Failure Insensitive Routing for Ensuring Service Availability ⋆

Srihari Nelakuditi[1], Sanghwan Lee[2], Yinzhe Yu[2], and Zhi-Li Zhang[2]

[1] Dept. of Computer Science & Engineering,
University of South Carolina,
Columbia, SC 29201, USA
srihari@cse.sc.edu
[2] Dept. of Computer Science & Engineering,
University of Minnesota,
Minneapolis, MN 55414, USA
{sanghwan,yyu,zhzhang}@cs.umn.edu

Abstract. Intra-domain routing protocols employed in the Internet route around failed links by having routers detect adjacent link failures, exchange link state changes, and recompute their routing tables. Due to several delays in detection, propagation and recomputation, it may take tens of seconds to minutes after a link failure to resume forwarding of packets to the affected destinations. This discontinuity in destination reachability adversely affects the quality of continuous media applications such as Voice over IP. Moreover, the resulting service unavailability for even a short duration could be catastrophic in the world of e-commerce. Though careful tuning of the various parameters of the routing protocols can accelerate convergence, it may cause instability when the majority of the failures are transient. To improve the failure resiliency without jeopardizing the routing stability, we propose a *local rerouting* based approach called *failure insensitive routing*. Under this approach, upon a link failure, adjacent router *suppresses* global updating and instead initiates local rerouting. All other routers *infer* potential link failures from the packet's incoming interface, *precompute* interface specific forwarding tables and route around failed links *without explicit* link state updates. We demonstrate that the proposed approach provides higher service availability than the existing routing schemes.

1 Introduction

Link state routing protocols such as OSPF and IS-IS are the most widely used protocols for intra-domain routing in today's Internet. Using these protocols, routers exchange changes in link state, recompute their routing tables, and thus respond to link and node failures in the network by routing around them. However, several recent studies [1, 5, 7] have reported that rerouting after a link failure takes tens of seconds to minutes. During this period, some destinations would be unreachable and the corresponding services

⋆ This work is partly supported by National Science Foundation Grants CAREER Award ANI-9734428, ANI-0073819, and ITR ANI-0085824. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the National Science Foundation.

unavailable. This discontinuity in routing adversely affects the quality of continuous media applications such as Voice over IP. Furthermore, downtime of even a few seconds could significantly impact the reputation and the profitability of a company in the world of e-commerce. Moreover, it has been observed [7] that link failures are fairly common in the day to day operation of a network due to various causes such as maintenance, faulty interfaces, and accidental fiber cuts. Hence, there is a growing demand for ensuring destination reachability and thus service continuity even in the presence of link failures.

There have been some modifications proposed [1, 2] for accelerating the convergence of link state routing protocols. But the recipe involves tuning several delays associated with link failure detection, link state propagation and routing table recomputation. Furthermore, it is not a suitable solution for handling transient failures. It has been found [7] that majority of the link failures are short-lived with around half of the failures lasting less than a minute. In such a scenario, it is not prudent to disseminate these link state changes globally and recompute routing tables at each router in the network. Instead, it is much more appropriate to perform local rerouting and trigger global updating and recomputation only if the link failure persists for a longer duration. Such a local rerouting approach can recover promptly from failures trading off optimality of routing for continuity of forwarding. Our objective is to devise a stable and robust routing scheme that ensures continuous loop-free forwarding of packets to their destinations regardless of the various delays in link state propagation and routing table recomputation.

We propose a local rerouting based approach for failure resiliency which we refer to as *failure insensitive routing* (FIR). Under FIR, when a link fails, adjacent nodes suppress global updating and instead initiate local rerouting of packets that were to be forwarded through the failed link. Though other nodes are not explicitly notified of the failure, they *infer* it from the packet's *flight*. When a packet arrives at a node through an *unusual* interface (through which it would never arrive had there been no failure), corresponding potential failures can be inferred and the next hop chosen avoiding those links. This way under FIR, the next hop for a packet is determined based on not only the destination address but also the incoming interface. Note that such *interface specific forwarding* is very much feasible with current router architectures as they anyway maintain a forwarding table at each line card of an interface for lookup efficiency. These interface specific forwarding tables can be *precomputed* since inferences about the link failures can be made in advance. Thus with the FIR approach, when a link fails, only the nodes adjacent to it locally reroute packets to the affected destinations and all the other nodes simply forward packets according to their precomputed interface specific forwarding tables without being explicitly aware of the failure. Once the failed link comes up again, original forwarding tables are locally restored and forwarding resumes over the recovered link as if nothing ever happened. This approach decouples destination reachability and routing stability by handling transient failures locally and notifying only persistent failures globally. Essentially with FIR, in the presence of link failures, packets get locally rerouted (possibly along suboptimal paths) without getting caught in a loop or dropped till the new shortest paths are globally recomputed.

There are several benefits in employing FIR. First, it can be deployed without altering the destination based forwarding paradigm used in the current Internet. Only the traditional interface independent routing table computation algorithm needs to be replaced with an FIR algorithm for computing interface dependent forwarding tables. Second, reachability of destinations does not depend on tuning of the various parameters associated with link failure propagation and routing table recomputation. Thus FIR improves the service availability without jeopardizing the routing stability. Third, under FIR approach local rerouting happens only during the time a link failure is suppressed, i.e., not reflected globally. But once all the routers have the same consistent view of the network, forwarding under FIR would be no different from traditional routing. So FIR can be used in conjunction with any other mechanism for engineering traffic. Finally, FIR increases network reliability and obviates the need for expensive and complex layer 2 protection schemes. Essentially, the FIR approach is about preparing for failures instead of reacting to them.

We make the following contributions in this paper. We propose a mechanism for facilitating prompt local rerouting. We present an efficient algorithm that computes interface specific forwarding tables for dealing with single link failures in $O(|\mathcal{E}| \log^2 |\mathcal{V}|)$ time, where $\mathcal{V}$ is the set of nodes and $\mathcal{E}$ is the set of edges. We demonstrate that by preparing for single link failures, most of the simultaneous failures can also be handled and the service availability can be improved by an order of magnitude. We describe an incremental algorithm for forwarding table computation that requires $O(D^2|\mathcal{V}|)$ space, where D is the network diameter, for remembering the intermediate steps of the previous computation but takes on average less than $O(|\mathcal{E}| \log |\mathcal{V}|)$ time. We argue that with its resiliency and stability, FIR is a better alternative to the existing routing schemes.

The rest of the paper is organized as follows. Section 2 introduces our FIR approach for failure resiliency. Efficient algorithms for computing interface specific forwarding tables are described in Section 3. Section 4 presents the results of our evaluation of the performance of FIR. The related work is discussed in Section 5. Finally, Section 6 concludes the paper.

