Failure Inference for Shortening Traffic Detours

Anmin Xu, Jun Bi,Baobao Zhang, Shuhe Wang, Jianping Wu
Tsinghua University
Email:{xam14@mails.tsinghua.edu.cn,junbi@tsinghua.edu.cn}

***Abstract*—To speed up the recovery from network failures, an extensive list of methods have been proposed. Many failure-recovery methods are proposed based on tunneling or marking, which increase the packet processing burden on routers and consume extra bandwidth. With neither tunneling nor marking, existing methods guarantee recovery from any single-link failure if a detour for the failed link exists, but they generate long traffic detours that will degrade the network performance, and even increase the operational cost, which is undesirable to network operators. Therefore, in this paper, we propose a Failure Inference approach to shortening Traffic Detours named as FITD, which works in OSPF/IS-IS networks. FITD does not use explicit failure notification, and can infer which link fails based on traffic information. FITD guarantees recovery from any single-link failure if a detour for the failed link exists. In particular, for networks with symmetric link weights, FITD guarantees to generate shortest detours for any single-link failure.**

Keywords: fast reroute; failure recovery; shortest detours

1. Introduction

With the increase in variety and quality of Internet applications and services, the higher network performance are demanded to support those applications and services, which range from delay sensitive applications, such as, instant messaging, financial applications and multiplayer online game, to computational intensive applications, such as, web search, data mining and scientific computing. Many studies focus on how to make the network provide the reliable service quality assurance to the various applications and services, and one of the most import problem is to handle the network failures. However, network failures occur very frequently based on diverse network parameters and situations [1], and the convergence time after network failures occur is of the order of hundreds of milliseconds [2], even the order of seconds.

Therefore, an extensive list of methods have been proposed to speed up recovery from network failures. A large body of failure-recovery methods, such as [3] [4] [5], are proposed based on tunneling or marking, which increases the packet processing burden on routers, consumes extra bandwidth and even causes packet re-fragmenting [6]. With neither tunneling nor marking, existing methods [7] [8] [9] [10] have achieved recovery from any single-link failure if a detour for the failed link exists.

Under single-link failure recovery, a detour is defined as a path from the source of a failed link to a destination and the length of a detour is defined as the sum of the weights of the links on the detour, while the weights of links are generally set to optimize multiple objectives, such as economic cost, throughput, delay, etc [11] [12]. Long traffic detours degrade the network performance, and even increase the operational cost, which is undesirable to network operators. The reason why the existing methods cannot guarantee to generate the shortest detours is that they are unable to precisely localize the link failure. As a result, they have to sacrifice the optimality of detours to protect more links.

In this paper, we propose a Failure Inference approach to shortening Traffic Detours named as FITD, which works in OSPF/IS-IS networks and uses neither tunneling nor marking. FITD guarantees recovery from any single-link failure if a detour for the failed link exists. In particular, for networks with symmetric link weights, FITD guarantees to generate shortest detours for any single-link failure. Although FITD has some limitations, it still has great significance due to the facts that 70% of unplanned network failures are single-link failures [1] and that the link weights in OSPF/IS-IS networks are generally symmetric.

We describe core ideas of FITD as follows. We prove that we can always find such a shortest detour with a good feature that remote routers can infer which link fails from traffic information without using explicit failure notification. The good feature is that the traffic affected by a failure is first reversely forwarded (i.e. forwarded along reverse routing paths), and then is normally forwarded. A distinct characteristic of packets going along the reversely forwarding path is that these packets come to routers from their primary next hops, where primary next hops refer to next hops generated by the normal OSPF/IS-IS routing. Therefore, if a router receives a packet towards a destination from the primary next hop towards the destination, the router can infer that a failure occurs on the routing path from the router to the destination. Further, by combining the Time-To-Live (TTL) information of the packet, the router can infer which link on the routing path from the router to the destination fails, and then immediately invoke the pre-installed backup next hop for the failed link towards the destination. As a summary, FITD has the following features:

- Once a link fails, the router that is adjacent to the link

978-1-5386-2704-4/17/$31.00 ©2017 IEEE

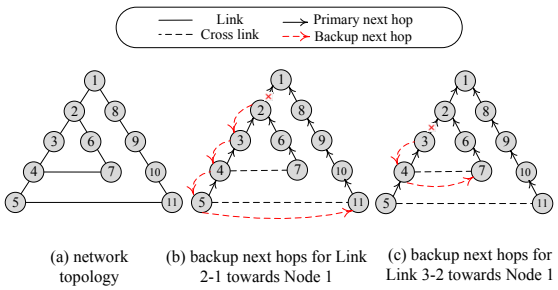


Figure 1. The example of update for TE. To decrease the load of link between node a and node b, all of those nodes need to update several flow entries.

failure can locally detect the link failure and immediately invoke the corresponding pre-installed backup next hops.Routers that are not adjacent to the link failure can infer the link failure based on traffic information without using explicit failure notification and also immediately invoke the corresponding pre-installed backup next hops.

- FITD guarantees recovery from any single-link failure if a detour for the failed link exists. In particular, for networks with symmetric link weights, FITD guarantees to generate shortest detours for single-link failure.
- FITD is based on existing OSPF/IS-IS protocol and does not add any new control-plane messages. To deploy FITD, we only need to update the software of routers.

The rest of this paper is organized as follows. In Section 2, we describe the shortest detour model. In Section 3, we design FITD-basic. Furthermore, in Section 4, we design FITD-extended to make FITD work in more general situations. In Section 5, we evaluate the proposed methods. In Section 6, we introduce the related work. In Section 7, we make some discussions and conclusions.

2. Shortest Detour Model

In this section, we first describe the three basic assumptions of shortest detour model.

Assumption 1. FITD works in OSPF/IS-IS networks, where the routing is shortest-path based.

Assumption 2. There is only one primary next hop towards one destination.

Assumption 3. If a router has two or more shortest-path next hops towards a destination, the router uses the shortest-path next hop with the smallest router-id as the primary next hop towards the destination.

Now, we describe core ideas and develop key theories for achieving shortest detours. We use $G = (V, E, W)$ to denote a strongly connected network topology that is composed of a set of nodes V, a set of directed links E and a set of link weights W. The nodes in V are represented by integers $1, 2, ..., |V|$. We use a two-tuple a-b to denote a link, where the router a is connected to the router b. Given any link a-b, if the weight of the link a-b is the same as the weight of the link b-a, we say the link weights of the network are symmetric. Here onwards the words node and router are used interchangeably.

