] J. Liu, B. Yan, S. Shenker, and M. Schapira, "Data-driven Network Connectivity," in *Proc. of HotNets*. ACM, 2011, p. 8.

[9] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling Innovation in Campus Networks," *CCR*, 2008.

[10] M. Chiesa, I. Nikolaevskiy, A. Panda, A. Gurtov, M. Schapira, and S. Shenker, "Exploring the limits of static failover routing," *CoRR*, vol. abs/1409.0034, 2014. [Online]. Available: http://arxiv.org/abs/1409.0034

[11] K.-W. Kwong, L. Gao, R. Guérin, and Z.-L. Zhang, "On the Feasibility and Efficacy of Protection Routing in IP Networks," *ToN*, 2011.

[12] M. Borokhovich and S. Schmid, "How (Not) to Shoot in Your Foot with SDN Local Fast Failover," in *OPODIS*, 2013, pp. 68–82.

[13] G. Schollmeier, J. Charzinski, A. Kirstadter, C. Reichert, K. J. Schrodi, Y. Glickman, and C. Winkler, "Improving the Resilience in IP Networks," in *Proc. HPSR*, 2003.

[14] A. Atlas and A. Zinin, "U-turn Alternates for IP/LDP Fast-Reroute," *IETF Internet draft version 03*, February 2006.

[15] B. Yang, J. Liu, S. Shenker, J. Li, and K. Zheng, "Keep Forwarding: Towards K-link Failure Resilient Routing, INFOCOM," 2014.

[16] G. Enyedi, G. Rétvári, and T. Cinkler, "A Novel Loop-free IP Fast Reroute Algorithm," in *Proc. EUNICE*, 2007.

[17] J. Feigenbaum, P. B. Godfrey, A. Panda, M. Schapira, S. Shenker, and A. Singla, "On the resilience of routing tables," in *PODC*, July 2012.

[18] S. Nelakuditi, S. Lee, Y. Yu, Z.-L. Zhang, and C.-N. Chuah, "Fast Local Rerouting for Handling Transient Link Failures," *ToN*, 2007.

[19] J. Wang and S. Nelakuditi, "IP Fast Reroute with Failure Inferencing," in *Workshop on Internet Network Management*, 2007.

[20] B. Zhang, J. Wu, and J. Bi, "RPFP: IP fast reroute with providing complete protection and without using tunnels," in *IWQoS*, 2013.

[21] Z. Zhong, S. Nelakuditi, Y. Yu, S. Lee, J. Wang, and C. nee Chuah, "Failure Inferencing based Fast Rerouting for Handling Transient Link and Node Failures," in *Proc. IEEE INFOCOM*, 2005.

[22] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu, "BCube: A High Performance, Server-centric Network Architecture for Modular Data Centers," in *Proc. of SIGCOMM*, 2009.

[23] J. Edmonds, "Edge-disjoint branchings," *Combinatorial Algorithms*.

[24] A. Bhalgat, R. Hariharan, T. Kavitha, and D. Panigrahi, "Fast Edge Splitting and Edmonds' Arborescence Construction for Unweighted Graphs," in *Proc. SODA*, 2008.

[25] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *SIGCOMM*, 2008.

[26] R. Diestel, *Graph Theory*. Springer, February 2000.

[27] A. Gopalan and S. Ramasubramanian, "Multipath Routing and Dual Link Failure Recovery in IP Networks Using Three Link-Independent Trees," in *Proc. ANTS*, 2011, pp. 1–6.

[28] S. Kini, S. Ramasubramanian, A. Kvalbein, and A. Hansen, "Fast recovery from dual-link or single-node failures in ip networks using tunneling," *IEEE/ACM Trans. Networking*, vol. 18, no. 6, pp. 1988–1999, Dec 2010.

[29] A. Kvalbein, A. F. Hansen, T. Čičic, S. Gjessing, and O. Lysne, "Multiple Routing Configurations for Fast IP Network Recovery," *IEEE/ACM Trans. Networking*, vol. 17, no. 2, pp. 473–486, April 2009.

[30] S. Lor, R. Landa, and M. Rio, "Packet re-cycling: eliminating packet losses due to network failures," in *HotNets IX*. ACM, 2010, p. 2.

[31] M. Motiwala, M. Elmore, N. Feamster, and S. Vempala, "Path Splicing," in *Proc. of SIGCOMM*, 2008, pp. 27–38.

[32] G. T. Nguyen, R. Agarwal, J. Liu, M. Caesar, P. B. Godfrey, and S. Shenker, "Slick Packets," *SIGMETRICS*, 2011.

[33] S. Sae Lor, R. Landa, R. Ali, and M. Rio, "Handling Transient Link Failures Using Alternate Next Hop Counters," in *NETWORKING*, 2010.

[34] M. Shand, S. Bryant, and S. Previdi, "IP Fast Reroute using Not-Via Addresses," *IETF Internet draft version 05*, March 2010.

[35] K. Xi and H. J. Chao, "IP Fast Reroute for Double-link Failure Recovery," in *Proc. IEEE GLOBECOM*, 2009.

[36] M. Xu, Q. Li, L. Pan, and Q. Li, "MPCT: Minimum Protection Cost Tree for IP Fast Reroute Using Tunnel," in *Proc. IWQoS*, 2011.

[37] G. Robertson and S. Nelakuditi, "Handling Multiple Failures in IP Networks through Localized On-Demand Link State Routing," *IEEE Transactions on Network and Service Management*, vol. 9, no. 3, 2012.

[38] V. Liu, D. Halperin, A. Krishnamurthy, and T. E. Anderson, "F10: A Fault-Tolerant Engineered Network," in *Proc. of NSDI*, 2013.

[39] C. Filsfils, P. Francois, B. Decraene, J. Uttaro, N. Leymann, and M. Horneffer, "Loop-Free Alternate (LFA) Applicability in Service Provider (SP) Networks," *RFC 6571, IETF, 2012*.