2 Failure Insensitive Routing

The fundamental issue in designing a local rerouting scheme is the avoidance of forwarding loops. A straightforward local recomputation of new shortest paths without the failed link by the adjacent node could result in a loop since other nodes are not aware of the failure and their routing tables do not reflect the failure. We propose to address this looping problem by forwarding a packet based on its incoming interface. This enables a router to *infer* failures when a packet arrives through an *unusual* interface due to local rerouting. These inferences about link failures can be made in advance and interface specific forwarding tables can be *precomputed* avoiding the potentially failed links. This way when a link fails, only the adjacent nodes reroute packets that were to be forwarded through the failed link. All other nodes simply forward packets according to their precomputed interface specific forwarding tables without being explicitly aware of the failure. We refer to this approach as *failure insensitive routing* (FIR). In the fol-

lowing, using an example topology, we illustrate how packets get forwarded under FIR and how these forwarding tables are computed.

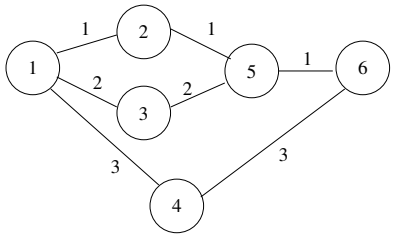


Fig. 1. Topology used for the illustration of the FIR approach

2.1 Forwarding under FIR

Consider the topology shown in Figure 1 where each link is labeled with its weight. The corresponding shortest path routing entries at each node to destination node 6 are shown in Figure 2. First, we point out the problem with the conventional routing in case of a link failure. Suppose link 2–5 is down. When node 2 recomputes its routing table, it will have 1 as the next hop to reach 6 as shown in Figure 2. If only node 2 recomputes its entries while others are not notified or still in the process of recomputing their entries, then packets from 1 to 6 get forwarded back and forth between nodes 2 and 1. This shows that using conventional forwarding tables, local rerouting is not viable as it causes forwarding loops.

node	1	2	3	4	5
next	2	5	5	6	6

node	1	2	3	4	5
next hop	2	**1**	5	6	6

Fig. 2. Routing entries: *before* and *after* local recomputation by node 2

Under FIR, forwarding loops are avoided by inferring link failures from the packet's incoming interface. When a packet with destination 6 arrives at 1 from 2, node 1 can sense that some link must have failed. Otherwise, based on shortest path routing, node 2 should never forward to 1, a packet destined for 6. Node 2 would forward packets for 6 to node 1 if the link 2–5 is down. Same is true even when 5–6 is down. So when a packet for 6 arrives at 1 from 2, node 1 can infer that one or both of these links are down. Since node 1 is not explicitly notified of the failures, it can ensure that the packet reaches 6 by forwarding it to 4 avoiding both the potentially failed links 2–5 and 5–6. That is why in Figure 3, a packet arriving at node 1 with destination 6 through neighbor node 2 is forwarded to 4 while it is forwarded to 2 if it arrives through the other two neighbors. Such interface specific forwarding makes it possible to perform local rerouting.

node	1			2		3		4		5		
prev	2	3	4	1	5	1	5	1	6	2	3	6
next	4	2	2	5	1	5	1	6	-	6	6	-

node	1			2		3		4		5		
prev	2	3	4	1	5	1	5	1	6	2	3	6
next	4	2	2	**1**	-	5	1	6	-	6	6	-

Fig. 3. Interface specific forwarding entries: *before* and *after* local recomputation by node 2

Let us again consider the case of link $2{-}5$ going down. Node 2 recomputes its forwarding table entries as shown in Figure 3. So a packet from 2 to 6 takes the route $2{\rightarrow}1{\rightarrow}4{\rightarrow}6$ when the link 2–5 is down. Since node 1 is not aware of the failure, a packet from 1 to 6 gets forwarded to 2 which reroutes it back to 1. Node 1 then forwards the packet to 4 according to its entry at the interface with previous hop 2. This way, packets from 1 to 6 traverse the path $1{\rightarrow}2{\rightarrow}1{\rightarrow}4{\rightarrow}6$. Note that though node 1 appears twice in the path, it doesn't constitute a loop. With interface specific forwarding, a packet would loop only if it traverses the same link in the same direction twice. Thus using interface specific forwarding tables, FIR avoids looping and provides local rerouting.

It should be noted that FIR adheres to conventional destination based forwarding paradigm though it has different forwarding table at each interface. While FIR requires that the next hop for a packet is determined based on its previous hop, it is very much feasible with the current router architectures as they anyway maintain a forwarding table at each line card of an interface for lookup efficiency. The only deviation is that unlike in the current routers with the same forwarding table at each interface, with the FIR approach these tables are different. However, the forwarding process remains the same — when a packet arrives at an incoming interface, the corresponding forwarding table is looked up to determine the next hop and the outgoing interface.

2.2 Forwarding Table Computation

The forwarding process under FIR is essentially the same as it is under the conventional routing. The key difference is in the way interface specific forwarding tables are computed. The computation of the forwarding table entries of an interface involves identifying a set of links whose individual or combined failure causes a packet to arrive at the node through that interface. We refer to these links as *key links* and denote by $\mathcal{K}^d_{j\rightarrow i}$ the set of links which when one or more down cause packets with destination d to arrive at node i from node j. Note that this key link set is empty, i.e., $\mathcal{K}^d_{j\rightarrow i} = \emptyset$ if node i is anyway the next hop along the shortest path from j to d without any link failures. For the topology in 1, $\mathcal{K}^6_{2\rightarrow 1} = \{2\text{–}5, 5\text{–}6\}$ and $\mathcal{K}^6_{3\rightarrow 1} = \{3\text{–}5\}$ while $\mathcal{K}^6_{1\rightarrow 2} = \emptyset$ as explained below.

Consider the node 1. The next hop along the shortest path from node 1 to reach 6 is 2, i.e., $\mathcal{K}^6_{1\rightarrow 2} = \emptyset$. So if all the links are up, node 1 should never receive from 2 a packet destined for 6. However, if the link $2{-}5$ is down, node 2 would forward packets with destination 6 to node 1. Similarly when the link 5–6 is down, packets from 5 to 6 would traverse the path $5{\rightarrow}2{\rightarrow}1{\rightarrow}4{\rightarrow}6$. So from the arrival of a packet with destination 6 from neighbor node 2, node 1 can infer that one or both of the links 2–5 and 5–6 are down,

i.e., $\mathcal{K}^6_{2\rightarrow 1} = \{2\text{–}5, 5\text{–}6\}$. Similarly, node 1 would receive a packet for the destination 6 through 3 when the link 3–5 is down. In the other case when link 5–6 is down, packets arrive at node 1 through 2 and not through 3 since from 5 to 6 the (recomputed) shortest path would be 5→2→1→4→6. Hence from arrival of packets with destination 6 through node 3, node 1 infers that only link 3–5 is down, i.e., $\mathcal{K}^6_{3\rightarrow 1} = \{3\text{–}5\}$.