A detour from a failed link a-b to a destination d is defined as a path from a to d without traversing the link a-b. The length of the detour is the sum of the weights of the links on the detour. For example, as shown in Figure 1(b), Path 2-3-4-5-11-10-9-8-1 is a detour from Link 2-1 to the destination node 1. The length of the detour is 8 units.

To achieve the shortest detours in IP networks, one challenge is how to construct the shortest detours for the affected traffic (i.e. how to forward the affected traffic along the shortest detour) by using the hop-by-hop forwarding way of IP networks. Fortunately, we can always find such a shortest detour with a very good feature that the shortest detour can be easily constructed. In this section, we first describe what a detour that has the good feature is. We then describe why a detour that has the good feature can be easily constructed by using the hop-by-hop forwarding way. Finally, we prove that we can always find such a detour that has the good feature.

1) What is a detour that has the good feature?

A detour that has the good feature is called a graceful detour. To describe what a graceful detour is, we need to first describe definitions of the routing tree, the cross link and the graceful path as follows.

Under Assumption 2, the links associated with the primary next hops towards a destination form a tree rooted at the destination only if there are no routing loops in the network. The tree rooted at the destination is called the routing tree of the destination. As shown in Figure 1(b), the primary next hops towards Node 1 forms the routing tree of the destination node 1. In OSPF/IS-IS networks, the routing tree of a destination can be computed by computing shortest-path tree rooted at the destination based on the ubiquitous topology in the link-state database. Because of Assumption 3, the routing trees of the destination computed by all the routers are the same.

We call a link as a cross link for a destination if and only if neither the link itself nor the reverse link of the link are in the routing tree of the destination. As shown in Figure 1(b), Link 4-7, Link 7-4, Link 5-11 and Link 11-5 are four cross links for the destination node 1.

We define a path as a graceful path for a destination if and only if the path satisfies the following four conditions: 1) there exists exactly one cross link for the destination on the path; 2) the (possibly empty) path before the cross link on the path is a reversely-forwarding path for the destination; 3) the (possibly empty) path after the cross link on the path is a normally-forwarding path for the destination; and 4) one node appears at most one time on the path. As shown in Figure 1(b), Path 2-3-4-5-11-10-9-8-1 is a graceful path for the destination node 1. Path 2-3-4-5 is a reversely-forwarding path for the destination node 1 and Path 11-10-9-8-1 is a normally-forwarding path for the destination node 1.

We call a detour from a failed link to a destination as a graceful detour if and only if the detour is a graceful path

for the destination. As shown in Figure 1(b), the shortest graceful detour from Link 2-1 to the destination node 1 is 2-3-4-5-11-10-9-8-1.

2) Core ideas of constructing a shortest detour that is also a graceful detour by using the hop-by-hop forwarding way

If a shortest detour from a link to a destination is a graceful detour, the detour can be easily constructed by using the hop-by-hop forwarding way of IP networks because of the feature of the graceful path. Routers on the normally-forwarding path only need to normally forward the affected traffic (i.e. using the primary next hops); only routers on the reversely-forwarding path need to invoke pre-installed backup next hops for the link towards the destination. Routers independently compute their backup next hops. In Subsection B of Section 3, we will describe how to compute backup next hops. Here we only need to know that backup next hops are assigned so as to form the shortest detour.

The condition of a router invoking backup next hops towards a destination is that a failure occurs on the routing path from the router itself to the destination. Different links may have different backup next hops to generate shortest detours. As shown in Figure 1, on Node 4, the backup next hop for Link 2-1 towards Node 1 is different from the backup next hop for Link 3-2 towards Node 1. Thus, routers should invoke the backup next hop for the failed link towards the destination. However, how does a remote router know whether a failure occurs on the routing path from itself to the destination? Further, how does the remote router know which link on the routing path from itself to the destination fails? One straightforward way is to use explicit failure notification. However, explicit failure notification not only delays the reaction time to the failure, but also brings communication cost. This is undesirable for network operators.

Fortunately, if a shortest detour is a graceful detour, by using the feature of the graceful detour, the failure information can be directly inferred from traffic information based on incoming interfaces and TTL values. For IPv6, the hop limit field has the same meaning as the TTL field in IPv4. For description simplicity, we use the term TTL in the rest of this paper. We will describe the specific way of inferring a failed link in Subsection A of Section 3.

3) Why does a shortest detour that is also a graceful detour always exist? How to find a shortest detour that is also a graceful detour?

We now describe why a shortest detour that is also a graceful detour must exist. To prove this, we first give a lemma.

Lemma 1. *In networks with symmetric link weights, given any detour from a link to a destination, we can always find a graceful detour that is not longer than the given detour.*

Proof. See the appendix of this paper [13]. □

Given any link and any destination, we suppose that p_1 is a shortest detour from the given link to the given destination. According to Lemma 1, we can find a graceful detour p_2 that is not longer than the shortest detour p_1. Therefore, the graceful detour p_2 is also a shortest detour from the given link to the given destination. As a result, only if a detour from a link to a destination exists, there must exist a shortest detour that is also a graceful detour from the link to the destination. We can also get a theorem as follows.

Theorem 1. *In networks with symmetric link weights, given any link and any destination, the shortest graceful detours for the given link and the given destination are also shortest detours for the given link and the given destination.*

Proof. See the appendix of this paper [13]. □

According to Theorem 1, to find a shortest detour from a link to a destination, we only need to find a shortest graceful detour among all the graceful detours for the link and the destination. The challenge is how to find a shortest graceful detour with low computational complexity.

3. FITD-BASIC

In this section, we first design the forwarding mechanism of FITD-basic, which is based on three assumptions in Section 2. Then, we design an algorithm of computing backup next hops with low computational complexity.

A. Forwarding Mechanism of FITD-basic

In this subsection, we design the forwarding mechanism of FITD-basic including the forwarding data structure and forwarding logic.

1) Forwarding data structure of FITD-basic

Given a local router and a destination, the local router only needs to assign backup next hops for the links on the routing path from the local router to the destination. This is because the failures of the other links will not affect the traffic from the local router to the destination.