$2 \rightarrow 1$					
dest	2	3	4	5	6
next hops	-	3	4	3	4

$3 \rightarrow 1$					
dest	2	3	4	5	6
next hops	2	-	4	2	2

$4 \rightarrow 1$					
dest	2	3	4	5	6
next hops	2	3	-	2	2

Fig. 4. Forwarding tables at node 1

Once the key links are determined, it is straightforward to compute the interface specific forwarding tables. Let $\mathcal{E}$ be the set of all links in the network. Suppose $\mathcal{R}^d_i(\mathcal{X})$ represents the set of next hops from i to d given the set of links $\mathcal{X}$. Let $\mathcal{F}^d_{j\rightarrow i}$ denote the forwarding table entry, i.e., the set of next hops to d for packets arriving at i through the interface associated with neighbor j. This entry can be computed using Dijkstra's Shortest Path First (SPF) algorithm after excluding the links in the set $\mathcal{K}^d_{j\rightarrow i}$ from the set of all links $\mathcal{E}$. Thus,

$$\mathcal{F}^d_{j\rightarrow i} = \mathcal{R}^d_i(\mathcal{E} \setminus \mathcal{K}^d_{j\rightarrow i})$$

The forwarding tables corresponding to node 1 of Figure 1 are shown in Figure 4. Given that $\mathcal{K}^6_{2\rightarrow 1} = \{2\text{–}5, 5\text{–}6\}$, the shortest path from 1 to 6 without those links be 1→4→6. Therefore, packets destined for 6 arriving at 1 through 2 are forwarded to next hop 4. On the other hand, the next hop for packets to destination 5 arriving through 2 is set to 3 since $\mathcal{K}^5_{2\rightarrow 1} = \{2\text{–}5\}$. The other entries are also determined similarly. Once the forwarding tables are computed, packets arriving through an interface are forwarded in the usual manner by looking up the table corresponding to that interface. We can prove [12] that with forwarding tables computed thus, when no more than one link fails, FIR always finds a loop-free path to a destination if such a path exists.

We reiterate that these inferences about potential link failures are made *not on the fly* but in advance and forwarding tables are precomputed according to these inferences. Furthermore, packets are forwarded according to their destination addresses only. In other words, FIR does not require any changes to the existing forwarding plane, making it amenable for ready deployment.

2.3 Local Recomputation of Forwarding Tables

The forwarding tables computed as explained above help perform local rerouting without any global recomputation of routing and forwarding tables. Only the nodes adjacent to a failed link have to recompute their entries. However, if the local recomputation takes significant time, then there would not be substantial savings due to this approach

over conventional global updating based approach. Fortunately, we do not have to compute these tables from scratch. It is possible to locally recompute the forwarding tables in negligible amount of time by maintaining what we refer to as *backwarding table* for each interface.

$1 \rightarrow 2$					
dest	2	3	4	5	6
back hops	3	4	3	3	3

$1 \rightarrow 3$					
dest	2	3	4	5	6
back hops	4	2	-	4	4

$1 \rightarrow 4$					
dest	2	3	4	5	6
back hops	-	-	2	-	-

Fig. 5. Backwarding tables at node 1

When an interface is down, its backwarding table can be used to reroute packets that were to be forwarded through that interface. The entries in this table, denoted by $\mathcal{B}^d_{i \rightarrow j}$, give the set of alternate next hops, referred to as *back hops*, from node i for forwarding a packet with destination d when the interface or the link to the usual next hop node j is down. The backwarding table entries can also be precomputed similar to forwarding table entries once the key links are identified as follows:

$$\mathcal{B}^d_{i \rightarrow j} \Leftarrow \mathcal{R}^d_i(\mathcal{E} \setminus \mathcal{K}^d_{i \rightarrow j} \setminus i\text{–}j)$$

Essentially we exclude all the links that would cause the packet to exit from the interface of i to j and also the link i–j itself in computing the back hops. When preparing for at most single link failures, this amounts to

$$\mathcal{B}^d_{i \rightarrow j} \Leftarrow \mathcal{R}^d_i(\mathcal{E} \setminus i\text{–}j)$$

The backwarding table entries for node 1 of the topology in Figure 1 are shown in Figure 5. Let us look at the entries for the interface $1 \rightarrow 2$. It is clear that when the link 1–2 is down, packets to destinations 2, 5 and 6 be rerouted to 3 since the shortest path to these nodes without 1–2 is through 3. But, it may not be obvious why the next hops for destinations 3 and 4 are 4 and 3 respectively. Consider the entry of 3. The corresponding set of key links $\mathcal{K}^3_{1 \rightarrow 2}$ is $\{1\text{–}3\}$, i.e., a packet with destination 3 is forwarded from 1 to 2 only if $\{1\text{–}3\}$ is down. So when $\{1\text{–}2\}$ is also down, the next best path is through 4. Similarly $\mathcal{B}^4_{1 \rightarrow 2}$ is 3. Now let us turn our attention to the backwarding table in Figure 5 for the interface 1–4. According to these entries, when link 1–4 is down, packets to 4 get rerouted to 2 and packets to any other destination are simply discarded as they are not reachable. This is because packets to other destinations are forwarded to 4 only when certain other links are also down. For example, $\mathcal{K}^6_{1 \rightarrow 4} = \{2\text{–}5, 5\text{–}6\}$ and when link 1–4 also fails, node 6 becomes unreachable from node 1.

By employing interface specific forwarding and backwarding tables, we can eliminate the delay due to any dynamic recomputation and reroute packets without any interruption even in the presence of link failures. The downside is that the deployment of backwarding tables requires changes to the forwarding plane. When an interface is

down, the corresponding backwarding table needs to be looked up to reroute the packet through another interface. This necessitates change in the router architecture, the cost of which we are not in a position to assess. To avoid altering the forwarding plane, we propose to maintain the backwarding tables in the control plane and recompute the forwarding tables as follows. Suppose the failed link is i–k and the new forwarding tables are denoted by $\tilde{\mathcal{F}}$. Then the forwarding table entry of destination d for $j \rightarrow i$ interface, where $j \neq k$, is computed as follows:

$$\tilde{\mathcal{F}}^d_{j \rightarrow i} = \begin{array}{ll} \mathcal{F}^d_{j \rightarrow i} \setminus k \cup \mathcal{B}^d_{i \rightarrow k} & \text{if } k \in \mathcal{F}^d_{j \rightarrow i} \\ \mathcal{F}^d_{j \rightarrow i} & \text{otherwise} \end{array}$$

The above expression takes into account the possibility of multiple next hops along equal cost paths to a destination. A simplified expression for single path routing would be

$$\tilde{\mathcal{F}}^d_{j \rightarrow i} = \begin{array}{ll} \mathcal{B}^d_{i \rightarrow k} & \text{if } \mathcal{F}^d_{j \rightarrow i} = k \\ \mathcal{F}^d_{j \rightarrow i} & \text{otherwise} \end{array}$$

Essentially, only those entries in the forwarding tables that have k as the next hop are reset according to the backwarding table associated with k. Thus, using the backwarding tables, in case of an adjacent link failure, a node quickly recomputes the forwarding tables locally and promptly resumes forwarding.

2.4 Summary of the FIR Scheme

We now summarize the operation of the FIR scheme. Each node i under FIR maintains a forwarding table $\mathcal{F}_{j \rightarrow i}$ per each neighbor j, and a backwarding table $\mathcal{B}_{i \rightarrow j}$ per each neighbor j. $\mathcal{F}_{j \rightarrow i}$ is used to forward packets arriving at i through neighbor j. $\mathcal{B}_{i \rightarrow j}$ is needed for locally recomputing the forwarding tables of i when the link i–j is down.