Suppose that there are n links on the routing path from the local router to the destination, the n links can be orderly numbered as the $1^{st}, 2^{nd}, ..., n^{th}$ link. Therefore, backup next hops of the local router towards the destination for the n links can be saved in an array as follows. Suppose that the local router is s and the destination is d, we use an array $Backup_{s,d}[1, 2, ..., n]$ to save the backup next hops of the router s towards the destination d for the n links on the routing path from the local router s to the destination d. The advantage of saving backup next hops in an array is that the backup next hop for a link can be directly fetched, so the lookup time is not needed. Note that backup next hops are saved in Static Random-Access Memory (SRAM) instead of Ternary Content-addressable Memory (TCAM). TCAM only needs to save an SRAM address for each destination, where the next hops including primary next hops and backup next hops are saved in SRAM. Therefore, FITD does not

impose extra pressure on TCAM. SRAM is cheap even in large volumes. The SRAM memory space needed by backup next hops is fully acceptable as shown in our experiments.

2) Forwarding logic of FITD-basic

Here we first assume that all the backup next hops have been assigned to form the shortest detours. We will describe the algorithm of computing backup next hops in Subsection C. If a router s infers that the k^{th} link on the routing path from s to d fails, the router s immediately invokes the backup next hop $Backup_{s,d}[k]$ towards the destination d.

The key is how to infer which link fails. We now describe the proposed mechanism of inferring which link fails without using explicit failure notification.

From the descriptions above, we know that only routers on reversely-forwarding paths need to invoke backup next hops. A distinct characteristic of packets going along a reversely forwarding path is that these packets come to the routers from their primary next hops. Therefore, if a router s receives a packet towards a destination d from its primary next hop, the router s can infer that a failure has occurred on the routing path from s to d. Further, by combining the Time-To-Live (TTL) information of the packet, the router s can infer which link on the routing path from s to d fails as follows.

Algorithm 1: Forwarding logic of FITD-basic

```
Input: a packet
Output: the next hop of the packet
Constant: the local router s, and the maximum
          routing-tree depth l;
reduce the TTL value of the packet by one;
d = the destination that the packet goes to;
v = the TTL value of the packet;
if the packet is from the primary next hop towards d then
    k = l − (v mod l); // number of traversed hops;
    return Backup_{s,d}[k + 1];
else if the primary next hop towards d fails then
    reset the TTL of the packet to be v − (v mod l);
    return Backup_{s,d}[1];
else
    return the primary next hop towards d;
```

The failed link can be further inferred from the number of traversed hops from the failed link to the router s. If the number of traversed hops from the failed link to the router s is k, this implies that the $k+1^{th}$ link on the routing path from s to d fails, so the backup next hop $Backup_{s,d}[k+1]$ should be invoked by the router s towards the destination d. Thus, the problem is changed to be: how to make the router s know the number of traversed hops from the failed link to the router s. The number of traversed hops can be inferred from the TTL field.

We mainly use the characteristic that the TTL value of a packet is reduced by one on every hop in practice. In addition, we need to use two operations on TTL. One operation is to read the TTL value. The other operation is to reset the TTL value. We put the operation of reducing the TTL value by one before our operation of reading TTL and our operation of resetting TTL. We first use a simple mechanism to describe the core idea of inferring the number of traversed hops by using TTL as follows.

If a router detects the primary next hop of a packet fails, the router resets the TTL field of the packet to be a new value x. When a remote router receives this packet, suppose that the TTL value of the packet is y. The reset TTL value x can be embed in each router, so the remote router can infer that the number of traversed hops from the failed link to itself is x-y. This simple mechanism only describes the core idea, but is not practicable. This is because the TTL value must be reduced on every hop [14]. It is impossible to ensure that the new reset TTL value x is less than the original TTL value.

Thus, we design a clever method as follows. If a router detects that the primary next hop of a packet fails, we suppose that the TTL value of the packet is v_1. The router resets the TTL field of the packet to be $v_1 - (v_2 \mod l)$ making the new TTL value be an integer multiple of l, where l is the depth of the routing tree that has the maximum depth among all the routing trees of the network. For simplicity, the parameter is called the maximum routing-tree depth. When a remote router receives this packet, we suppose that the TTL value of the packet is . The remote router can infer that the number of traversed hops from the failed link to itself is $l - (v \mod l)$. In addition, the reset TTL value $v_1 - (v_1 \mod l)$ must be less than the original TTL value of the packet, obeying the rule that the TTL value is reduced on every hop. We refer readers to the appendix of this paper for the proof [13].

The maximum routing-tree depth can be computed based on the topology in the link-state database, which is the same on all routers. On the router that is adjacent to the link failure, the TTL value of the packet affected by the link failure is reduced by $v \mod l$.

Readers may wonder to know whether such TTL reduction will lead to the packet being discarded at a latter node. In theory, such TTL reduction may lead to the packet being discarded at a latter node. However, in general, this will not happen because of the following fact. The effect of such TTL reduction is equivalent to the effect of the packet traversing one extra autonomous system. Generally, the packet will not be discarded by a middle router only due to traversing one extra autonomous system. In addition, readers may wonder to know whether FITD can work in the case of $v_1 < l$. In theory, if $v_1 < l$, the TTL of the packet will be reset to be zero, so the packet will be dropped.

However, in general, the situation $v_1 < l$ will not happen because of the following fact. If $v_1 < l$, it implies that the packet may be dropped if the packet traverses one more autonomous system. In general, the initial TTL value of a packet should be set to be a large enough value to traverse more enough and an unknown number of autonomous systems.

Algorithm 1 shows the specific forwarding logic of FITD-basic. The main forwarding logic of FITD-basic is if

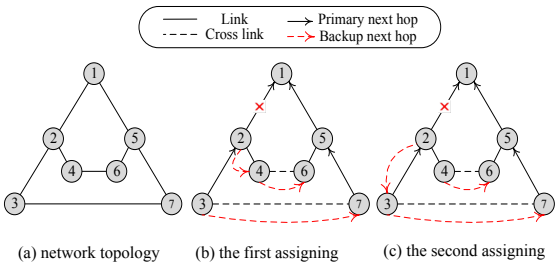


Figure 2. Example for Rule 1. The weight of each link is one unit

a router s infers that the k^{th} link on the routing path from s to d fails, the router s immediately invokes the backup next hop $Backup_{s,d}[k]$ towards the destination d. The failure inference mechanism is based on incoming interfaces and TTL values as described above. We move the operation of reducing the TTL by one to the forwarding logic of FITD-basic in order to ensure that the operation of reducing the TTL value by one is before our operation of reading TTL and our operation of resetting TTL. Like other failure-recovery methods (e.g. FIR [7] and RPFP [8]), FITD also suppresses the network convergence process under a single-link failure. On a router, the network convergence process is not triggered until the router finds that there are multiple link failures based on received Link-State Announcements (LSAs). Suppression of the network convergence process is reasonable because of the evidence that most failures are short-lived [1] . Measurements in [1] show that 50% of network failures last less than one minute and 85% of network failures last less than ten minutes. Rapidly triggering the network convergence process can cause route flapping and increase network instability [15].