Suppose the failure of the link i–j is detected by node i at time t_{down}. Then node i locally recomputes its forwarding tables and performs local rerouting of the packets that were to be forwarded to j. If the failure persists for a preset duration T_{down}, then a global link state update is triggered at $t_{down}+T_{down}$ and forwarding tables at all routers are recomputed. During the time period between t_{down} and $t_{down} + T_{down}$, the link failure update is said to be *suppressed* since all the nodes other than the adjacent nodes i and j are not aware of the failure. Local rerouting is in effect when and only when there exists a suppressed failure event.

After sometime, suppose at time t_{up}, link i–j comes up. Then the action taken by node i depends on whether the failure event is being suppressed or not. If the failure event is being suppressed, original forwarding tables are locally restored and forwarding resumes over the recovered link as if nothing ever happened. Otherwise, the link is observed for a preset period T_{up} and if it stays up, then at time $t_{up} + T_{up}$, a global update is triggered announcing that the link is up. This way, failures of short duration are handled locally while persistent failures are updated globally. When the failures are transient, FIR not only improves reachability but also reduces overhead.

3 Efficient FIR Algorithms

The process of forwarding and backwarding table computation, as explained in the previous section, involves determining a set of key links for each interface of a node. In this section, we develop efficient algorithms for identifying key links. We show that by saving some intermediate steps of the previous computation, forwarding and backwarding tables can be obtained incrementally in time less than an SPF computation.

The algorithms described here assume that all the links are point to point, and bidirectional with equal weight in both directions, which is generally true for the backbone networks. It is also assumed that no more than one link fails simultaneously. There are several reasons for concentrating on singe link failures. First, it has been observed [13] that failure of a single link is more common than simultaneous multiple link failures. Second, under FIR a failure is *suppressed* for a certain duration and if it persists beyond that time, a global update is triggered. Only simultaneous suppressed failures could pose problem for FIR. The possibility of multiple simultaneous suppressed failures happening in the network is rare considering that suppress interval would be in the order of a minute. Third, as we demonstrate in the next section, by preparing just for single link failures, FIR can deal with the majority of the multiple simultaneous failures also.

3.1 Available Shortest Path First

We now present an algorithm for determining key links and computing forwarding and backwarding tables. We refer to this procedure as *available shortest path first* (ASPF) since it computes shortest paths excluding the unavailable (potentially failed) links. The notation used here and the rest of the paper is listed in a table along with all the algorithms. A straightforward method for determining key links would be to invoke Dijkstra's SPF procedure once per each link in the network. Its time complexity would be $O(|\mathcal{E}|^2 \log |\mathcal{V}|)$, which is too high to be practical. Fortunately, it is possible to compute key links more efficiently for single link failures in $O(|\mathcal{E}| \log^2 |\mathcal{V}|)$ time based on the following observations:

- Only the failure of a link along the shortest path from node i to a destination d may require *unusual* forwarding of packets to d arriving at i. Otherwise packets are forwarded simply along the usual shortest path. As per this *revised* definition of key links, for the topology in Figure 1, $\mathcal{K}^6_{3\to 1} = \emptyset$ instead of the original set $\{3\text{–}5\}$ since 3–5 is not along the shortest path from 1 to 6. This new interpretation limits the search space for key links to links in SPT rooted at i. Given that the number of links in a tree would be $O(|\mathcal{V}|)$, search space is reduced from $O(|\mathcal{E}|)$ to $O(|\mathcal{V}|)$.
- A packet needs to be forwarded to an unusual next hop only when it arrives back from a usual next hop. In other words, an edge e is included in $\mathcal{K}^d_{j\to i}$ only if j *is a next hop from* i to d *with* e, and i *is a next hop from* j to d *without* e. This helps segregate nodes and links based on the next hops from i, i.e., $\mathcal{K}^d_{j\to i}$ is $\emptyset$ if j is not a usual next hop from i to d. Also, an edge e is not a member of $\mathcal{K}^d_{j\to i}$ if e is not in the subtree below j of the SPT of i. Therefore, the key links of all the interfaces together can be determined within $O(|\mathcal{V}|)$ SPT computations.

Notation	
$\mathcal{V}$	set of all vertices
$\mathcal{E}$	set of all edges
$\mathcal{N}_i$	set of neighbors of node i
W_e	weight of edge e
$\|\mathcal{N}\|$	avg no. of neighbors of a node
D	diameter of the network
$\mathcal{R}_i^d$	set of next hops from i to d
$\mathcal{F}_{j\rightarrow i}^d$	set of next hops from $j\rightarrow i$ to d.
$\mathcal{B}_{j\rightarrow i}^d$	set of back hops from $i\rightarrow j$ to d.
$\mathcal{K}_{j\rightarrow i}^d$	key links from $j\rightarrow i$ to d.
$\mathcal{T}_i$	shortest path tree rooted at i
$\mathcal{T}_i^e$	SPT of i without edge e
$C(k,\mathcal{T})$	cost to node k from root of $\mathcal{T}$
$P(k,\mathcal{T})$	parents of node k in tree $\mathcal{T}$
$N(k,\mathcal{T})$	next hops to k from root of $\mathcal{T}$
$S(k,\mathcal{T})$	subtree below k in tree $\mathcal{T}$
$V(\mathcal{T})$	set of all vertices in tree $\mathcal{T}$
$E(\mathcal{T})$	set of all edges in tree $\mathcal{T}$
$\mathcal{Q}$	priority queue

Algorithm 1 : $\mathbf{ASPF}(i)$

for all $j \in \mathcal{N}_i$ **do**
  for all $d \in \mathcal{V}$ **do**
    $\mathcal{K}_{j\rightarrow i}^d \Leftarrow \emptyset$

$\mathcal{T}_i \Leftarrow \texttt{SPF}(i, \mathcal{V}, \mathcal{E})$
for all $j \in \mathcal{N}_i$ *and* $j \in N(j, \mathcal{T}_i)$ **do**
  $\mathcal{T}_j \Leftarrow \texttt{SPF}(j, \mathcal{V}, \mathcal{E})$
  $\mathcal{E}' \Leftarrow E(S(j, \mathcal{T}_i))$
  for all $u\rightarrow v \in \mathcal{E}'$ **do**
    $\mathcal{V}' \Leftarrow V(S(v, \mathcal{T}_i))$
    $\mathcal{T}_j^{u-v} = \texttt{ISPF}(\mathcal{T}_j, \mathcal{V}', \{u\text{–}v\})$
    for all $d \in \mathcal{V}'$ **do**
      if $i \in N(d, \mathcal{T}_j^{u-v})$ **then**
        $\mathcal{K}_{j\rightarrow i}^d \Leftarrow \mathcal{K}_{j\rightarrow i}^d \cup \{u\text{–}v\}$

return TABLES(i)