B. Pre-computing Backup Next Hops

The key to computing backup next hops is how to compute shortest graceful detours. Once the shortest graceful detour from a link to a destination is computed, backup next hops can be directly assigned according to the shortest graceful detour.

Theorem 2. *Given a shortest graceful detour from a failed link to a destination d, if a router s is on the reversely-forwarding path of the detour, the path from s to d on the detour must be a shortest graceful path from s to d without traversing the failed link.*

Proof. See the appendix of this paper [13]. □

Therefore, according to Theorem 2, we design Rule 1 to assign backup next hops.

Rule 1. *Given a router s, a destination d and a link a-b on the routing path from s to d, the backup next hop of the router s for the link a-b towards the destination d is assigned to be the next hop of the router s on a shortest graceful path from s to d without traversing the link a-b.*

We now give an example for Rule 1. Figure 2(a) shows a network topology, where the weight of each link is one unit. On Node 2, there are two shortest graceful paths (Path 2-4-6-5-1 and Path 2-3-7-5-1) from Node 2 to Node 1 without traversing Link 2-1. Node 2 can assign the backup next hop for Link 2-1 towards Node 1 according to any of the two shortest graceful paths. Figure 2(b) and Figure 2(c) separately show the two feasible ways of assigning backup next hops. Of course, Node 2 can use the both feasible backup next hops for Link 2-1 towards Node 1 to balance load, but it is not the focus of this paper.

Algorithm 2: Computing backup next hops to a destination for FITD-basic

Input: Topology $G = (V, E, W)$, a local router s, a destination d
Output: $Backup_{s,d}[]$

```
1  Compute the routing tree of d;
2  Compute depth_d[], rouLen_d[] and revLen_d[];
3  Compute cross_{s,d}, LCA_d[] and nextHop_{s,d}[]
     // Algorithm 3 shows the pseudocode of Line 3 ;
4  sort_cross_{s,d} = In creasing sort cross_{s,d} according to
     crossLen_{s,d}[];
5  low = depth[s] − 1;
6  for each cross link c ∈ sort_cross_{s,d} do
7      peak = depth[LCA_d[c]];
8      while low ≥ peak do
9          k = depth[s] − low;
10         Backup_{s,d}[k] = nextHop_{s,d}[c];
11         low = low − 1;
```

Therefore, according to Rule 1, the key to computing backup next hop of a router for a link towards a destination is to compute one shortest graceful path from the router to the destination without traversing the link. We first describe what the Least Common Ancestor (LCA) is. The least common ancestor of two nodes in the routing tree of a destination is the node that is an ancestor of the two nodes and that has the greatest depth in the routing tree of the destination. For example, in the routing tree in Figure 1, the LCA of Node 4 and Node 7 is Node 2.

Theorem 3. *Given a router s, a destination d and a cross link x-y of d, the cross link x-y can generate exactly one graceful path from s to d if and only if in the routing tree of d the LCA of x and y is an ancestor of s and x is s or a descendant of s. The graceful path generated by the cross link x-y is the reverse path of the routing path from x to s, plus the cross link x-y and plus the routing path from y to d.*

Proof. See the appendix of this paper [13]. □

Theorem 4. *Given a router s, a destination d and a cross link x-y that can generate a graceful path from s to d, the graceful path generated by the cross link x-y from s to d must not traverse the links on the routing path from s to $LCA(x, y)$, where $LCA(x, y)$ is the least common ancestor of x and y in the routing tree of d.*

Proof. See the appendix of this paper [13]. □

We now describe the algorithm of computing backup next hops as shown in Algorithm 2. Table 1 shows the meanings of variables for Algorithm 2 and Algorithm 3.

TABLE 1. NOTATION TABLE

Signs	Meanings
$G = (V, E, W)$	Network Topology composed of a set of nodes V, a set of links E and a set of link weights W
$Backup_{s,d}[i]$	The backup next hop of the router s towards the destination d for the i^{th} link on the routing path from s to d
$Father_d^n[s]$ $(n \geq 1)$	The father of $Father_d^{n-1}[s]$ in the routing tree of d, where $Father_d^0 = s$.
$depth_d[i]$	The depth of Node i in the routing tree of d
$rouLen_d[i]$	The length of the routing path from i to d
$revLen_d[i]$	The length of the reverse path of the routing path from i to d
$cross_{s,d}$	The set of cross links that can generate graceful paths from s to d
$LCA_d[i]$	The least common ancestor of the cross link $i \in cross_{s,d}$ in the routing tree of d
$crossLen_{s,d}$	The length of the graceful path from s to d generated by the cross link $i \in cross_{s,d}$
$nextHop_{s,d}[i]$	The next hop of the router s on the graceful path from s to d generated by the cross link $i \in cross_{s,d}$

In the first step, we compute the routing tree of the destination d and compute some variables by traversing the routing tree. Because networks are sparse graphs, we use the Johnson algorithm [16] to compute the routing tree of a destination in computational complexity of $O(|E| * log(|V|))$. We use the Depth First Search (DFS) algorithm [16] to compute the variables $depth_d[]$, $rouLen_d[]$ and $revLen_d[]$ in the computational complexity of $O(|E|)$. Therefore, the computational complexity of this step is $O(|E|*log((|V|)))$. The pseudocode of this step is Line $1-2$ in Algorithm 2.

In the second step, the main task is to compute the set of cross links that can generate graceful paths from s to d. In addition, LCAs, lengths of graceful paths and next hops associated with the cross links are computed in this step. The computational complexity of this step is $O(|E|)$. The pseudocode of this step is Line 3 in Algorithm 2. Algorithm 3 is the specific algorithm for Line 3 in Algorithm 2. We now describe Algorithm 3. According to Theorem 3, given a cross link x-y, the cross link x-y can generate exactly one graceful path from s to d if and only if in the routing tree of d, 1) the LCA of x and y is an ancestor of s ; and 2) x is s or a descendant of s. Specifically, we judge whether the two conditions are satisfied by cleverly coloring nodes in the routing tree of d as follows. We use $Father_d^n[s](n \geq 1)$ to denote the father of $Father_d^{n-1}$, where $Father_d^0 = s$. We first color s and all the descendants of s with the ID 0 in the routing tree of d.

For the nodes that have not been colored, in the sequence of $Father_d^1[s]$,$Father_d^2[s]$, ..., $Father_d^n[s] = d$, we color $Father_d^k[s](1 \leq k \leq n)$ and all the descendants of $Father_d^k[s]$ with the ID k in the routing tree of d. Given a cross link x-y of the destination d, 1) the LCA of x and y is an ancestor of s; and 2) x is s or a descendant of s if and only if the color of x is 0 and the color of y is not 0. In addition, if the color of x is 0 and the color of y is not 0, the LCA of x and y is $Father_d^k[s]$, where k is the color of y. We prove these in the appendix [13].

Algorithm 3: Specific algorithm for Line 3 in Algorithm 2

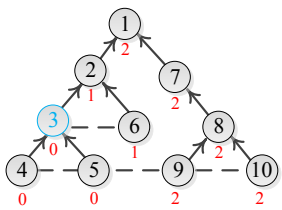
Input: a local router s, a destination d, T is the routing tree of the destination d, C is the set of cross links for the destination d, $rouLen_d[]$ and $revLen_d[]$

Output: $cross_{s,d}$,$LCA_d[i]$,$crossLen_{s,d}[]$ and $nextHop_{s,d}[]$

```
1  LastIter = −1 //LastIter denotes the last visited node;
2  Iter = s // Iter denotes the visiting node;
3  ColorNum = 0 //ColorNum is the id of the current color;
4  while Iter is not NULL do
5      color[Iter] = ColorNum //Color the node Iter with the id ColorNum;
6      Peak[ColorNum] = Iter //Peak of nodes with color ColorNum is Iter;
7      Color all the descendants of Iter in T excluding the descendants of LastIterwith ColorNum;
8      LastIter = Iter;
9      Iter =The father of Iter in T;
10     ColorNum = ColorNum + 1;
11 cross_{s,d} = ∅;
12 for each cross link c ∈ C do
13     c is denoted by x-y;
14     if color[x] ≠ 0 then
15         continue;
16     if color[y] = 0 then
17         continue;
18     cross_{s,d} = cross_{s,d} ∪ c;
19     LCA_d[c] = Peak[color[y]] //Peak[color[y]] is Father_d^{color}[s];
20     crossLen_{s,d}[c] = revLen_d[x] − rouLen_d[s] + rouLen_d[y]+ the weight of the cross link x-y;
21     // crossLen_{s,d}[c] = rouLen_d[x] − rouLen_d[s] + rouLen_d[y]+ the weight of the cross link y-s;
22 nextHop_{s,d}[] can be easily computed by coloring the descendants of each son of s in T with the node id of the corresponding son;
```

We now use an example to illustrate Algorithm 3. Figure 3 shows a network topology where the weight of each link is one unit. The solid arrows construct the routing tree of Node 1. The dash lines refer to the cross links of Node 1. We suppose that the local router is Node 3 and the destination is Node 1. We consider how to compute the set of cross links that can generate graceful paths from Node 3 to Node 1. We color all the nodes by using the method above. The colors of the nodes are marked under the nodes. For Cross link 5-9, the color of Node 5 is 0 and the color of Node 9 is not 0, so Cross link 5-9 can generate a graceful path 3-5-9-8-7-1 from Node 3 to Node 1. The length of the graceful path generated by Cross link 5-9 is 5 units. If Node 3 uses the graceful path 3-5-9-8-7-1 generated by Cross link 5-9, the next hop of Node 3 towards Node 1 is Node 5. The LCA of Node 5 and Node 9 is $Father_1^2[3]$, which is Node 1.

In the third step, we increasingly sort the cross links



The weight of each link on this network topology is one unit. The solid arrows construct the routing tree of Node 1. The dash lines refer to the cross links of Node 1. We suppose that the local router is Node 3 and the destination is Node 1.

Figure 3. The example of Algorithm 3.

that can generate graceful paths from s to d according to the lengths of the graceful paths. The computational complexity of this step is $O(|E|*log(|V|))$. The pseudocode of this step is Line 4 in Algorithm 2.

In the fourth step, we assign backup next hops of the router s towards the destination d for the links on the routing path from s to d. The shorter graceful path has the higher priority to be used for assigning backup next hops. Theorem 4 tells us which links the graceful path generated by a cross link does not traverse, i.e. which links the graceful path generated by the cross link can protect. The computational complexity of this step is $O(|E|)$. The pseudocode of this step is Line $5-11$ in Algorithm 2.

As a result, the total complexity of computing backup next hops towards a destination is $O(|E|*log(|V|))$. Because backup next hops are pre-computed in an off-line way, such computational complexity is fully acceptable. We can get Theorem 5 for FITD-basic.

Theorem 5. *In any network, where Assumption 1, Assumption 2 and Assumption 3 hold, FITD-basic guarantees recovery from any single-link failure only if a detour for the failed link exists. In particular, if the link weights of the network are symmetric, FITD-basic guarantees to generate the shortest detour from any failed link to any destination.*

Proof. See the appendix of this paper [13]. □

4. FITD-Extended

To make FITD work in more general situations, we propose FITD-extended by canceling Assumption 2 and Assumption 3, but still keep Assumption 1.

Without Assumption 2 and Assumption 3, if a router has multiple shortest-path next hops towards a destination, the router can use any one of them as the primary next hop towards the destination, or can use some (all) of them as multiple primary next hops towards the destination to balance the load. The difficulty is that the links associated with the primary next hops towards a destination forms a Directed Acyclic Graph (DAG), which is not always a tree. Moreover, because routers do not know that other routers use which shortest-path next hops as primary next hops, routers cannot compute the DAG only based on the global topology. Our solution for overcoming the difficulty is to define a virtual routing as follows.