Algorithm 2 : $\mathbf{TABLES}(i)$

for all $j \in \mathcal{V}$ **do**
  $\mathcal{T}_i^{i-j} \Leftarrow \texttt{ISPF}(\mathcal{T}_i, \mathcal{V}, \{i\text{–}j\})$
  for all $d \in \mathcal{V}$ **do**
    $\mathcal{B}_{i\rightarrow j}^d \Leftarrow N(d, \mathcal{T}_i^{i-j})$
    if not exists $\mathcal{T}_i^{\mathcal{K}_{j\rightarrow i}^d}$ **then**
      $\mathcal{T}_i^{\mathcal{K}_{j\rightarrow i}^d} \Leftarrow \texttt{ISPF}(\mathcal{T}_i, \mathcal{V}, \mathcal{K}_{j\rightarrow i}^d)$
    $\mathcal{F}_{j\rightarrow i}^d \Leftarrow N(d, \mathcal{T}_i^{\mathcal{K}_{j\rightarrow i}^d})$
return $\mathcal{F}_{j\rightarrow i}, \mathcal{B}_{i\rightarrow j}\ \forall j \in \mathcal{N}_i$

Algorithm 3 : $\mathbf{IASPF1}(i, f)$

$\tilde{\mathcal{T}}_i \Leftarrow \texttt{ISPF}(\mathcal{T}_i, \mathcal{V}, \{f\})$
for all $j \in \mathcal{N}_i$ *and* $j \in N(j, \mathcal{T}_i)$ **do**
  $\tilde{\mathcal{T}}_j \Leftarrow \texttt{ISPF}(\mathcal{T}_j, \mathcal{V}, \{f\})$
  $\mathcal{E}' \Leftarrow E(S(j, \tilde{\mathcal{T}}_i))$
  for all $u\rightarrow v \in \mathcal{E}'$ **do**
    $\mathcal{V}' \Leftarrow V(S(v, \tilde{\mathcal{T}}_i))$
    $\tilde{\mathcal{T}}_j^{u-v} = \texttt{ISPF}(\tilde{\mathcal{T}}_j, \mathcal{V}', \{u\text{–}v\})$
    for all $d \in \mathcal{V}'$ **do**
      if $i \in N(d, \tilde{\mathcal{T}}_j^{u-v})$ **then**
        $\mathcal{K}_{j\rightarrow i}^d \Leftarrow \mathcal{K}_{j\rightarrow i}^d \cup \{u\text{–}v\}$

return TABLES(i)

Algorithm 4 : $\mathbf{IASPF2}(i, f)$

$\tilde{\mathcal{T}}_i \Leftarrow \texttt{ISPF}(\mathcal{T}_i, \mathcal{V}, \{f\})$
for all $j \in \mathcal{N}_i$ *and* $j \in N(j, \mathcal{T}_i)$ **do**
  $\tilde{\mathcal{T}}_j \Leftarrow \texttt{ISPF}(\mathcal{T}_j, \mathcal{V}, \{f\})$
  $\mathcal{E}' \Leftarrow E(S(j, \tilde{\mathcal{T}}_i))$
  for all $u\rightarrow v \in \mathcal{E}'$ **do**
    $\mathcal{V}' \Leftarrow V(S(v, \tilde{\mathcal{T}}_i))$
    if $\nexists \mathcal{T}_j^{u-v}$ *or* $\mathcal{V}' \not\subseteq V(\mathcal{T}_j^{u-v})$ *or* $f \in E(\mathcal{T}_j^{u-v})$ **then**
      $\tilde{\mathcal{T}}_j^{u-v} \Leftarrow \texttt{ISPF}(\tilde{\mathcal{T}}_j, \mathcal{V}', \{u\text{–}v\})$
    else
      $\tilde{\mathcal{T}}_j^{u-v} \Leftarrow \mathcal{T}_j^{u-v}$
    for all $d \in \mathcal{V}'$ **do**
      if $i \in N(d, \tilde{\mathcal{T}}_j^{u-v})$ **then**
        $\mathcal{K}_{j\rightarrow i}^d \Leftarrow \mathcal{K}_{j\rightarrow i}^d \cup \{u\text{–}v\}$

return TABLES(i)

– Incremental SPF (ISPF) procedure can be used for efficiently figuring out the effect of a link failure. ISPF adjusts an existing shortest path tree instead of constructing it from scratch. The complexity of the ISPF is proportional to the number of nodes affected by the link failure which on the average is much smaller than $|\mathcal{V}|$.

The ASPF procedure based on the above observations is shown in Algorithm 1. It uses ISPF procedure (not shown here but can be found in [12]), for incrementally building a new SPT from an existing SPT. The arguments to ISPF include the tree $\mathcal{T}$ corresponding to the edge set $\mathcal{E}$, the set $\mathcal{E}'$ of (failed) edges to be removed and the set $\mathcal{V}'$ of interested destinations. It returns a new tree consisting of nodes in $\mathcal{V}'$ without the links in $\mathcal{E}'$. In ASPF procedure, the sets of key links are first initialized to $\emptyset$ (lines $1-3$). Then the shortest path tree $\mathcal{T}_i$ rooted at i is computed using SPF procedure (line 5). Each neighbor j that is a next hop to some destination is considered in turn (line 6). If not, the key links for the corresponding interface $j{\rightarrow}i$ remain $\emptyset$. Otherwise, j is the next hop to all the nodes in the subtree below j. Only the links $E(S(j,\mathcal{T}_i))$ in this subtree $S(j,\mathcal{T}_i)$ could be key links for the nodes $V(S(j,\mathcal{T}_i))$. So the search for key links is restricted only to $E(S(j,\mathcal{T}_i))$ (lines $8-9$). A SPT $\mathcal{T}_j^{u-v}$ without each of these edges u–v is incrementally computed using ISPF (line 11) from $\mathcal{T}_j$ which was computed earlier using SPF (line 7). These SPTs are partial trees computed to span only the affected nodes below u–v in tree $\mathcal{T}_i$ (lines $10-11$). Finally, a link u–v is included in $\mathcal{K}_{j\rightarrow i}^d$ for all d in $V(S(v,\mathcal{T}_i))$ if i is a next hop to d from j in tree $\mathcal{T}_j^{u-v}$ rooted at j without edge u–v (lines $12-14$).

Once key links are determined, forwarding and backwarding tables are computed using TABLES procedure shown in Algorithm 2. Since we are preparing the forwarding tables for handling single link failures, the backwarding table for an interface $i \rightarrow j$ contains the next hops without only the edge $i{-}j$. These entries are obtained using ISPF on $\mathcal{T}_i$ (lines $2-4$). The forwarding table entry for destination of $j{\rightarrow}i$ interface is computed by excluding the links in the set $\mathcal{K}_{j\rightarrow i}^d$. A tree $\mathcal{T}_i^{\mathcal{K}_{j\rightarrow i}^d}$ corresponding to key link set $\mathcal{K}_{j\rightarrow i}^d$ is computed only if it wasn't previously computed (lines $5-6$). In particular, when the key link set is empty, existing tree $\mathcal{T}_i$ can be reused. Essentially, a shortest path tree is computed only once for each distinct set of key links.