Given a router and a destination, if the router has only one shortest-path next hop towards the destination, the virtual primary next hop of the router towards the destination is the shortest-path next hop. If the router has multiple shortest-path next hops towards the destination, the virtual primary next hop of the router towards the destination is the shortest-path next hop with the smallest router-id among those shortest-path next hops. Accordingly, routing trees, cross links and graceful paths defined on the primary next hops are called virtual routing trees, virtual cross links and virtual graceful paths. For computing backup next hops, FITD-extended also uses Algorithm 2 and Algorithm 3 by replacing Line 20 with Line 21 in Algorithm 3. In a network with symmetric link weights, the value of Line 20 is the same as the value of Line 21 in Algorithm 3. The motivation of the modification is to make FITD-extended work in networks with asymmetric link weights. Note that for FITD-extended, the routing trees and cross links in Algorithm 2 and Algorithm 3 refer to virtual routing trees and virtual cross links. The forwarding logic of FITD-extended should be modified to be Algorithm 4.

Algorithm 4: Forwarding logic of FITD-extended

Input: a packet
Output: the next hop of the packet
Constant: the local router s, and the maximum routing-tree depth l

1 reduce the TTL value of the packet by one;
2 $v =$ the TTL value of the packet;
3 $d =$ the destination that the packet goes to;
4 $p =$ the real primary next hop that s means to forward the packet to;
5 **if** *the packet is from the virtual primary next hop towards* d **then**
6 $\quad k = l - (v \bmod l)$ // number of traversed hops;
7 $\quad$ return $Backup_{s,d}[1]$;
8 **else if** *the real primary next hop* p *fails* **then**
9 $\quad$ **if** p *is the virtual primary next hop towards* d **then**
10 $\quad\quad$ reset the TTL of the packet to be $v - (v \bmod l)$;
11 $\quad\quad$ return $Backup_{s,d}[1]$;
12 $\quad$ **else**
13 $\quad\quad$ return the virtual primary next hop towards d;
14 **else if** *the packet is from a virtual cross link* **then**
15 $\quad$ return the virtual primary next hop towards d;
16 **else**
17 $\quad$ return p;

For FITD-extended, backup next hops, virtual primary next hops and virtual cross links adjacent to the router are saved in SRAM. We now give an example to illustrate how FITD-extended works.

Figure 4(a) shows a network topology, where the weight of each link is one unit. Figure 4(b) shows the real routing towards the destination node 1. In Figure 4(b), the solid arrows direct the primary next hops towards the destination node 1. We use Figure 4(c) to illustrate how FITD-extended works. In Figure 4(c), the solid arrows direct to the virtual primary next hops towards the destination node 1, which forms the virtual routing tree of the destination node 1. The

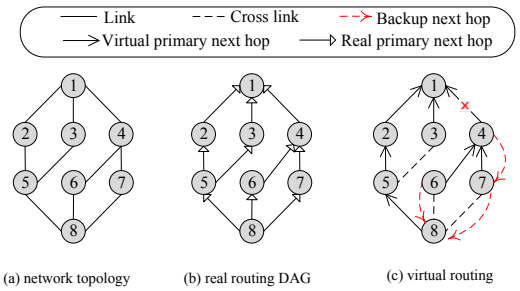


Figure 4. The example for FITD-extend

dash arrows direct to the backup next hops pre-computed by FITD-extended for Link 4-1 towards the destination node 1 based on the virtual routing. Once Link 4-1 fails, the traffic affected by the failed link 4-1 towards Node 1 will be forwarded along Path 4-7-8-5-2-1 and Path 4-7-8-5-3-1 according to the forwarding logic of FITD-extended as follows. Node 4 can locally detect the link failure, so Node 4 can invoke the corresponding backup next hop (Node 7) towards Node 1. Node 7 receives the affected traffic from its virtual primary next hop (Node 4), so Node 7 can also infer the link failure, and invoke the corresponding backup next hop (Node 8) towards Node 1. Node 8 receives the affected traffic from a virtual cross link, so Node 8 can also infer the link failure, and invoke the virtual primary next hop (Node 5) towards Node 1. From Node 5, the affected traffic towards Node 1 can be normally forwarded along the two shortest paths 5-2-1 and 5-3-1. We can get Theorem 6 for FITD-extended.

Theorem 6. *In any network, where Assumption 1 holds, FITD-extended guarantees recovery from any single-link failure only if a detour for the failed link exists. In particular, if the link weights of the network are symmetric, FITD-extended guarantees to generate the shortest detour from any failed link to any destination.*

Proof. See the appendix of this paper [13]. □

5. Evaluation

In this section, we first describe our datasets. We then compare FITD and FIR in terms of the detour stretch. Finally, we evaluate the size of the forwarding table generated by FITD.

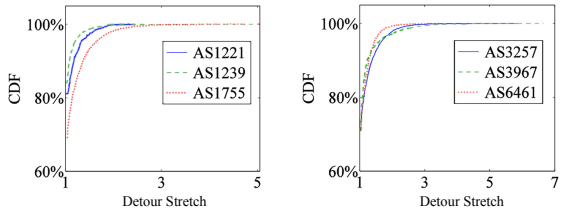
A. Datasets

We consider six intra-domain topologies with inferred link weights from Rocketfuel [17], where weights of links are symmetric. The summary information of those topologies is shown in Table 2.

The column header describes the names of those topologies. In the row header, #nodes denotes the number of nodes and #links denotes the number of links.

TABLE 2. TOPOLOGY INFORMATION

	AS 1221	AS 1239	AS 1755	AS 3257	AS 3967	AS 6461
#nodes	208	630	174	322	158	276
#links	302	1944	322	656	294	744



(a) Results on the first three topologies **(b) Results on the last three topologies**

Figure 5. CDF curves of detour stretches generated by FIR

B. Detour Stretch

In this subsection, we compare FITD with existing methods in terms of the detour stretch. The stretch of a detour from a failed link to a destination generated by a failure-recovery method is defined as the ratio of the length of the detour from the failed link to the destination generated by the failure-recovery method to the length of the shortest detour from the failed link to the destination. By analyzing the algorithms of FIR [7] , ESCAP [9], DisPath [10] and RPFP [8] , we know that detours generated by FIR are the shortest among them. This is because FIR uses heuristics to shorten traffic detours. Thus, we compare FITD with only FIR in terms of the detour stretch. Given a topology and a failure-recovery method, we enumerated each detour case (d, e), where d is a node on the topology and e is a link in the routing tree of the destination d. For each detour case (d, e), we calculated the stretch of the detour from the link e to the destination d under the given failure-recovery method. Our evaluation results show that detour stretches generated by FITD are always one on those topologies, which further validates our theory. Our evaluation results also show that the average detour stretch generated by FIR is 1.05 $\sim$ 1.11 and the maximum detour stretch generated by FIR is 2.43 $\sim$ 6.65 on those topologies. To know the distribution of detour stretches generated by FIR, we also plot Cumulative Distribution Function (CDF) curves of the detour stretches generated by FIR on those topologies as shown in Figure5.