We now analyze the complexity of the ASPF procedure . There are $|\mathcal{N}|+1$ invocations of SPF (lines 5 and 7) and $O(|\mathcal{V}|)$ of ISPF invocations (line 11). The running time of an incremental algorithm such as ISPF depends on the number of nodes affected (requiring recomputation of paths) by the changes in the edge set. So let us measure the complexity in terms of the affected nodes. Each link e in the tree $\mathcal{T}_i$ is pulled down in turn to see its impact on the next hops from a neighbor j. Only those nodes that are below the link e are affected by the removal of e. A node is affected by the removal of any of the links along the path to it from the root. The number of link removals (the ISPF computations) affecting a node in the worst case would be the diameter of the network D. So the total number of affected nodes due to $O(|\mathcal{V}|)$ ISPF invocations would be $O(D|\mathcal{V}|)$. Since regular SPF computation has to start from scratch, we can say that the affected nodes are $O(|\mathcal{V}|)$. So the complexity of key link computation is then $O(D+|\mathcal{N}|+1)$ times regular SPF computation. The time taken by TABLES depends on the sets of key links and it is found to be dominated by the time for key link

computation. Therefore, considering that D can be approximated by $\log|\mathcal{V}|$ and SPF takes $O(|\mathcal{E}|\log|\mathcal{V}|)$, the complexity of ASPF is $O(|\mathcal{E}|\log^2|\mathcal{V}|)$.

3.2 Incremental ASPF Algorithms

The ASPF procedure described above computes forwarding tables efficiently and thus makes the deployment of FIR feasible. Its running time can be further improved by saving the intermediate steps of the previous computation of these tables (corresponding to the previous global update) instead of obtaining them from scratch. We devised two incremental versions IASPF1 and IASPF2 that take advantage of the saved information in determining new key links and tables when an update is received notifying the failure of a link. These two versions differ in the amount of memory usage. IASPF1 remembers $\mathcal{T}_i$ rooted at i, $\mathcal{T}_j$ and $\mathcal{T}_i^{i-j}$ for each neighbor j. So the total space required for IASPF1 is $O((2|\mathcal{N}|+1)|\mathcal{V}|)$. In addition to this, IASPF2 saves partial trees $\mathcal{T}_j^{u-v}$ for each edge u–v in $\mathcal{T}_i$. The additional space required for IASPF2 is $O(D^2|\mathcal{V}|)$.

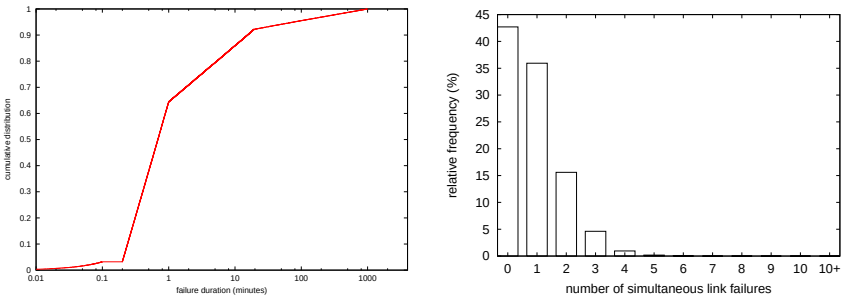
The procedure IASPF1 shown in Algorithm 3 is quite similar to ASPF with changes only in lines 5 and 7 (renumbered 1 and 3 respectively in IASPF1). Suppose the failed link is f. While ASPF uses SPF (line 5), IASPF1 invokes ISPF to compute new $\tilde{\mathcal{T}}_i$ without link f based on the saved old $\mathcal{T}_i$ (line 1). Similarly $\tilde{\mathcal{T}}_j$ is computed for each j using old $\mathcal{T}_j$ (line 3). The backwarding table computation time can also be improved by using the saved $\mathcal{T}_i^{i-j}$. The rest of the IASPF1 procedure is no different from ASPF. With only minor changes, using $O((2|\mathcal{N}|+1)|\mathcal{V}|)$ space, IASPF1 reduces approximately $|\mathcal{N}|+2$ SPF computations. These procedures are shown only for a link down event. A link up event can also be treated analogously.

The IASPF2 procedure shown in Algorithm 4 further improves the running time by avoiding unnecessary computations of the partial trees $\mathcal{T}_j^{u-v}$ for each edge u–v in $\mathcal{T}_i$. This procedure is similar to IASPF1 except for lines $7-10$. A tree $\mathcal{T}_j^{u-v}$ is reused if it exists and spans all the nodes affected when u–v is down without including the failed link f. Otherwise, a new such tree is constructed by invoking IASPF. Since these trees are partial trees and a link is not part of many such trees, a large fraction of IASPF invocations can be avoided. In the next section, we show that the average running time of IASPF2 is less than even a single SPF computation. Now let us look at the additional space required for storing these partial trees. As mentioned earlier, a node is affected by all the links along its path from the root and their count in the worst case would be the network diameter D. So a node would be a member of at most D partial trees. The space needed for a partial tree in the worst case would be D times the number of affected nodes in it. So the total space for all the partial trees put together would be less than $D^2|\mathcal{V}|$ which is only linear in terms of the number of nodes in the network.

4 Evaluation of the FIR scheme

We now evaluate the performance of the FIR scheme and demonstrate its failure resiliency and forwarding efficiency. We first describe how link failures in random topologies are modeled. Then, we show how service downtime is reduced substantially by employing FIR. It is also shown that compared to the optimal shortest path routing

the extent of path elongation due to local rerouting by FIR is not significant. Finally, the relative computational complexity of ASPF and incremental ASPF algorithms w.r.t. Dijkstra's SPF algorithm is presented to affirm that FIR is viable.



(a) failure duration (b) simultaneous failures

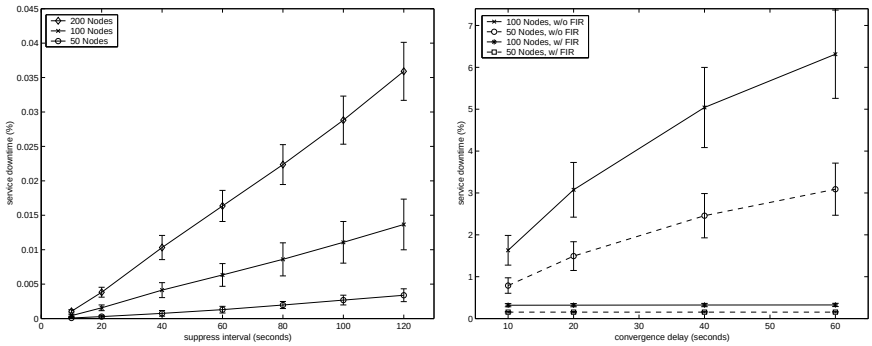
Fig. 6. Distribution of failures

4.1 Link Failure Model

The pattern of link failures in large operational networks is yet to be characterized very well. In [7], some detailed measurements and analysis on the link failure events in the Sprint's IP backbone network are reported. They presented a histogram of the mean time between failure of links and the cumulative distribution of failure durations. Their findings are used in this paper as the basis for inducing failures on random topologies generated using the BRITE topology generator [9] with link weights chosen randomly from the range 100 to 300. We modeled the mean time between failures (MTBF) of links with a heavy tailed distribution, with the distribution function obtained by curve fitting on the histogram reported in [7]. The MTBF values generated in this way vary from several hours to tens of days. Our model of failure events duration was based on the cumulative distribution reported in [7]. We partitioned that distribution function into several segments and use straight lines to approximate each segment as shown in Figure 6(a). Histograms on the relative frequency of the number of simultaneous failures is shown in Figure 6(b) for 50 node topology with average degree 4.

4.2 Service Downtime

We now compare the routing performance with and without employing FIR. The performance is measured in terms of *service downtime* which is defined as the total time any two nodes in the network are unreachable from each other. First consider the performance with FIR. When a router under FIR detects an adjacent link failure, it does



(a) with FIR due to multiple failures (b) with and without FIR

Fig. 7. Performance evaluation in terms of service downtime

not propagate the LSP immediately. Instead it *suppresses* the global update and initiates local rerouting. There would not be any delay between failure detection and local rerouting if backwarding tables are employed in the forwarding plane. However, local rerouting by different nodes due to multiple suppressed failures can result in a forwarding loop contributing to service downtime. For example, suppose the links $2{-}5$ and $4{-}6$ of the topology in Figure 1 are down. Then packets from 1 to 6 take the path $1{\rightarrow}2{\rightarrow}1{\rightarrow}4\ {\rightarrow}1\ {\rightarrow}2{\rightarrow}1\cdots$, thus keep looping even though 6 is reachable through $1{\rightarrow}3{\rightarrow}5{\rightarrow}6$. Nevertheless, since failures are suppressed only for a certain *suppress interval*, it is less likely that multiple links fail simultaneously within a short duration. Moreover, only a specific scenario of failures of links along the shortest path and the alternate path can cause looping.

To demonstrate the ability of FIR in handling simultaneous failures, the downtime with FIR is plotted as a function of the suppress interval in Figure 7(a). The results are shown for network topologies of different size (50, 100, and 200 nodes) and average degree of 4. Every point in the plot is the average of 5 simulation runs, with the vertical bars reporting 95% confidence intervals. When the suppress interval is 60 seconds, the fraction of the time some destination is unreachable due to loop-causing simultaneous multiple suppressed failures is less than 0.02%. Even when the suppress interval is made 2 minutes to further reduce the global link state update overhead, all nodes are reachable 99.95% of the time. These results suggest that by preparing for single link failures, FIR can also handle most of the simultaneous link failures.

The discussion above assumed that local rerouting does not incur any delay. But when the backwarding tables are not employed in the forwarding plane there would be some delay in locally sensing the failure, recomputing the forwarding tables and updating FIBs. The time to detect a link failure would be much shorter with local rerouting than with global rerouting. For example, a link can be considered failed and local rerouting is triggered with the loss of single hello packet, while the failure event is notified globally only after the loss of 5 hello packets. Essentially, local rerouting enables

swift response to failures without causing routing instability. Using the backwarding tables stored in the control plane, the forwarding tables can be recomputed in negligible amount of time. Then, the time to update FIBs depends on the number of entries changed. Assuming that the total local rerouting delay is 2 seconds, the service downtime with FIR is contrasted with downtime without FIR in Figure 7(b).

Let us look at the downtime without FIR. Suppose a link fails at time t and after a period T all routers reconverge and forwarding to the affected destinations is resumed. We refer to this time T as the convergence delay which is the sum of all the delays due to several contributing factors such as *lsp-generation* interval, and *spf-interval* as explained in [7]. During this period certain node pairs that have shortest paths through the failed link are not reachable. Figure 7(b) shows the service downtime without FIR as a function of the convergence delay. It also shows the downtime with FIR assuming local rerouting delay of 2 seconds and suppress interval of 1 minute. It is clear that by employing FIR, service downtime can be improved by at least an order of magnitude. In addition, by suppressing the update of failures that last less than a minute, majority of the failures are handled without global updating and recomputation. These results indicate that FIR not only increases failure resiliency but also ensures routing stability while reducing update overhead.

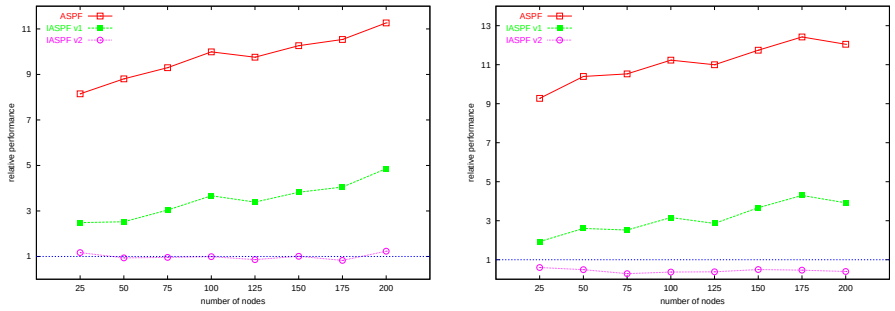
4.3 Path Length Stretch

Under FIR, only the node adjacent to a failed link is aware of the failure and all other nodes are not. So, a packet takes the usual shortest path till the point of failure and then gets rerouted along the alternate path. Consequently, in the presence of link failures, FIR may forward packets along longer paths compared to the globally recomputed optimal paths based on the link state updates. For example in the topology of Figure 1, when the link 2–5 is down, packets from 1 to 6 are forwarded along the path 1→2→1→4 →6. Had node 1 been made aware of the link failure, packets would be forwarded along the shorter path $1 \rightarrow 3 \rightarrow 5 \rightarrow 6$. However, we found that on realistic large topologies the extent of this elongation is not significant. Let *stretch* of a path between a pair of nodes be the ratio of the lengths of the path under FIR and the optimal shortest path. When the weights of all the links are not same, path length is said to be the sum of the weights of its links. Without any link failures, there is no difference between the FIR paths and the optimal shortest paths. So the stretch is 1. We have measured the stretch under link failures due to FIR for random topologies of various sizes. Across all topologies the average stretch is less than 1.2 and in most cases it is close to 1.

4.4 Forwarding Table Computation Complexity

As explained before, the main change required in the control plane for the deployment of FIR is the replacement of traditional interface independent routing table computation algorithm with an algorithm for computing interface dependent forwarding tables. This algorithm is invoked only when a link failure lasts longer than a suppress interval and a global update is triggered. This computation is done while packets to the affected destinations are locally rerouted. Therefore, unlike in the existing routing schemes, the running time of the FIR algorithms does not affect the reachability of destinations.

Nevertheless, it is desirable to reduce the computational overhead on a router. Here we evaluate the running time of the FIR algorithms and show that the forwarding tables can be incrementally computed in less than a SPF computation time.