For a point (x, y) on a CDF curve, the vertical coordinate y is the fraction of cases whose values are smaller than or equal to the horizontal coordinate x. The detour stretch generated by FITD is always one on those topologies, so we omit the distribution of detour stretches generated by FITD. We find that FIR has long detour stretches. Long detour stretches degrade the network performance, and even increase the operational cost, which is undesirable to network operators.

TABLE 3. ENTRY SIZE FOR A DESTINATION (IN BITS)

		AS 1221	AS 1239	AS 1755	AS 3257	AS 3967	AS 6461
FITD-basic	Avg	96	128	160	128	160	160
	Max	288	416	384	480	416	448
FITD-extended	Avg	131	166	196	164	196	197
	Max	338	493	427	541	460	500

C. Forwarding Table Size

For FITD-basic, a router needs to save one primary next hop and several backup next hops for each destination. For FITD-extended, a router needs to save one primary next hop, one virtual primary next hop, several backup next hops and adjacent virtual cross links for each destination. As described in Section III, the forwarding information for each destination is saved in SRAM, which is cheap even in large volumes. We suppose that one next hop needs four bytes, which is 32 bits. We now describe how to efficiently save virtual cross links. Suppose a router s has m neighbors, we use $nei[i]$ to denote i^{th} neighbor of the router s. The router only needs to use m bits denoted by $mark[1, 2, ...m]$ to save the virtual cross links. On each topology, we calculated the memory size used by the entry for each destination. We use $num_{(s,d)}$ to denote the memory size used by the entry for the destination router d on the given topology. We use $cases$ to denote the set of our evaluation cases, where $cases = \{(s, d)|s \in V, d \in V, s \neq d\}$. We calculated the average entry size for a destination, which is $\frac{\sum_{(s,d)\in cases} num_{(s,d)}}{|cases|}$. We also calculated the maximum entry size for a destination, which is $\max_{(s,d)\in cases} num_{(s,d)}$. The results are shown in Table 3.

The unit is bit. The column header denotes names of the topologies. The row header denotes the methods. Avg denotes the average entry size for a destination. Max denotes the maximum entry size for a destination. We now estimate the size that FITD uses in the worst case. As shown in Table 3, an entry needs at most 541 bits for a destination. Up to now, the number of IPv4 prefixes is less than 600K [18]. Thus, on those topologies, the SRAM size that FITD uses is at most $541 \times 600K$ bits $< 350M$ bits. 350M bits of space is fully acceptable by SRAM. In the average cases, the SRAM space that FITD uses will be much less than 350M bits.

6. Related Work

An extensive list of methods have been proposed to speed up recovery from network failures for IP networks. A large body of failure-recovery methods, such as [3] [4] [5], are proposed based on tunneling or marking. We now mainly describe failure-recovery methods using neither tunneling nor marking, which are LFA [19] , U-turn [20], FIR [7] , ESCAP [9] , DisPath [10] and RPFP [8] . LFA and U-turn cannot guarantee recovery from all single-link failures even if detours for the failures exist. The main idea of LFA [19] is to reroute the affected traffic through the pre-installed loop-free next hops after a failure is detected. Atlas et al. later proposed U-turn [20] to protect more single-link failure cases as compared to LFA by using loop-free next hops of neighbors. We now introduce FIR, ESCAP, DisPath and RPFP, which guarantee recovery from all single-link failures only if detours for the failures exist. Lee et al. propose FIR [7] . For FIR, each interface is associated with a forwarding table. FIR invokes the corresponding forwarding table according to incoming interfaces. In [21] , FIR is expanded to deal with node failures. In [22], FIR is expanded to support networks with asymmetric link weights. Xi et al. propose ESCAP [9], which can work for any primary routing not limited to shortest-path routing. Antonakopoulos et al. propose DisPath [10], which needs less memory space than FIR. In [8], Zhang et al. propose RPFP, which supports partial deployment. Note that ESCAP, DisPath and RPFP cannot work when the primary routing supports Equal-Cost Multi-Path (ECMP) routing while FITD can.

Moreover, some other papers are related to failure recovery, but they are not the focus of this paper as follows. In [23], a load-balancing algorithm is proposed for failure recovery in IP networks. In reality, load balancing is a single-objective optimization issue while link weights are set to optimize multiple objectives. Of course, the load-balancing algorithms can also be used for FITD. A lot of papers [24] [25] [26] show that the near-optimal, even optimal load balancing can be achieved by only using shortest paths. How to balance load along multiple equal-cost shortest detours is not the focus of this paper. In addition, there are many failure recovery methods proposed for the inter-domain routing, such as [27] [28] [29] . There are many failure recovery methods proposed for other types of networks. For example, [30] is proposed for the MPLS network.

7. Conclusions

In the previous context of this paper, we assume that FITD works in OSPF/IS-IS networks, where the routing is shortest-path based (Assumption 1). The motivation of Assumption 1 is to ensure that each router can independently compute backup next hops in a distributed way without modifying OSPF/IS-IS control-plane messages. If the routing towards a destination is a Directed Acyclic Graph (DAG), where some routing paths are not the shortest, we can use the centralized control way to compute and install backup next hops based on the DAG or make each router know the DAG by adding new control-plane messages. For the loop-prevention issue, the existing loop-prevention mechanisms [10] [31] [32] can also be applied to FITD, so we do not address this issue in this paper.

In this paper, we propose a failure inference approach to shortening traffic detours named as FITD. FITD does not use explicit failure notification, and can infer which link fails based on traffic information. FITD guarantees recovery from any single-link failure if a detour for the failed link exists. In particular, for networks with symmetric link weights, FITD guarantees to generate shortest detours for single-link failure.

In the future we will describe how to further extend FITD-extended by cancelling Assumption 1 in detail. FITD provides a good idea of inferring failed links without using explicit failure notification. And we will study how to make FITD deal with multiple link failures and how to make FITD achieve the shortest detours in networks with asymmetric link weights.

Acknowledgments

This work is supported by the National Science Foundation of China (No.61472213). Jun Bi is the corresponding author.

References

[1] A. Markopoulou, G. Iannaccone, S. Bhattacharyya, C.-N. Chuah, and C. Diot, "Characterization of failures in an IP backbone," in *IEEE INFOCOM 2004*, vol. 4, March 2004, pp. 2307–2317 vol.4.

[2] P. Francois, C. Filsfils, J. Evans, and O. Bonaventure, "Achieving subsecond IGP convergence in large IP networks," *SIGCOMM Comput. Commun. Rev.*, vol. 35, no. 3, pp. 35–44, Jul. 2005.

[3] 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. New York, NY, USA: ACM, 2007, pp. 241–252.

[4] S. P. S. Bryant, C. Filsfils and M. Shand, "IP fast reroute using tunnels." Internet IETF Draft draft-bryant-ipfrr-tunnels-03, Nov. 2007.

[5] S. P. S. Bryant, M. Shand and M. Shand, "IP fast reroute using not-via address." Internet IETF Draft draft-bryant-shand-ipfrr-notvia-addresses, May 2013.

[6] W. Simpson, "IP in IP tunneling." RFC 1853, Oct. 1995.

[7] S. Lee, Y. Yu, S. Nelakuditi, Z.-L. Zhang, and C.-N. Chuah, "Proactive vs reactive approaches to failure resilient routing," in *IEEE INFOCOM 2004*, vol. 1, March 2004, p. 186.

[8] B. Zhang, J. Wu, and J. Bi, "RPFP: IP fast reroute with providing complete protection and without using tunnels," in *2013 IEEE/ACM 21st International Symposium on Quality of Service (IWQoS)*, June 2013, pp. 1–10.

[9] K. Xi and H. J. Chao, "IP fast rerouting for single-link/node failure recovery," in *Broadband Communications, Networks and Systems, 2007. BROADNETS 2007. Fourth International Conference on*, Sep. 2007, pp. 142–151.

[10] S. Antonakopoulos, Y. Bejerano, and P. Koppol, "A simple IP fast reroute scheme for full coverage," in *2012 IEEE 13th International Conference on High Performance Switching and Routing*, June 2012, pp. 15–22.

[11] J. L. Sobrinho, "Algebra and algorithms for QoS path computation and hop-by-hop routing in the Internet," *IEEE/ACM Transactions on Networking*, vol. 10, no. 4, pp. 541–550, Aug 2002.

[12] B. Fortz and M. Thorup, "Internet traffic engineering by optimizing ospf weights," in *Proceedings IEEE INFOCOM 2000. Conference on Computer Communications. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (Cat. No.00CH37064)*, vol. 2, 2000, pp. 519–528 vol.2.

[13] The appendix of this paper. [Online]. Available: https://github.com/eipiplus/FITD-proof/blob/master/appendix.pdf

[14] J. Postel, "Internet protocol." RFC 791, Sep. 1981.

[15] A. Basu and J. Riecke, "Stability issues in OSPF routing," in *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, ser. SIGCOMM '01. New York, NY, USA: ACM, 2001, pp. 225–236.

[16] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. the second editor, 2001.

[17] The router-level ISP topologies from rocketfuel project.

[18] "Number of FIB prefixes." [Online]. Available: http://www.cidr-report. org/as2.0/

[19] A. Atlas, A. Zinin, R. Torvi, G. Choudhury, C. Martin, B. Imhoff, and D. Fedyk, "Basic specification for IP fast-reroute: loop-free alternates." RFC 5287, Sep. 2008.

[20] A. Atlas, "U-turn alternates for IP/LDP fast-reroute." RFC 5286, Feb. 2006.

[21] Z. Zhong, Z. Nelakuditi, Y. Yu, S. Lee, J. Wang, and C. N. Chuah, "Failure inferencing based fast rerouting for handling transient link and node failures," in *Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies.*, vol. 4, March 2005, pp. 2859–2863 vol. 4.

[22] J. Wang and S. Nelakuditi, "IP fast reroute with failure inferencing," in *Proceedings of the 2007 SIGCOMM Workshop on Internet Network Management*, ser. INM '07. New York, NY, USA: ACM, 2007, pp. 268–273.

[23] M. Zhang and B. Liu, "Traffic engineering for proactive failure recovery of IP networks," *Tsinghua Science and Technology*, vol. 16, no. 1, pp. 55–61, Feb 2011.

[24] Y. Wang, Z. Wang, and L. Zhang, "Internet traffic engineering without full mesh overlaying," in *Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No.01CH37213)*, vol. 1, 2001, pp. 565–571 vol.1.

[25] K. Nmeth, A. Krsi, and G. Rtvri, "Optimal OSPF traffic engineering using legacy Equal Cost Multipath load balancing," in *2013 IFIP Networking Conference*, May 2013, pp. 1–9.

[26] A. Sridharan, R. Guérin, and C. Diot, "Achieving near-optimal traffic engineering solutions for current OSPF/IS-IS networks," *IEEE/ACM Trans. Netw.*, vol. 13, no. 2, pp. 234–247, Apr. 2005.

[27] P. Francois, O. Bonaventure, B. Decraene, and P. A. Coste, "Avoiding disruptions during maintenance operations on BGP sessions," *IEEE Transactions on Network and Service Management*, vol. 4, no. 3, pp. 1–11, Dec 2007.

[28] O. Bonaventure, C. Filsfils, and P. Francois, "Achieving sub-50 milliseconds recovery upon BGP peering link failures," *IEEE/ACM Transactions on Networking*, vol. 15, no. 5, pp. 1123–1135, Oct 2007.

[29] N. Kushman, S. Kandula, D. Katabi, and B. M. Maggs, "R-BGP: Staying connected in a connected world," in *Proceedings of the 4th USENIX Conference on Networked Systems Design & Implementation*, ser. NSDI'07. Berkeley, CA, USA: USENIX Association, 2007, pp. 25–25.

[30] V. Sharma and F. Hellstrand, "Framework for Multi-Protocol Label Switching (MPLS)-based Recovery." RFC 3469, Feb. 2003.

[31] P. Francois, O. Bonaventure, M. Shand, S. Bryant, and C. Filsfils, "Loop-free convergence using oFIB." IETF Internet-Draft, May 2013.

[32] P. Francois and O. Bonaventure, "Avoiding transient loops during the convergence of link-state routing protocols," *IEEE/ACM Transactions on Networking*, vol. 15, no. 6, pp. 1280–1292, Dec 2007.