(a) average degree of 4 (b) average degree of 6

Fig. 8. Comparison of run time complexity of FIR algorithms

We measured the time complexity of all these SPF based algorithms in terms of the number of distance comparisons made as was done in [10]. The distances of two nodes are compared for updating distance of one of them or for readjusting the priority queue after an `extract` or `enque` operation. The running time of ASPF and its incremental versions `IASPF1` and `IASPF2` are shown in Figure 8. We show the relative performance of these algorithms w.r.t. well known Dijkstra's SPF algorithm. Since Dijkstra's algorithm is widely deployed, using it as a reference helps in assessing the running time of these algorithms. The memoryless ASPF procedure takes around 10 times longer than SPF for computing forwarding tables from scratch. The incremental procedure `IASPF1` remembers $2|\mathcal{N}| + 1$ shortest path trees and improves the running time to less than 5 times SPF. Using an additional space of less than $D^2|\mathcal{V}|$, `IASPF2` takes no more than a single SPF computation. Its relative performance gets better as the connectedness increases. Apart from the modest space requirement, `IASPF2` does not add any additional processing burden on routers that currently employ Dijkstra's SPF algorithm for computing routes.

These results establish that FIR is feasible, reliable, and stable. Furthermore, it requires minimal changes in control plane only and also reduces communication overhead. These features make FIR an attractive alternative to the existing routing schemes.

5 Related Work

The nature of link failures in a network and their impact on the traffic has received a great deal of attention recently. The frequency and the duration of link failures in a backbone network has been studied and reported in [5, 7]. They observe that link failures are

part of everyday operation of a network due to various causes such as maintenance, accidental fiber cuts, and misconfigurations. It is also found that the majority of the failures are transient lasting less than a minute warranting local rerouting. The impact of link failures on Voice-over-IP is assessed in [4]. They noticed that link failures may be followed by routing instabilities that last for tens of minutes resulting in the loss of reachability of large sets of end hosts. Since the level of congestion in a backbone is almost negligible, offering high availability of service is identified as the major concern for VoIP. These findings about the link failures and their debilitating effect on the network services provide a strong motivation for schemes such as FIR that focus on ensuring service continuity.

There have been several proposals for mitigating the impact of link failures on network performance. [6] and [13] address the issue of assigning weights to links such that the traffic is balanced across the network even in the presence of link failures. These schemes can be thought of as preparing for link failures in terms of reducing overload while FIR is concerned with increasing availability. As mentioned earlier, guaranteeing reachability is found to be an overriding concern than avoiding congestion in a backbone network. Moreover, these schemes can be used in conjunction with FIR. A detailed analysis of the sources of delay in routing reconvergence after a link failure is provided in [1, 2]. They suggest tuning various parameters related to link state propagation and routing table computation for accelerating the convergence and reducing the downtime. This may not be the best recipe for handling common transient link failures. The objective of FIR is to make forwarding insensitive to the parameter values chosen for accelerating convergence and insuring stability.

A recent work closely related to FIR is the deflection routing proposed in [8]. The basic idea underlying their approach is to select a next hop node based on strictly decreasing cost criterion. While deflection routing guarantees loop-free paths, it may not always find such a path even if one exists. For example, in a simple triangle topology when a link with the smallest cost goes down, the corresponding pair of nodes are not reachable. Apart from this last hop problem, deflecting routing requires that the weights of links satisfy a certain condition. FIR imposes no such restrictions on weight assignment and assures loop-free forwarding to any reachable destinations in case of single link failures. An algorithm proposed in [11] performs local restoration by informing only the routers in the neighborhood about link failure events instead of all routers. FIR achieves similar effect without requiring any changes to link state propagation mechanism. An application layer solution is proposed in [3] for detecting and recovering from path outages using a resilient overlay network. While RON is an attempt to overcome the slow convergence of BGP based inter-domain routing, FIR is a remedy for outages in intra-domain routing. Nevertheless, we believe network layer schemes such as FIR obviate the need for application layer approaches like RON.

6 Conclusions and Future Work

In this paper, we addressed the problem of ensuring destination reachability in the presence of link failures. We proposed a *failure insensitive routing* approach where routers infer link failures from the packet's flight and precompute interface specific forward-

ing tables avoiding the potentially failed links. When a link fails, only adjacent nodes locally reroute packets while all other nodes simply forward them according to their precomputed interface specific forwarding tables without being explicitly aware of the failure. We presented an *available shortest path first* algorithm that computes interface specific forwarding tables for dealing with single link failures in $O(|\mathcal{E}| \log^2 |\mathcal{V}|)$ time. We have also described an incremental ASPF algorithm that requires $O(D^2|\mathcal{V}|)$ space for remembering intermediate steps of the previous computation but runs in less time than a SPF computation. We have demonstrated that FIR handles simultaneous multiple failures also and reduces service downtime by an order of magnitude. Essentially FIR approach improves failure resiliency without jeopardizing routing stability. It does so without altering the forwarding plane while reducing communication overhead. Hence, we believe that FIR is an attractive alternative to the existing routing schemes. We are currently in the process of conducting packet level simulations to assess the utility of FIR in terms of throughput received by TCP flows and quality experienced by VoIP flows. Also, we plan to actually implement FIR and evaluate its performance to make its case more compelling.

References

1. C. Alattinoglu, V. Jacobson, and H. Yu, "Towards Milli-Second IGP Convergence," draft-alaettinoglu-ISIS-convergence-00.txt, November 2000.
2. C. Alattinoglu, and S. Casner, "ISIS routing on the Qwest backbone: A recipe for subsecond ISIS convergence," NANOG 24, 2/2002.
3. D. Anderson, H. Balakrishnan, F. Kaashoek, and R. Morris, "Resilient Overlay Networks," SOSP, 2001.
4. C. Boutremans, G. Iannaccone, and C. Diot, "Impact of Link Failures on VoIP Performance," NOSSDAV, 2002.
5. C.-N. Chuah, S. Bhattacharyya, G. Iannaccone, C. Diot, "Studying failures & their impact on traffic within a tier-1 IP backbone", CCW, 2002.
6. B. Fortz, "Optimizing OSPF/IS-IS weights in a changing world", IEEE JSAC Special Issue on Advances in Fundamentals of Network Management, Spring 2002.
7. G. Iannaccone, C.-N. Chuah, R. Mortier, S. Bhattacharyya, C. Diot, "Analysis of link failures in an IP backbone", IMW 2002.
8. S. Iyer, S. Bhattacharyya, N. Taft, N. McKeown, and C. Diot, "An approach to alleviate link overload as observed on an IP backbone," INFOCOM, 2003.
9. A. Medina, A. Lakhina, I. Matta, and J. Byers, "BRITE: An Approach to Universal Topology Generation", Proceedings of MASCOTS 2001, Cincinnati, August 2001.
10. P. Narvaez, "Routing reconfiguration in IP networks", Ph.D. Dissertation, MIT, June 2000.
11. P. Narvaez, K.-Y. Siu, and H.-Y. Tzeng, "Local Restoration Algorithms for Link-State Routing Protocols", ICCCN, 1999.
12. S. Nelakuditi, S. Lee, Y. Yu, and Z.-L. Zhang, "Failure Insensitive Routing for Ensuring Service Availability," Technical Report, University of South Carolina, Columbia, February 2003.
13. A. Nucci, B. Schroeder, S. Bhattachrayya, N. Taft, C. Diot, "IS-IS link weight assignment for transient link failures," SPRINT ATL Technical Report TR02-ATL-071000.